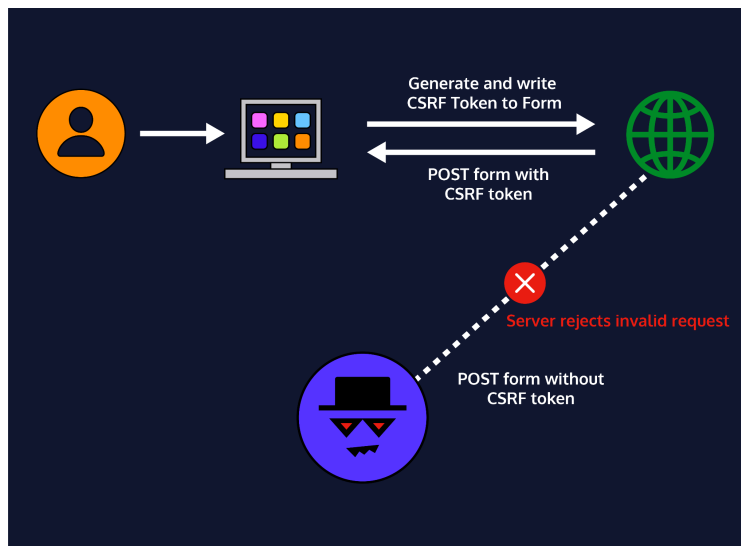


# Cross-Site Request Forgery (CSRF) Attack Lab

Submitted to:  
Dr. Md. Shariful Islam  
Professor, Institute of Information Technology(IIT)  
University of Dhaka

Submitted by:  
Md Arif Hasan  
BSSE-1112



Submission date: 19.09.2022



Institute of Information Technology

### 3.2)

#### Task 2: CSRF Attack using GET Request

This is a Cross-site request forgery attack where GET request is used to add Samy to Alice's friend list. So firstly, Samy wants to become a friend to Alice but Alice refuses to add him to her Elgg friend list. Samy decides to use the CSRF attack to achieve his goal. For doing this, he adds someone to his friend list and captures that request using LiveHTTPHeaders. So, based on that, he recreates a URL that adds himself to the friend list of Alice.

So, we create an index.html file on the /var/www/CSRF/Attacker and paste the following HTML code on that:

```
<html>
  <head>
    <title> Testing </title>
  </head>
  <body>
    <h1> Win a ticket !</h1>
    
  </body>
</html>
```

Figure-1: Html code

```
[09/12/22]seed@VM:~$ cd /var/
[09/12/22]seed@VM:/var$ cd /www
bash: cd: /www: No such file or directory
[09/12/22]seed@VM:/var$ cd www/
[09/12/22]seed@VM:../www$ ls
CSRF  html  RepackagingAttack  SQLInjection  XSS
[09/12/22]seed@VM:../www$ cd CSRF/
[09/12/22]seed@VM:../CSRF$ cd Attacker/
[09/12/22]seed@VM:../Attacker$ ls
[09/12/22]seed@VM:../Attacker$ gedit index.html
[09/12/22]seed@VM:../Attacker$ sudo gedit index.html
```

Figure-2: Location of this html file

It was created by Samy. Samy has to find his own id to add himself to Alice's friend list. He goes to the member's page, inspects the elements using firefox, and finds his own id. The id of Samy is: 43. Secondly, he constructs this malicious URL so that he can generate the GET request that adds him to Alice's friend list. The malicious website link to become a friend of victim:

<http://www.csrflabelgg.com/action/friends/add?friend=43>

Samy sends to Alice a malicious URL (via an email or a posting in Elgg). Here we have a trusted site [www.csrflabelgg.com](http://www.csrflabelgg.com).

We restart the server with `sudo service apache2 restart`.

A user Alice logged into the trusted site, and there is a malicious website. Samy places it as an src attribute in the img tag of a malicious URL that he created. He sends this webpage as the content of a blog. So, when Alice clicks on the link, Samy gets added as a friend. Here a request is sent from the malicious site to the elgg site posing as Alice. This is a cross-site request forgery.

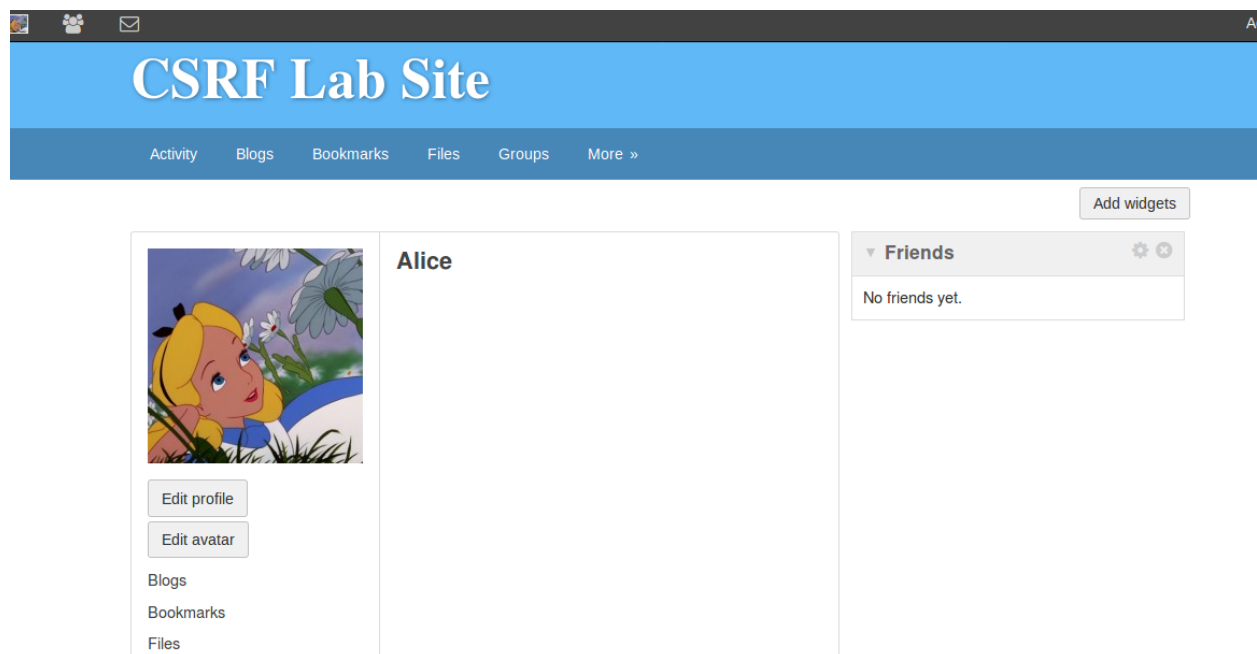


Figure-3: Before visiting csrfattacker.com, no friends

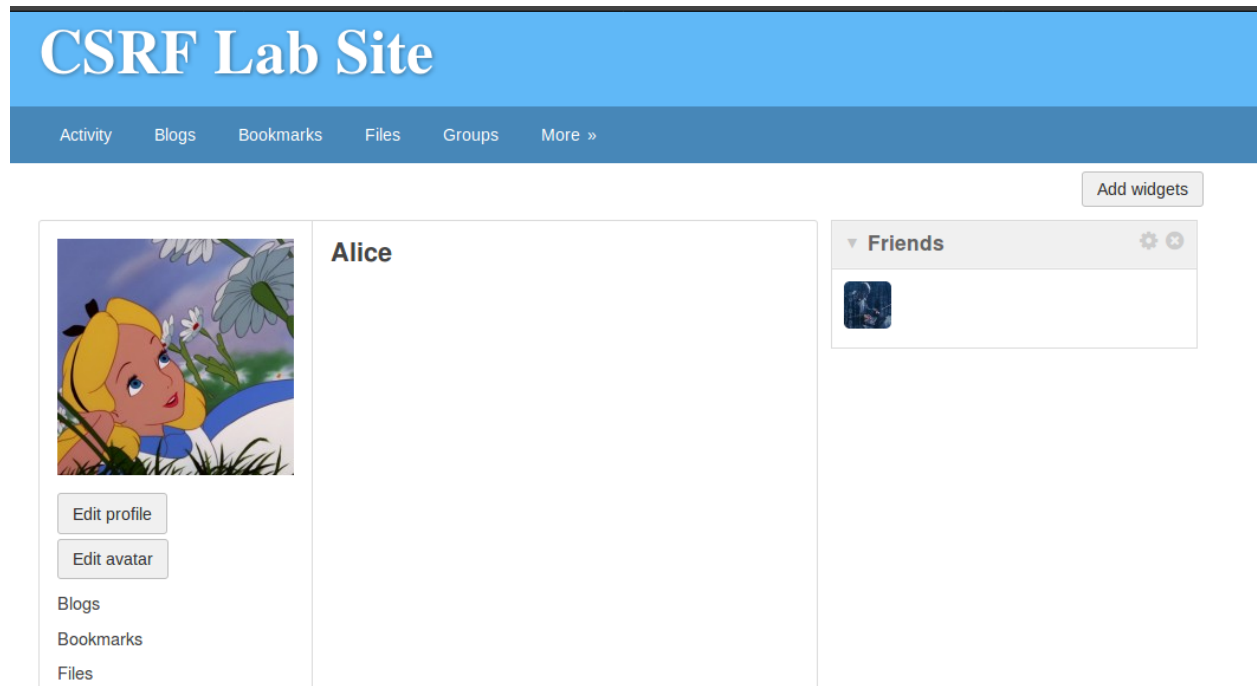


Figure-4: After visiting csrfattacker.com, Samy added as Alice's friend

### 3.3)

#### Task 3: CSRF Attack using POST Request

In this task, after adding himself to Alice's friend list, Samy wants to do something more. He wants Alice to say "Sami is my hero" in her profile. As we have to send data to Alice's profile, POST requests must be sent for this attack. This is a Cross-site request forgery attack where a POST request is used to modify the contents of Alice's profile. Here we have a trusted site [www.csrfllabelgg.com](http://www.csrfllabelgg.com).



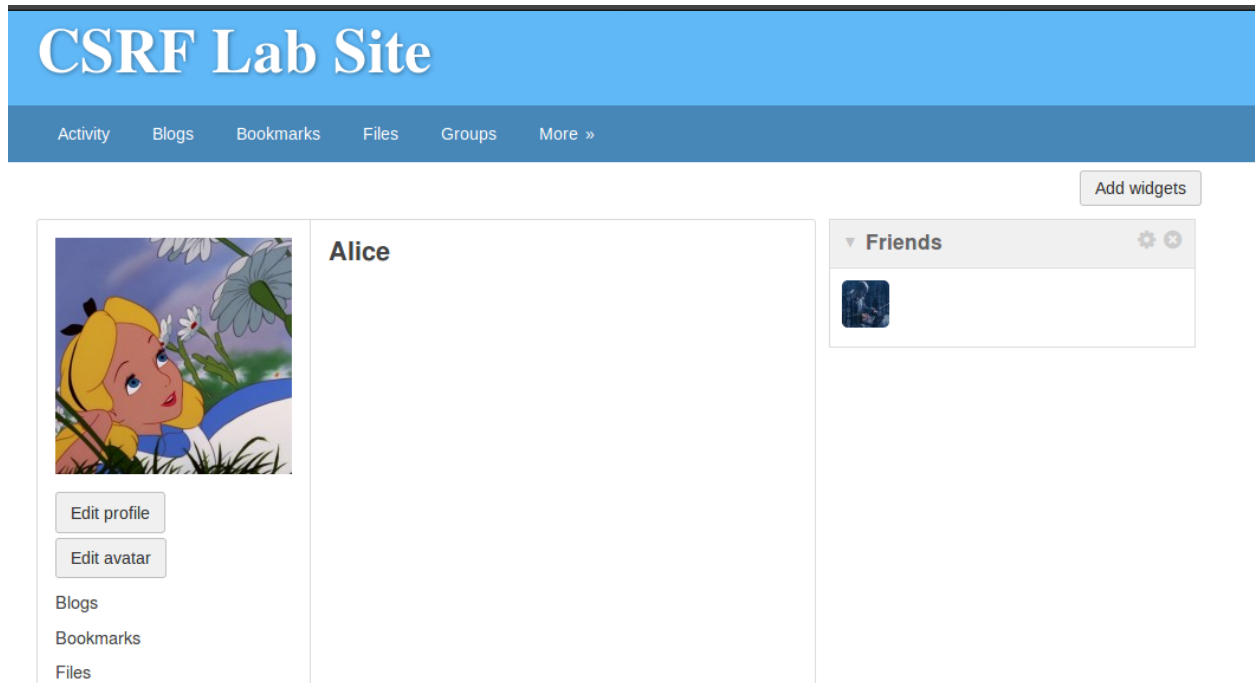
Figure-5: After browsing this html code

Alice logged into the trusted site and a malicious website **www.csrlabattacker.com** created by Samy. So firstly, has to find Alice's id so that she can modify the contents of this profile. Samy goes to the member's page, inspects the elements using firefox, and finds her id. After that, he should construct the URL so that she can generate the POST request that modifies the profile of Alice. Samy changes the brief description in her profile and captures that request using LiveHTTPHeaders. Using the id and changed description of Alice, he writes this html code for executing.

A screenshot of a text editor window titled '\*index.html (~/) - gedit'. The editor contains a JavaScript function named 'forge\_post()' which is designed to forge an HTTP POST request. The code includes comments in blue and JavaScript syntax in various colors. The function constructs a form with hidden inputs for 'name' (Alice), 'description' (Samy is my hero), 'accesslevel[briefdescription]' (2), and 'guid' (43). It then creates a form element, sets its action to 'http://www.csrlabattacker.com/action/profile/edit', sets the method to 'post', and appends the form to the document body. Finally, it submits the form and invokes the 'forge\_post()' function after the page is loaded. The status bar at the bottom indicates 'HTML', 'Tab Width: 8', 'Ln 13, Col 74', and 'INS'.

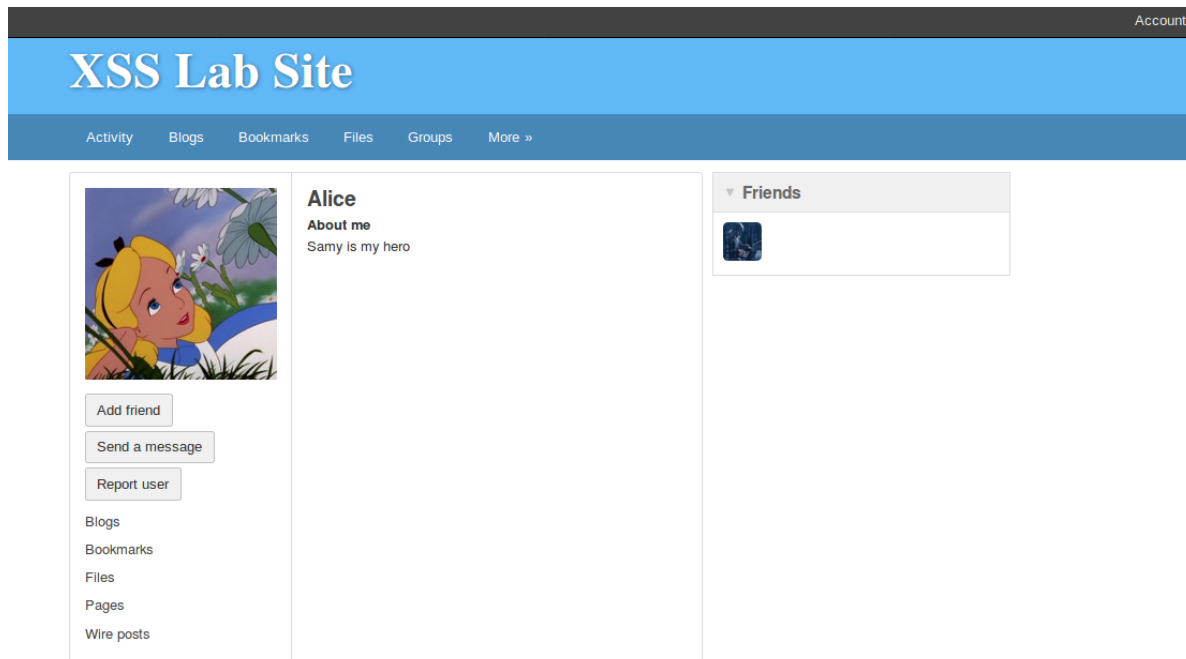
Figure 6: Malicious POST request on index.html of csrfattacker.com

Now, he creates a web page that sends a POST request to the server which recreates the form submission of the profile page with changed contents. He sends this webpage as the content of a blog.



**Figure 7:** Alice's profile before visiting csrfattacker.com

So, when Alice clicks on the link, the contents of the profile are changed. Here a request is sent from the malicious site to the elgg site posing as Alice. This is a cross-site request forgery.



**Figure 8:** Alice's profile after visiting csrfattacker.com

There are 2 questions in this segment. The answer of these questions are given below:

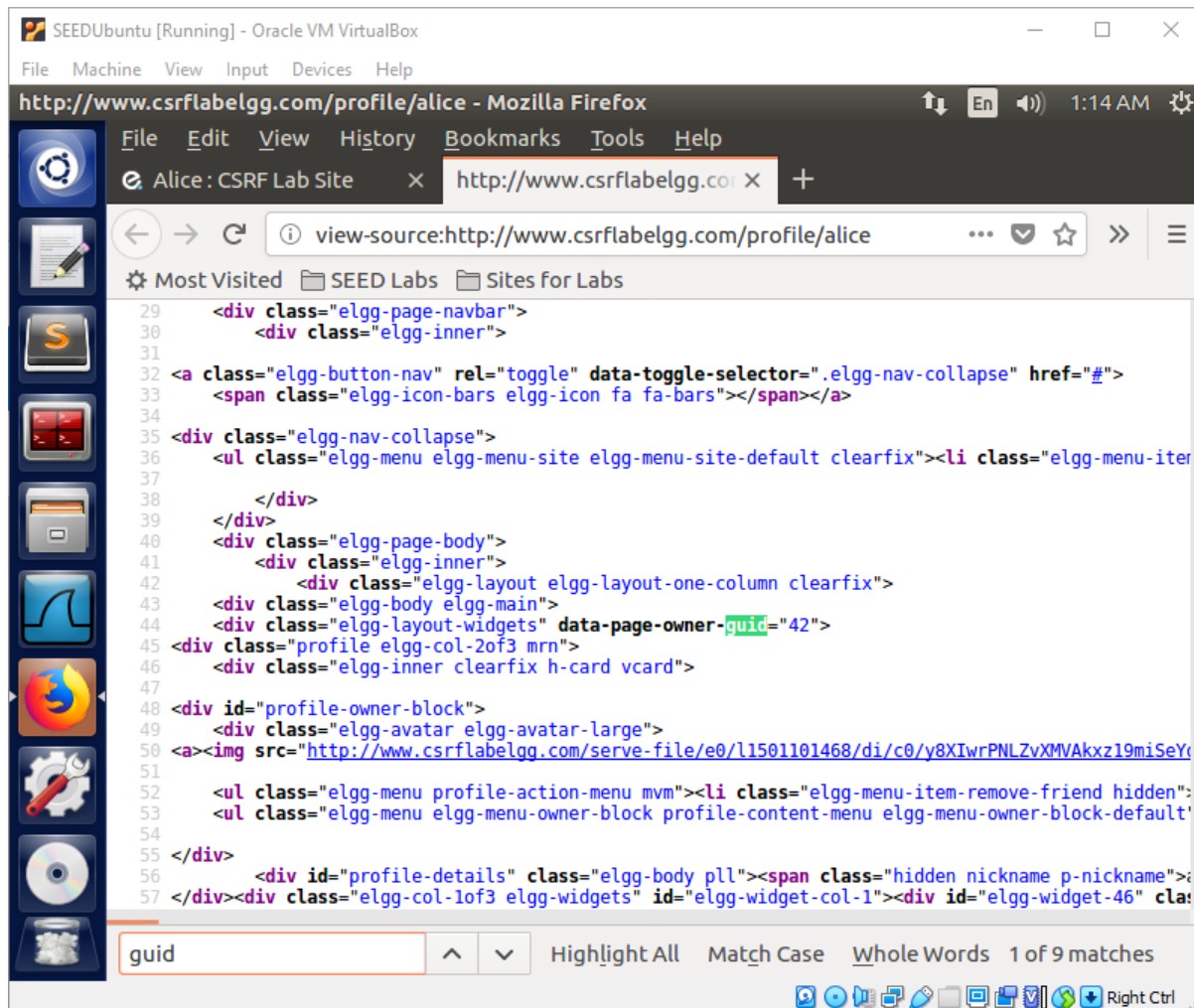
**Question 1:**

The forged HTTP request needs Alice's user id (guid) to work properly. If Samy targets Alice specifically, before the attack, he can find ways to get Alice's user id. Samy does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Samy can solve this problem.

**Ans:** Samy certainly can visit Alice's profile page and view the page source. The page source contains the guid of alice. Then he can modify the guid of HTTP POST and perform the attack.

**view-source:**<http://www.csrflabelgg.com/profile/alice>

**guid = 42**



**Figure 9:** Finding any user's from their profile's source page

**Question 2:** If Samy 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.

**Ans:** From the explanation and html code, we have already seen that it contains the name and the guid. So, in case the attacker doesn't have the following information,

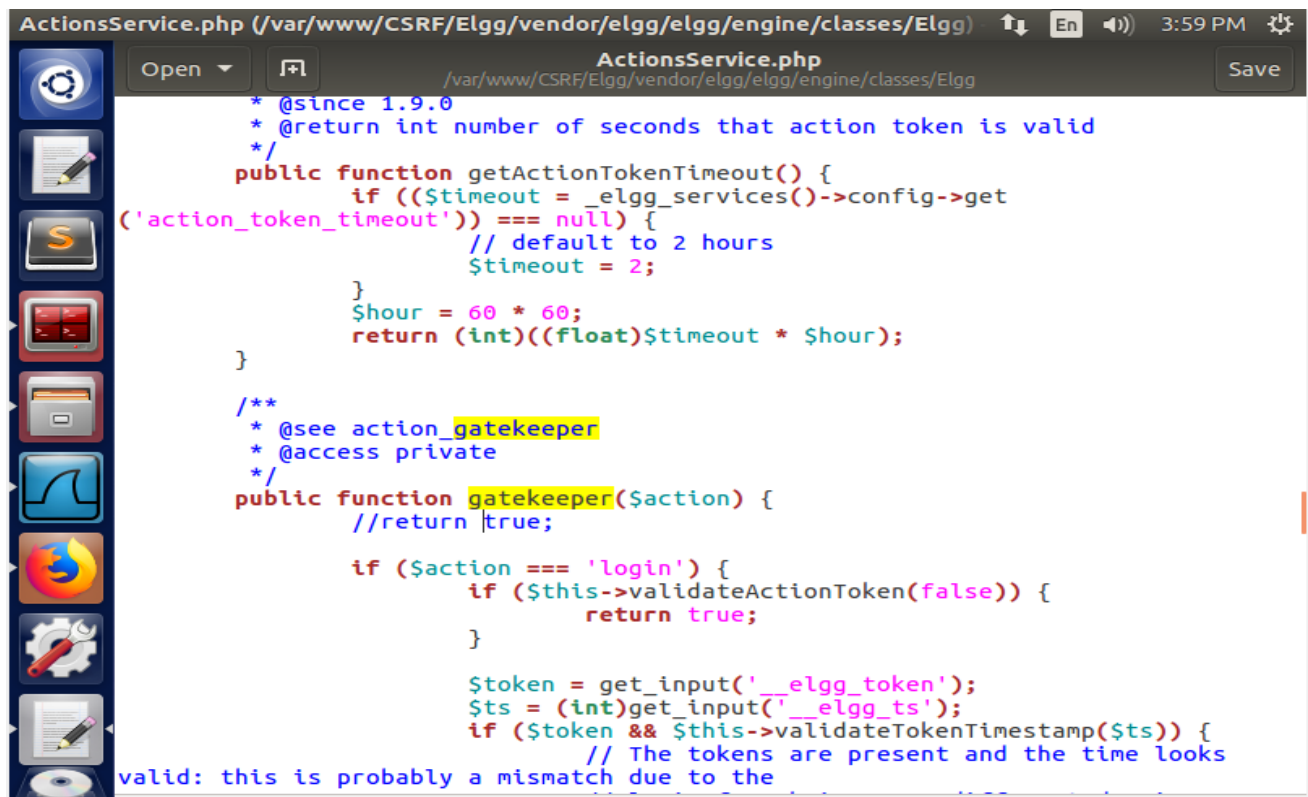


more specifically, without knowing the name and guid, Samy(the attacker) can't launch the CSRF attack to modify the victim's elgg profile.

### Task 3: Implementing a countermeasure for Elgg

Firstly, we have to visit this specific path and access the ActionService.php file. The path is: `/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionService.php`.

In that Actionservice.php, we will modify the gatekeeper function. It's clear that this function always returns true. So, we will comment on that line. As a result, the function is checking the user cookies. It also checks timestamps of the user.



```
ActionsService.php (/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg)  En  3:59 PM
Open  ActionsService.php  Save
/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg
* @since 1.9.0
* @return int number of seconds that action token is valid
*/
public function getActionTokenTimeout() {
    if (($timeout = _elgg_services()->config->get
('action_token_timeout')) === null) {
        // default to 2 hours
        $timeout = 2;
    }
    $hour = 60 * 60;
    return (int)((float)$timeout * $hour);
}

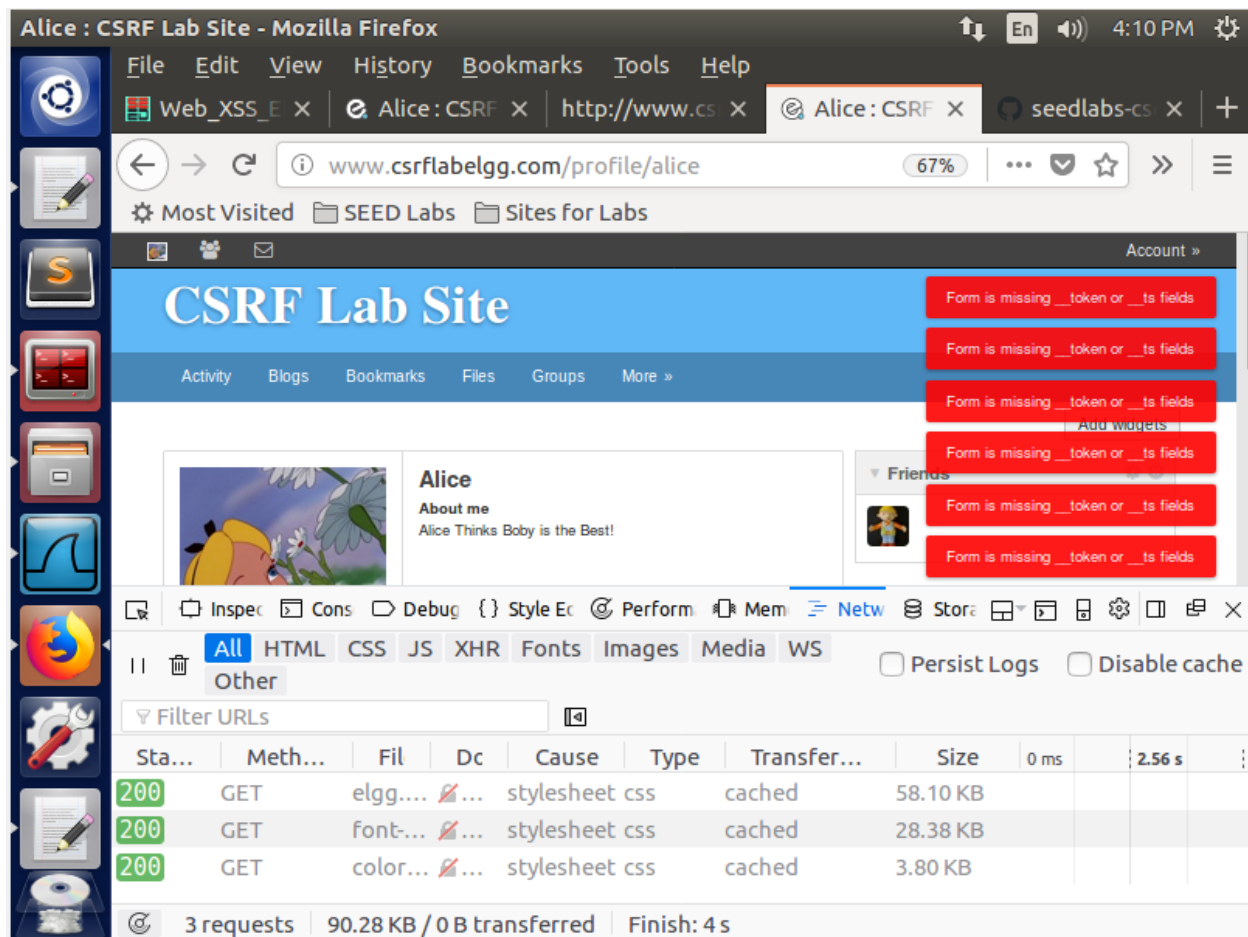
/**
 * @see action_gatekeeper
 * @access private
 */
public function gatekeeper($action) {
    //return true;

    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 is probably a mismatch due to the
        }
    }
}
```

**Figure 10:** Commenting out return true statement

This can stop the request forgery attack.



**Figure 11:** Attack unsuccessful

Now if we perform the same attack again, we see failure in CSRF attacks and get the error message:

**“Form is missing \_\_\_token\_\_ or \_\_ts\_\_”.**