Daniel Oliveros – A02093272

CS 5460

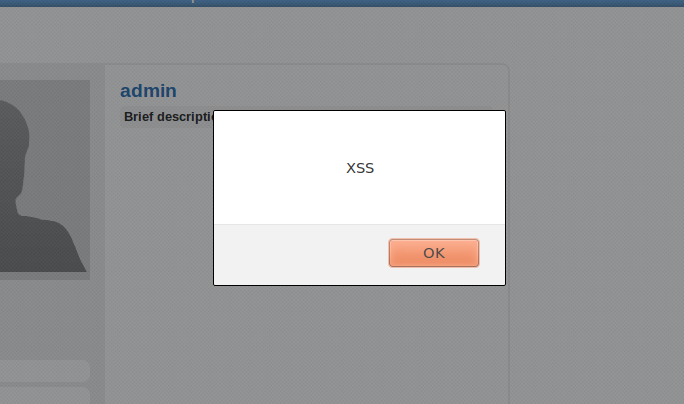
**Assignment 6 – XSS Attack Lab**

**3.1. Task 1:**

I logged in using the Admin login information. After this, I proceeded to write the command: <script>alert(’XSS’);</script> into the field titled *brief description*.

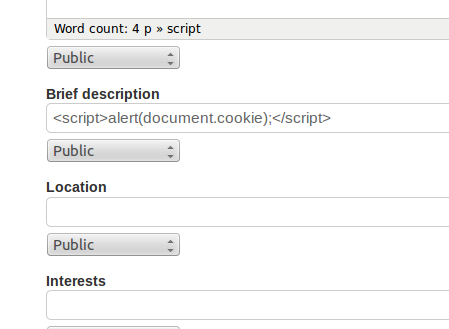


As soon as I inserted the correct code (single quotes are evil), this popup occurred:

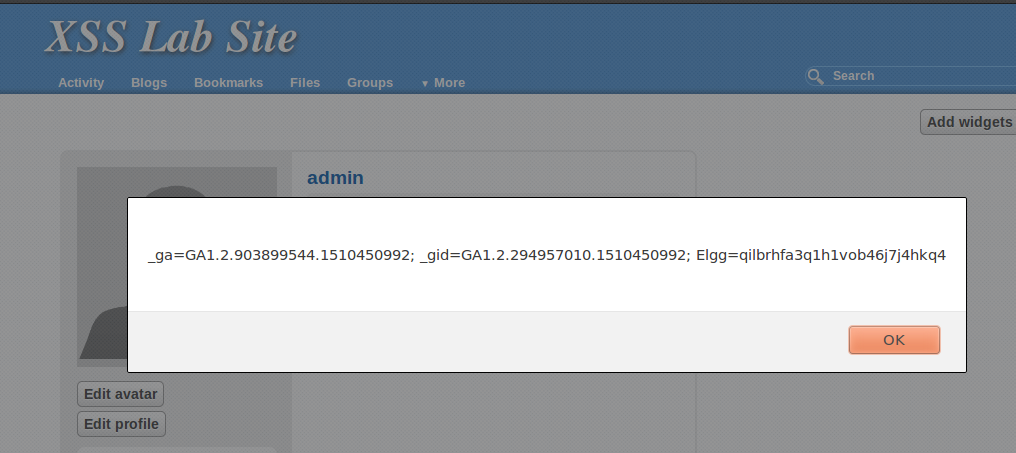


**3.2. Task 2:**

I changed the information in the field to this:



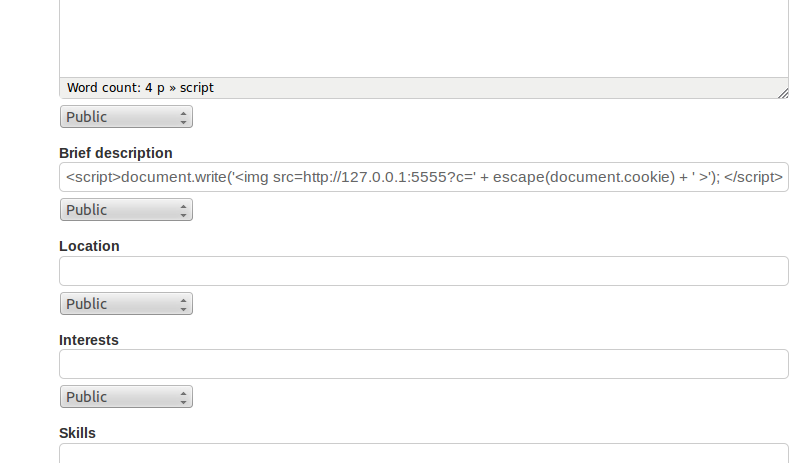
Which, as soon as I pressed save, gave me this lovely popup:



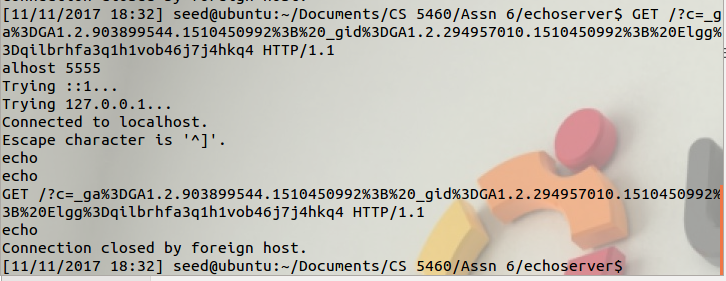
This popup displayed various pieces of cookie-related information, which could be used for evil.

**3.3. Task 3:**

I started by compiling and executing the file echoserv.c. From then, I proceeded to change the information in my brief description field to:

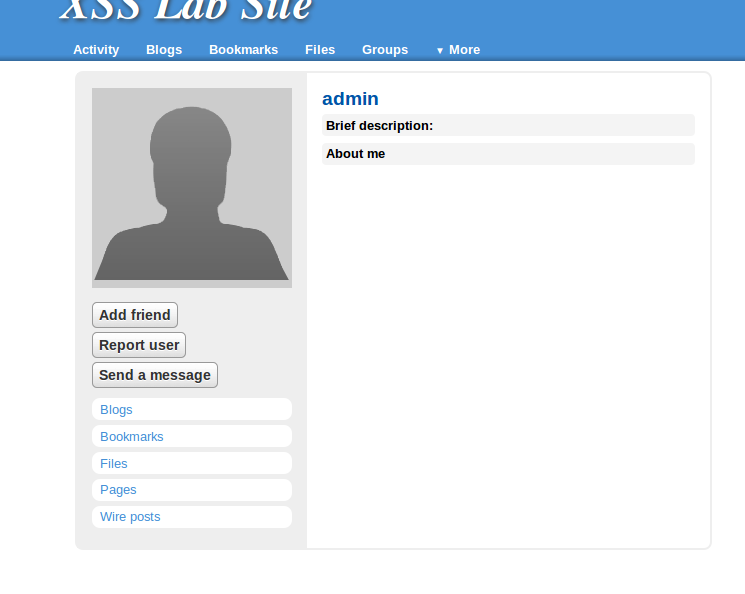


Which, after fixing the quotation symbols issue, sent out the cookie to my machine’s 5555 port. These were my results:



**3.4. Task 4:**

We added the cookie stealing script into our brief description field. This allows us to get the user’s cookie, the current time stamp, and their token into our command line. From here, we hardcoded the cookie, the timestamp, and the token. This is how the profile page looks like to someone visiting it:



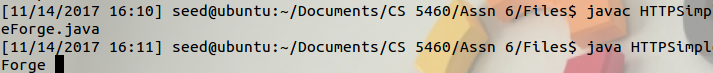
The user can’t tell there’s a script running. However, their visit sends me their cookie:



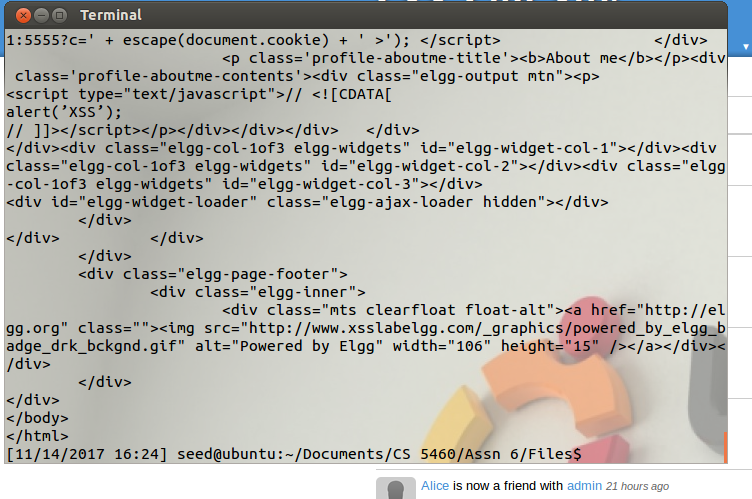
I hard coded this information into my java program, which will send a friend request to my user. Here is the program:



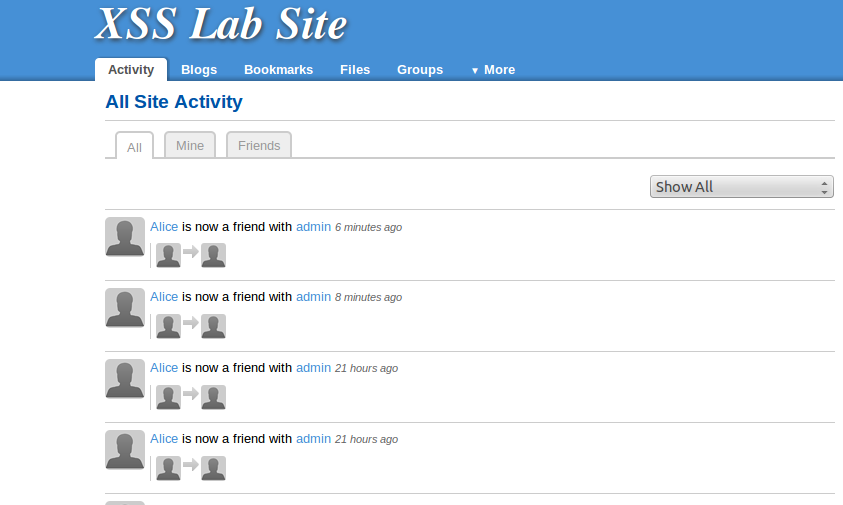
Here is how I compiled and ran the program:



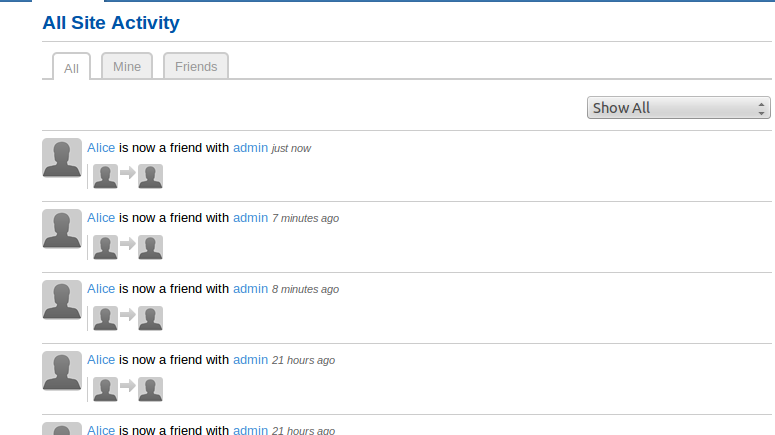
This is the output from the program, not necessarily all that interesting



However, some interesting developments have happened. This is what the activity tab looked like before:



And this is what it looks like now:



As we can tell, our program has helped us hijack Alice’s cookies and use them to add admin. Here is how Admin’s profile looks like from Alice’s view:



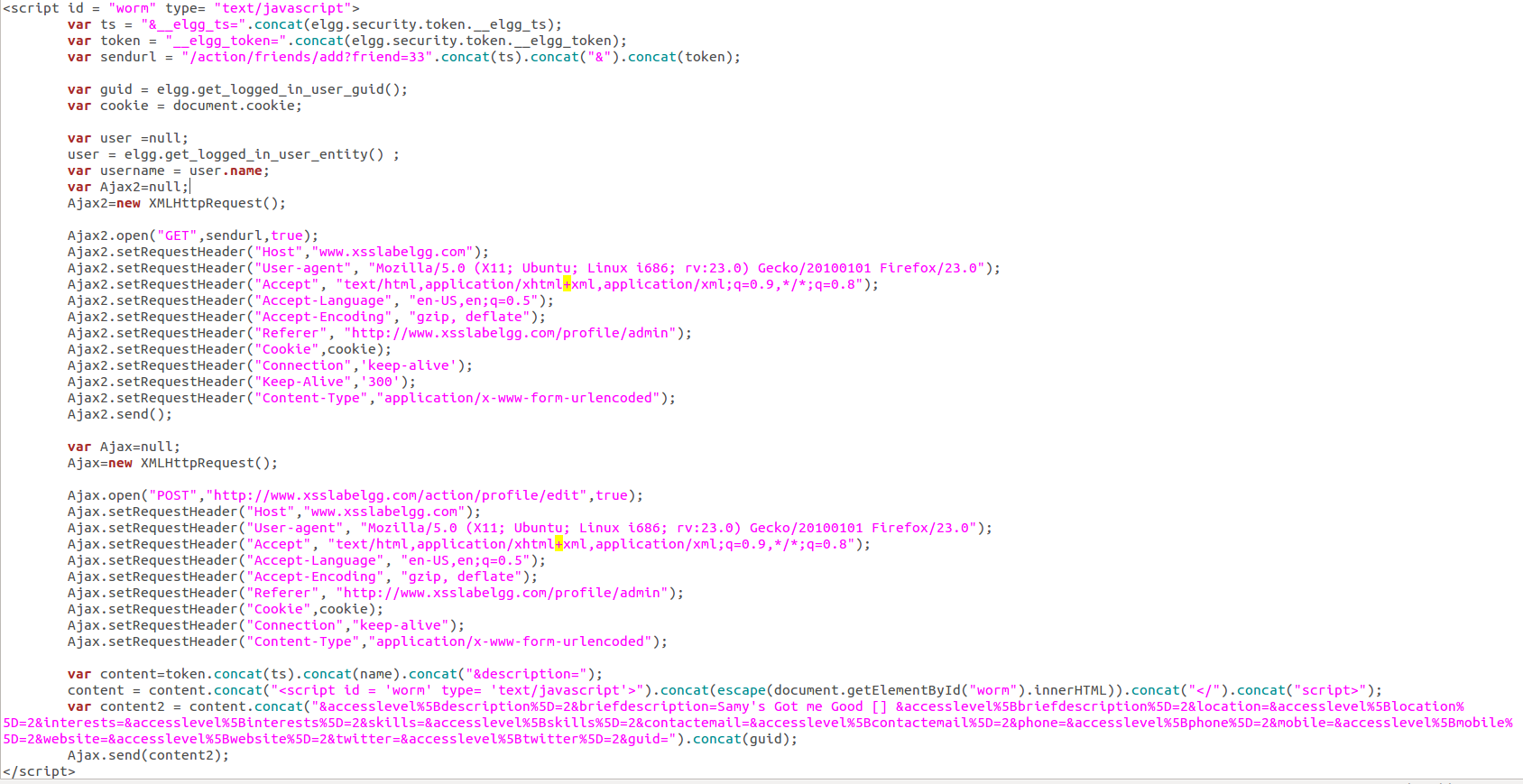
**3.5. Task 5:**

After giving it enough though, I realized this task cannot be done on its own. It is impossible to add the code that changes the description on its own. This issue happens because as soon as we save the script into our page the post request that is sent out deletes it. So, the solution for this task is now task 6.

**3.6. Task 6:**

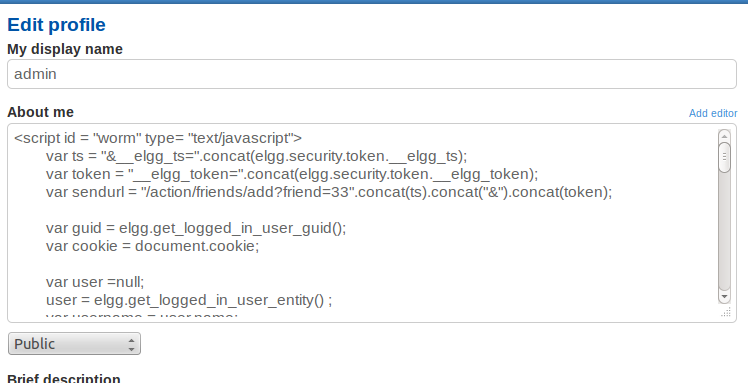
I didn’t keep progress of everything we had to do to make this work. Working through this took over 10 hours. In the end, we got our script to propagate using the ID approach (hooray for extra points!).

Anyway, here is what my script ended up looking like after all the changes it went through:

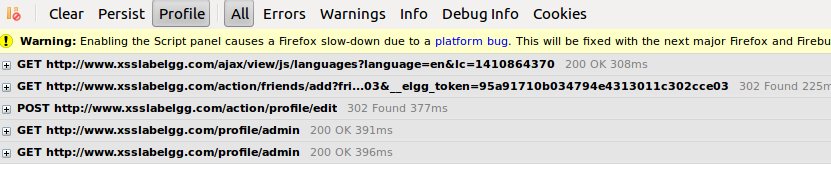


A couple of important things to note about this script are that it uses separate XMLHttpRequest objects, this is because when I used one the different requests would abort each other. Another important part of this is the use of the escape command, this command is needed for the formatted content to make sense to the post request. Also, various variables within this code were obtained using some of the protocols we found within the elgg documentation, these were used for tokens, guid, and the user’s name. Finally, the use of the concatenate function was essential for the request, this was due to escape messing up “+” signs.

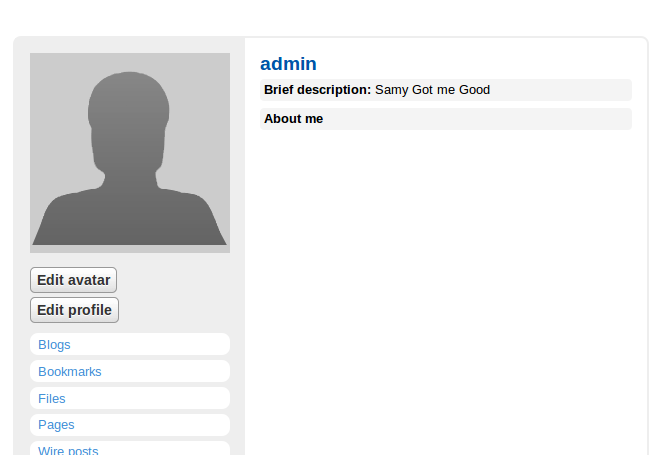
This code was added to the “About me” section of the user’s profile, every other section has size restrictions, so using this section was pretty much necessary. Here is how the script looks like in the form:



When saved, these are the javascript pieces of code that are triggered:

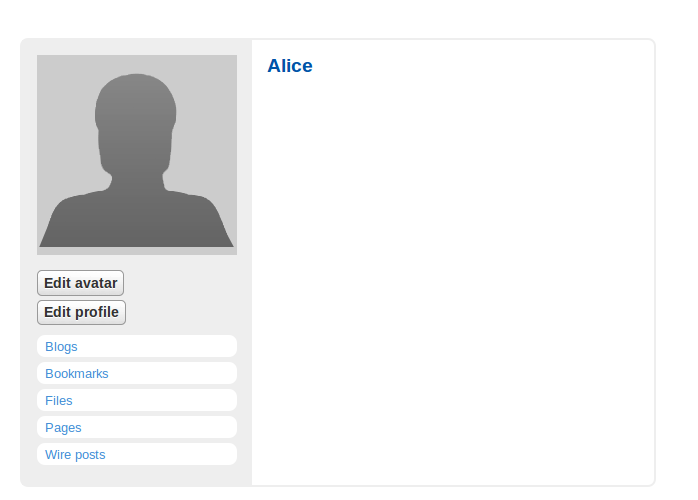


The GET request that adds a friend and the POST request that edits the profile are the ones our script is running. Reloading the page shows the changes made through the POST request:

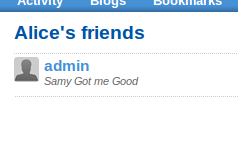


From here, any user that looks at the profile is pretty much screwed. The script is copied onto their page with the brief description “Samy Got me Good”, they also will add Samy, even if they view a profile different than his.

Here is a simple demonstration of this process. We log in as the user Alice:

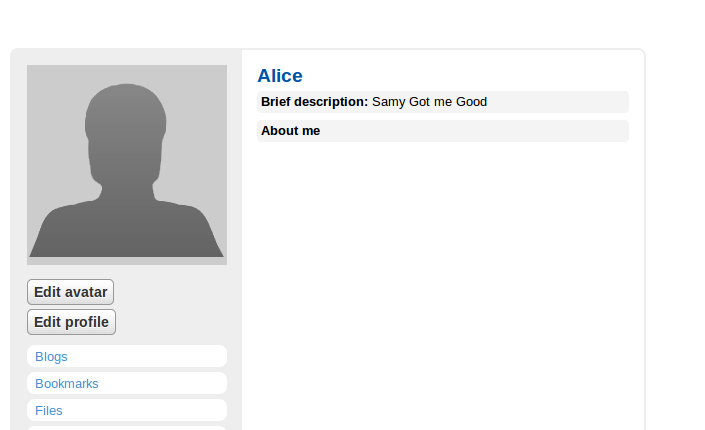


Alice’s profile is empty, for now. We have added her as a friend on the infected user’s end. If Alice decides to go and check who our infected user is, she will only see this:

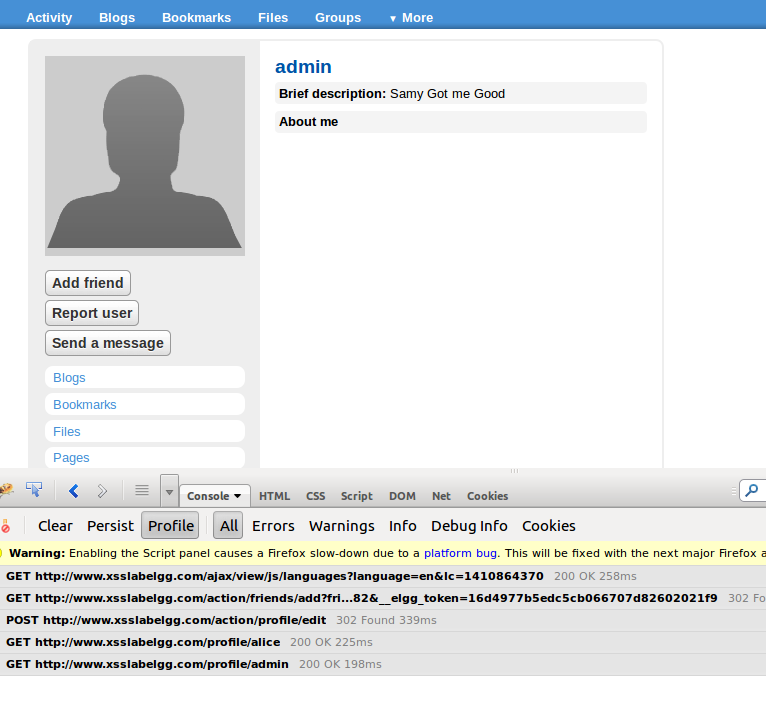




Nothing too flashy to show, however, Alice has already been infected. If we look back on her profile:

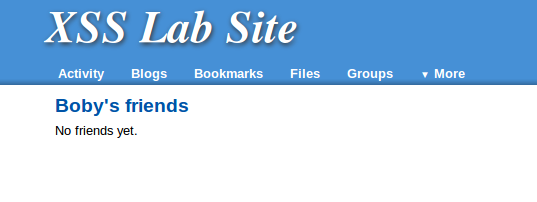


We can see she’s infected. She also cannot delete the infected user from her friend list. This is because whenever she tries to unfriend him:

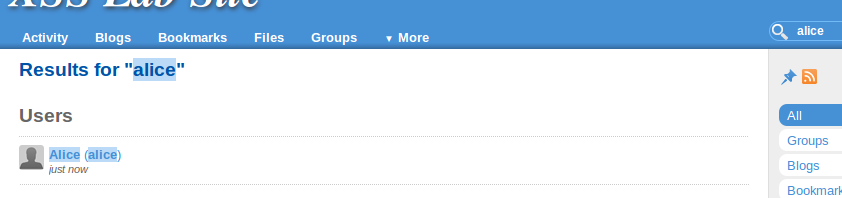


The scripts on the bottom are sent again, adding the infected user as her friend, and adding the malicious code onto her page as well.

Any users who are unaware that Alice is infected and visit her page will be infected as well. Take for example Boby. Boby has no friends:



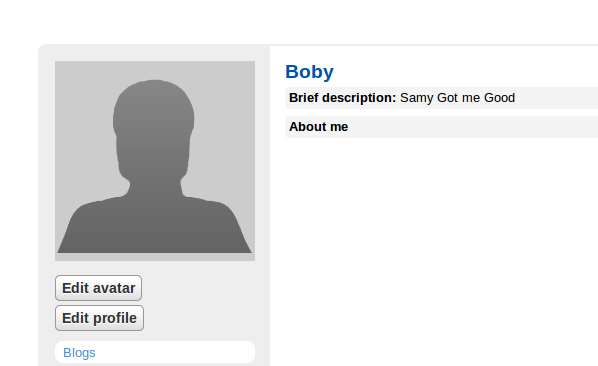
Boby is looking for friends and comes across Alice:



He looks at her profile, he figures “Hey, this person seems cool”, so he adds her, ignoring the subtle hints that some Samy guy may be involved in all this.



Boby goes back to his own page. He decides he should get a profile picture or something to make himself look more interesting:



But what’s this? Who is this Samy guy, and how did he get into Boby’s page? Confused, Boby looks at his friend list:

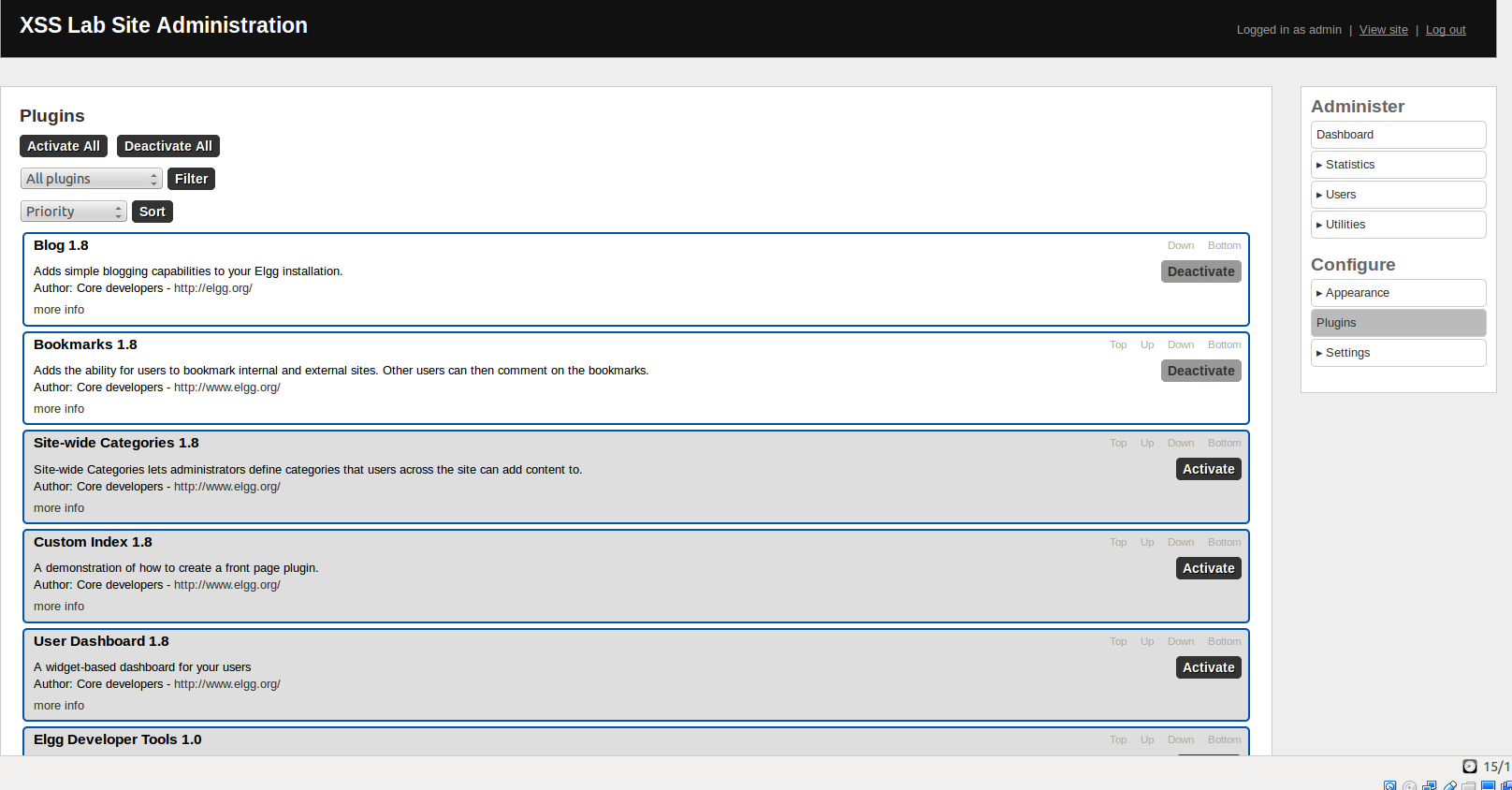


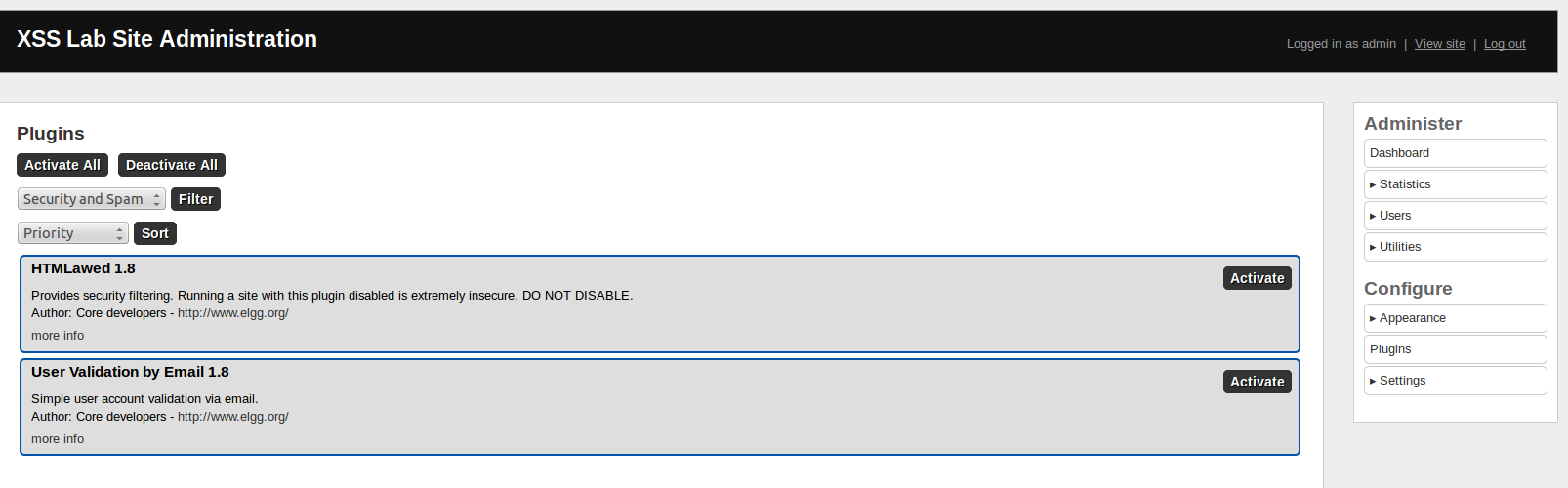
And he sees he’s not only friends with Alice, but he’s now friends with someone named admin. Admin is the original infected profile, and as soon as Boby looked at Alice’s page he added him and became infected as well.

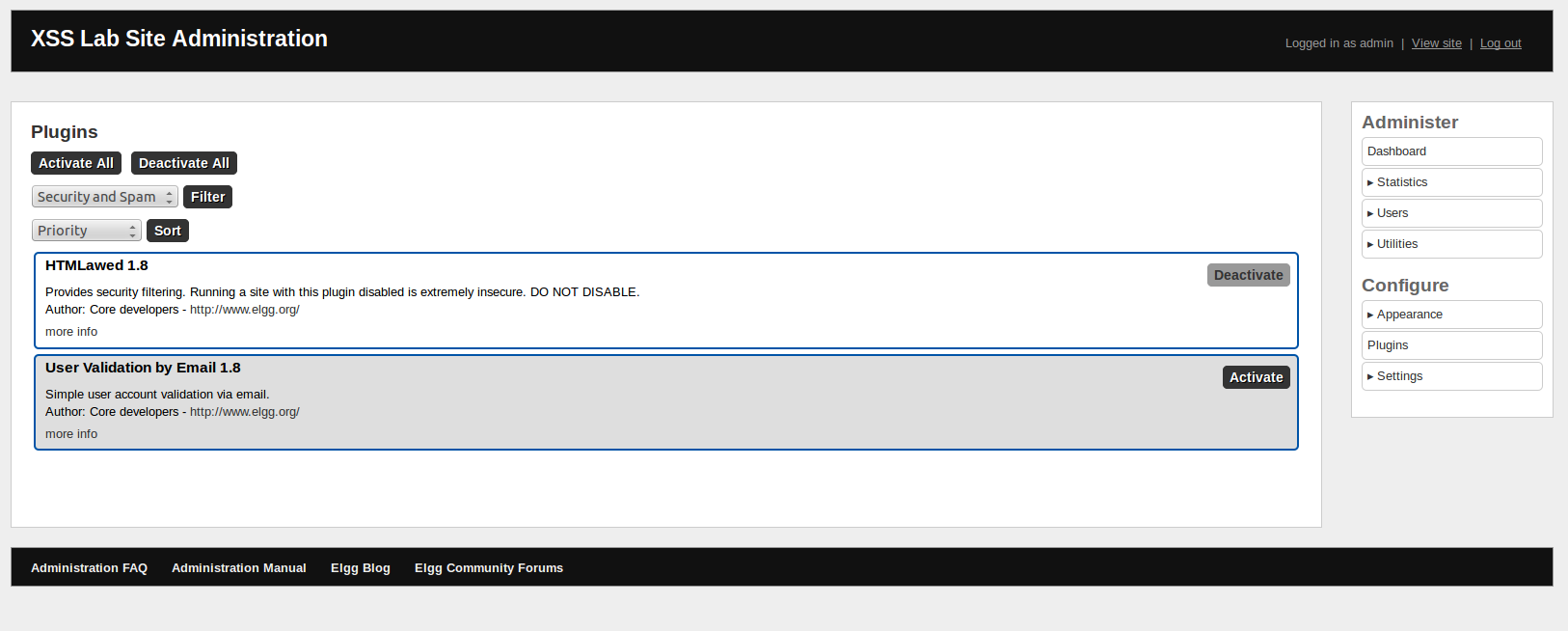
Now anyone who looks at Boby’s profile will follow the same fate, until our patient zero has friended everyone in the network. Or someone finds a fix.

**3.7. Task 7: Someone finds a fix, or two**

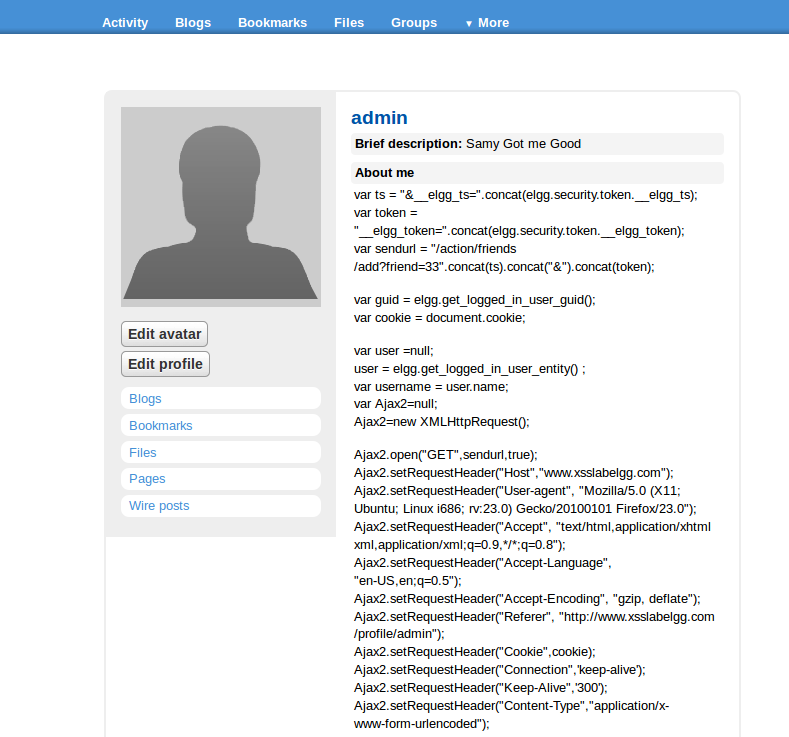
We got in the location requested, and proceeded to look for the given plugin to activate it:





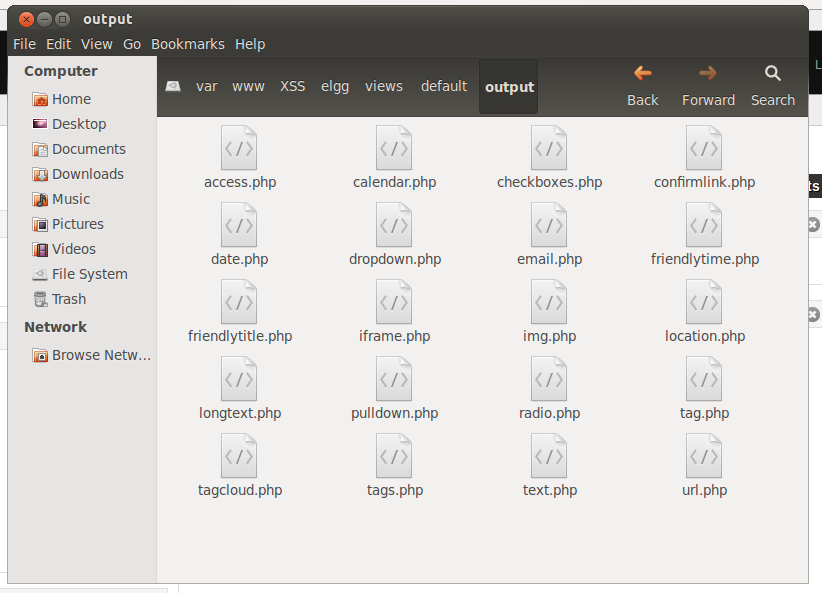


From here, we looked at a profile and this is what we found:

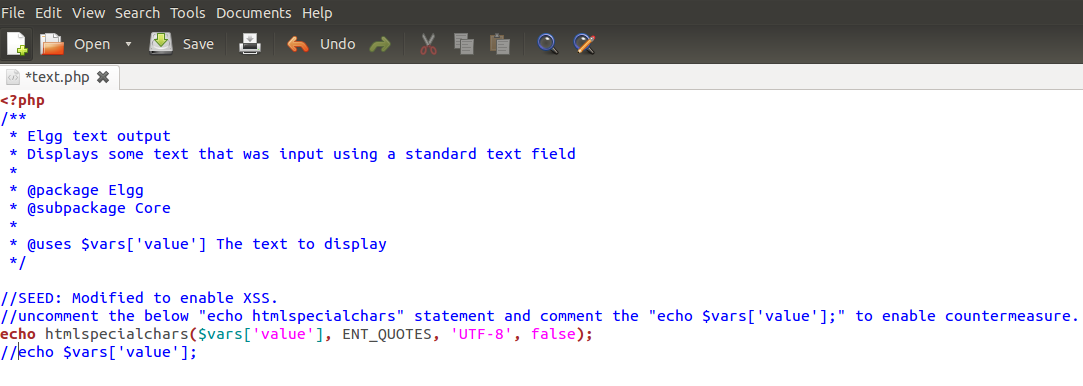


The javascript code has become useless. It won’t execute anything and it is clearly visible to anyone who visits this page.

From here, we proceeded to re-enable the second countermeasure. We got to this folder:



And changed everything back to the original way it was set up.



When looking at an infected user’s page, however, there didn’t seem to be any major changes. It seems perhaps the first fix was enough to take care of the problem itself.

