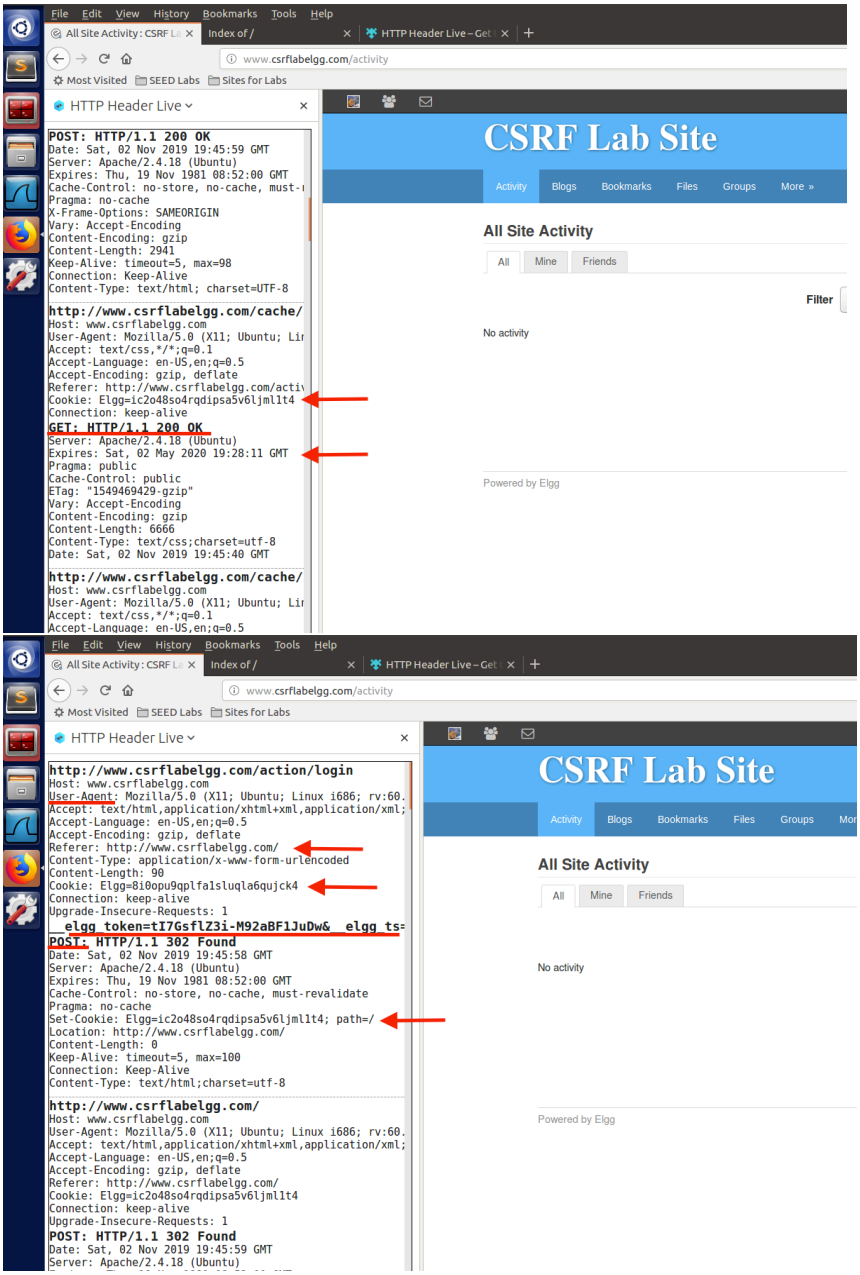
Name: Bharath Karumudi
Lab: Cross Site Request Forgery Attack

Task 1: Observing HTTP Request.



Observation: Installed the “HTTP Header Live” addon to the Firefox and opened the Elgg website to examine the HTTP headers. I see the header requests of the HTTP between browser and host.

Explanation: When opened a page, we can see various parameters like host, User-Agent, Referrer, POST and GET requests, cookie details, expiry date and time. These will be helpful in understanding some part of information exchange between browser and host.

Task 2: CSRF Attack using GET Request

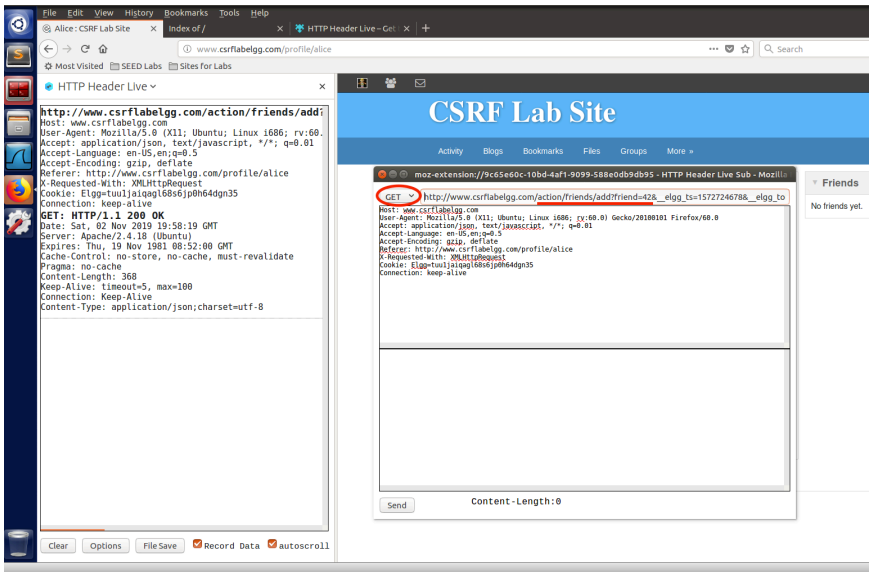


Fig: Examining a GET request to add a friend.

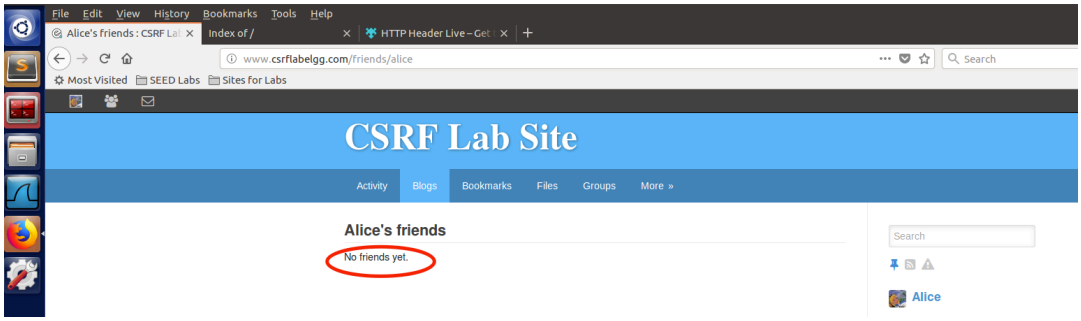


Fig: Confirmation showing “No Friends” in Alice account.

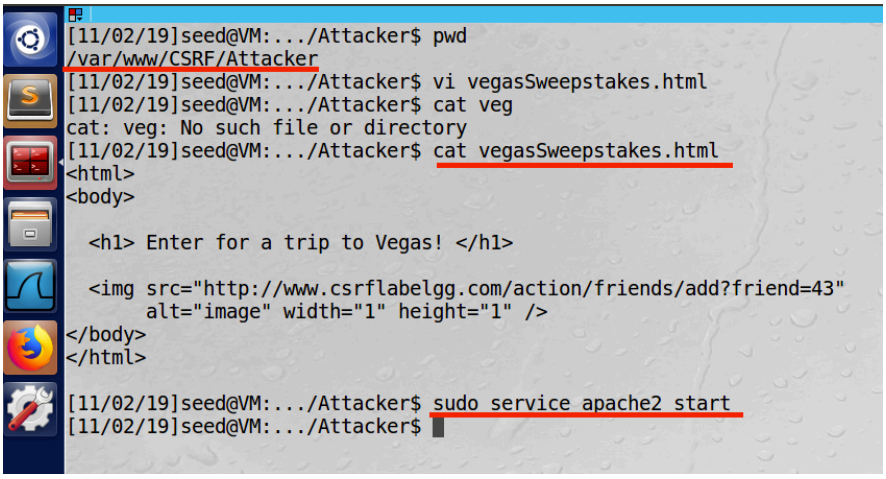


Fig: Created a page to attract Alice – vegasSweepStakes

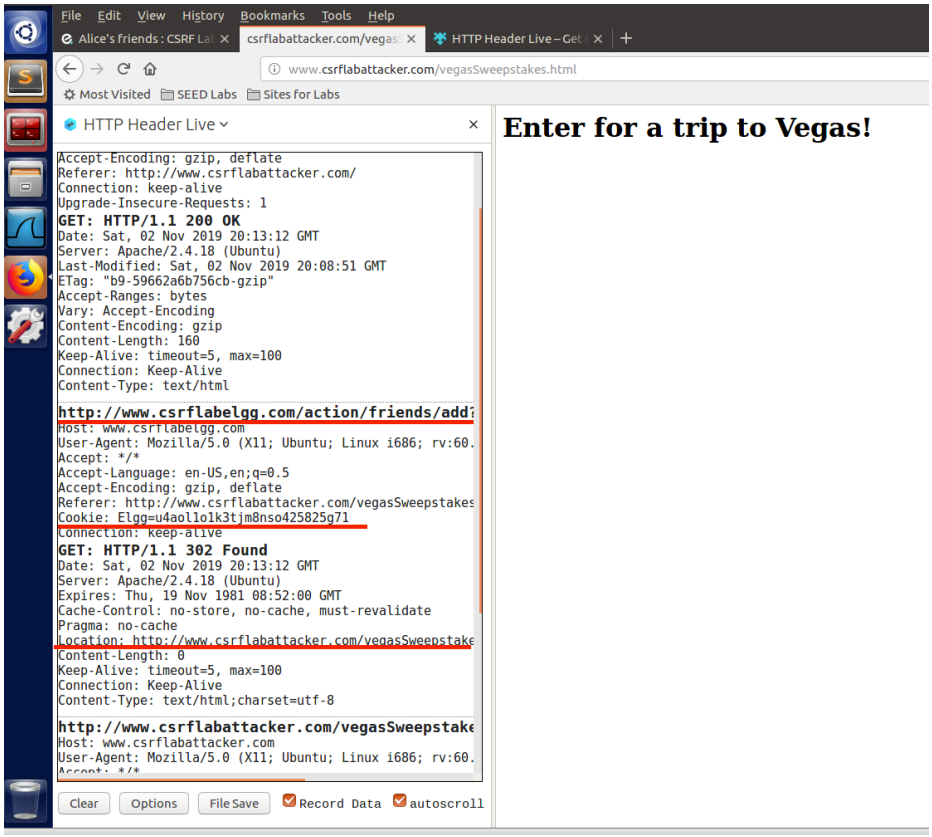


Fig: Alice opened the vegasSweepstakes page, which is attacker page.

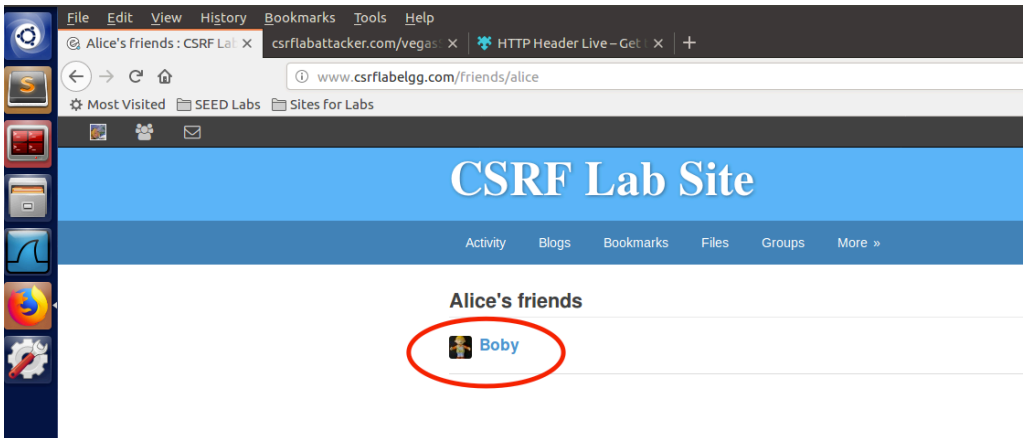


Fig: Boby, the attacker was added to Alice friends without Alice knowledge.

Observation: Created a malicious webpage with the GET request by observing how Elgg handles the add friend requests and made Alice to click the link. When Alice opened the link, Boby was added to Alice Friends list without her real intention.

Explanation: This is due to the Cross-Site Request Forgery attack (CSRF). Boby used the CSRF vulnerability and observed how to add a friend and built the GET request which server will accept for Adding a Friend and placed it in his malicious web page.
When Alice clicked the link, the browser loaded the page, but due to GET request embedded in the malicious page, the browser sent a GET request to the Elgg along with the cookie that is associated with the Alice. The server validated the cookie and considered as a legit request and added the Boby as Alice friend.

Task 3: CSRF Attack using POST Request

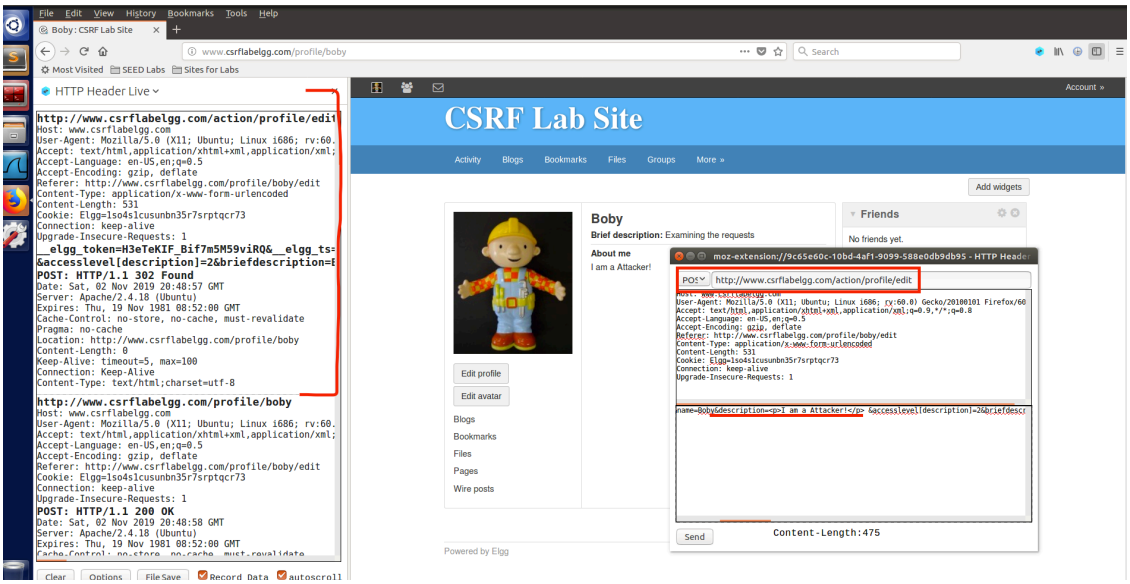


Fig: Examining the POST requests for Profile editing

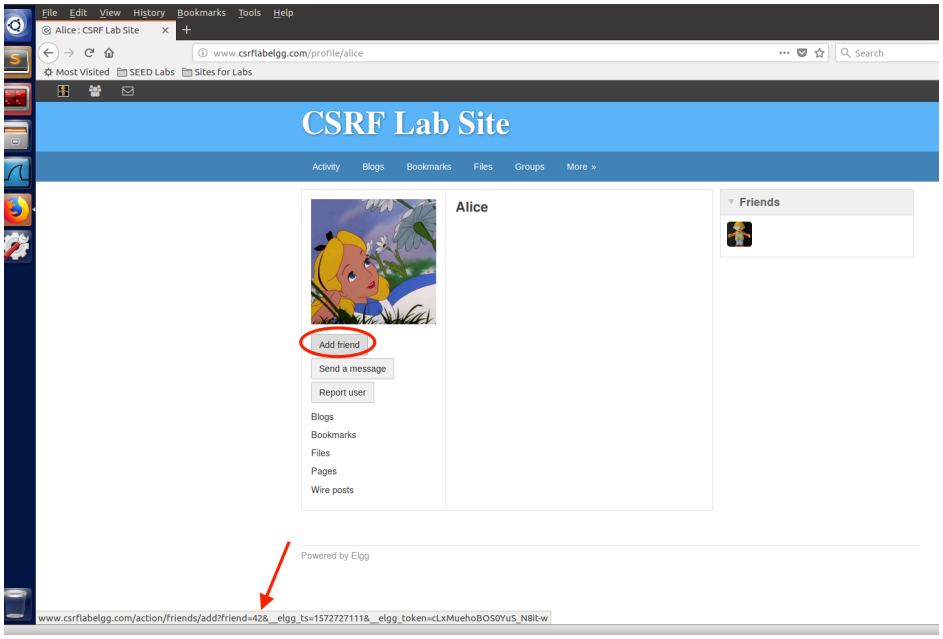


Fig: Finding Alice GID

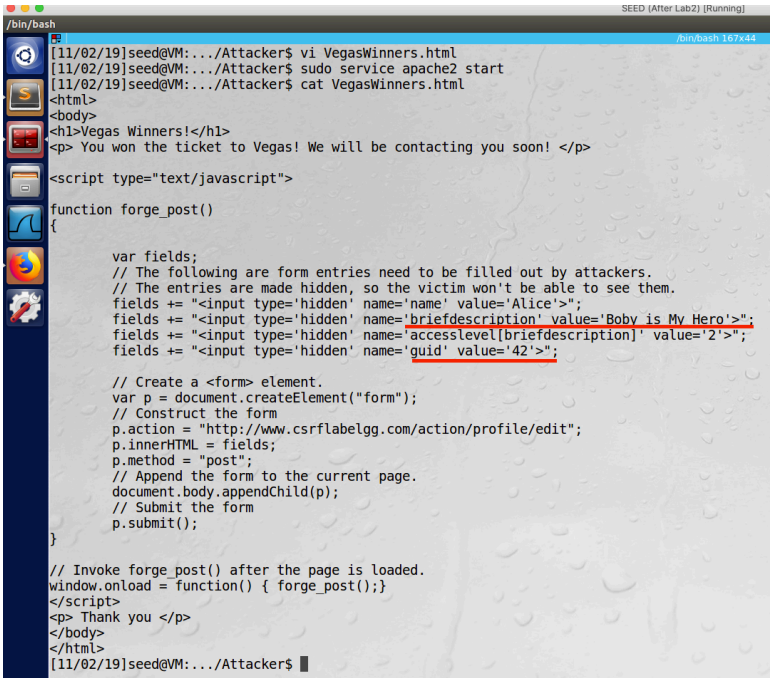


Fig: Created a new web page to attack Alice and update her profile page

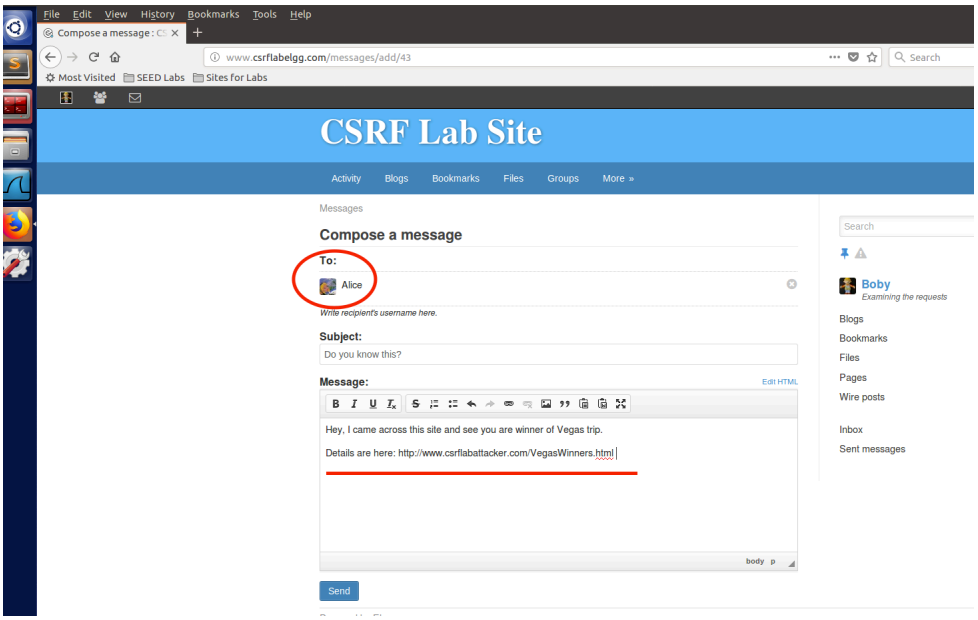


Fig: Sending a message to Alice with the malicious URL

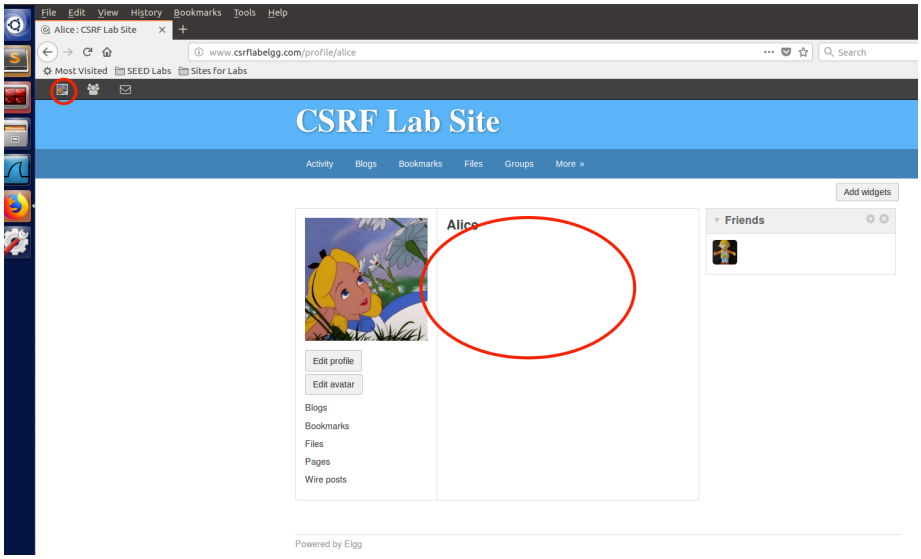


Fig: Logged in with Alice and verified no profile descriptions.

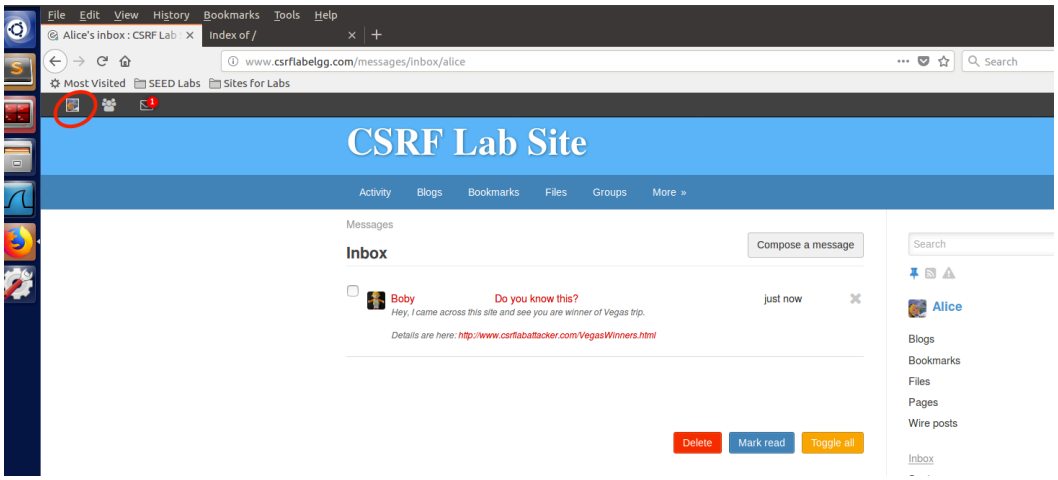


Fig: Received a message from Boby

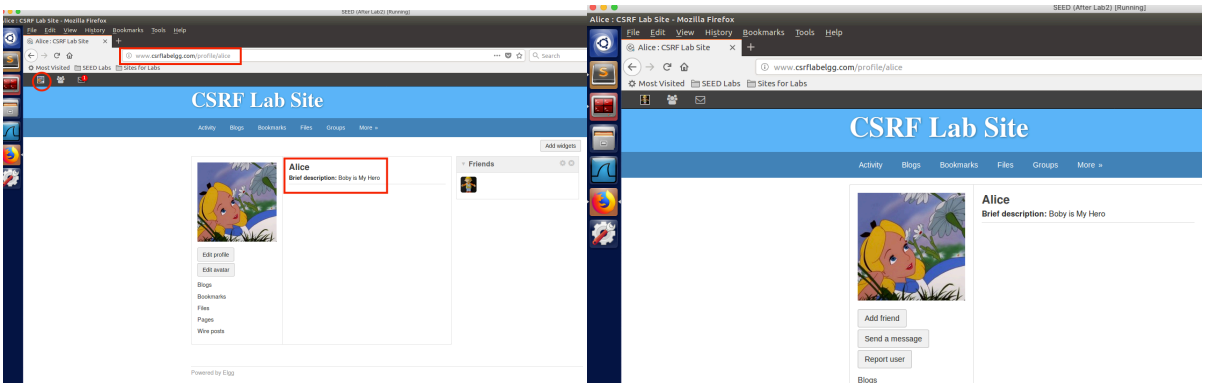
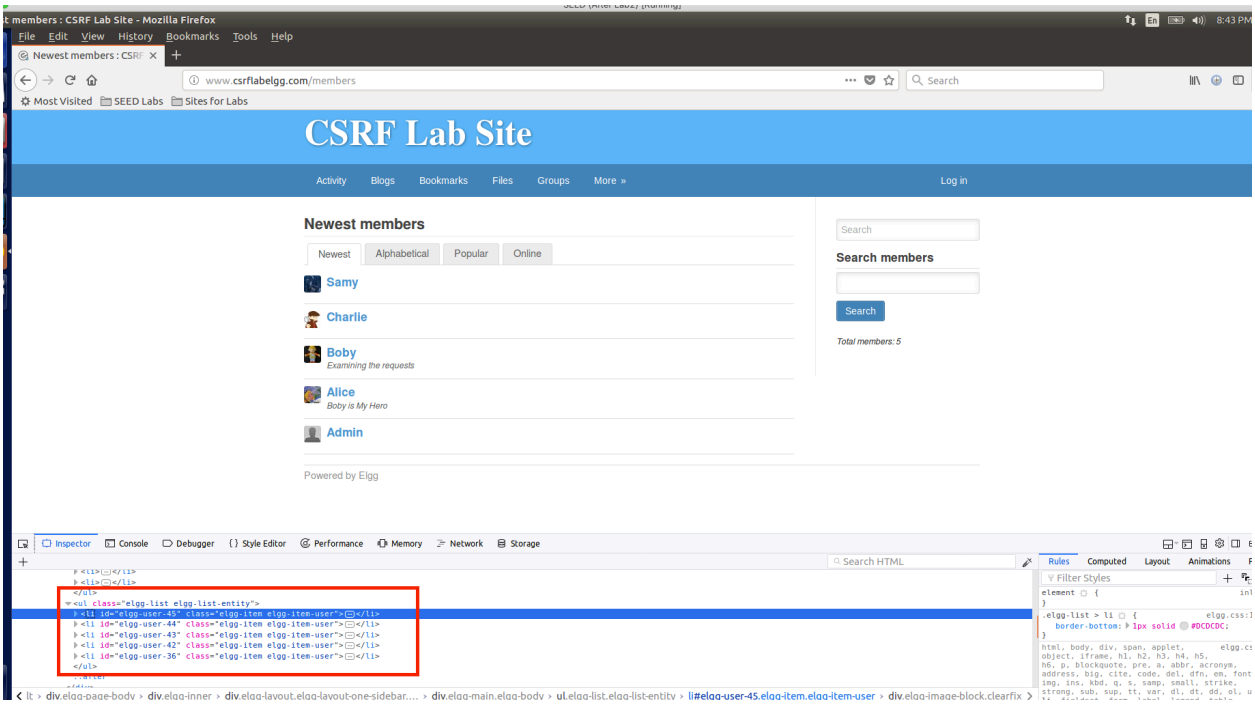


Fig: Alice profile has updated, and the description is publicly visible. *Attack successful.*

Observation: Created a new web page with POST request embedded into it which the server can understand the request to edit the profile. Once Alice clicked the link, her profile description has been updated to “Boby is my Hero” and also it is public.

Explanation: This is due to CSRF attack, where Boby examined how Elgg handles the edit profile request. Based on that, he created a malicious web page with the POST request and shared the malicious URL with Alice. When Alice clicked the URL, the browser sends the POST request to Elgg website and also her cookie. The webserver authorized the profile description change as it is coming from Alice Elgg session and the profile has been updated without her knowledge.

Question1:



Boby can just search for members and go to Alice profile. When he hovers the mouse on the "Add Friend" button, the URL will show the GID (friend=) (as shown in second image above). Also, the other way is, by going to the members page and examine the html using developer's tool, inspector (as shown in above screenshot). By this way, Boby can get GIDs of all users in one place.

Question2: No, it is not possible for Boby to launch the attack on anybody who visits his malicious web page. Because, he needs GID of the visiting user first to attack. The attack will be successful only when the ID that was mentioned in the malicious page is matched with the Elgg active user who is visiting the malicious page.
To do a brute force attack on all IDs, it will be difficult, if Elgg has large number of members and also if GIDs are random.

Task 4: Implementing a countermeasure for Elgg

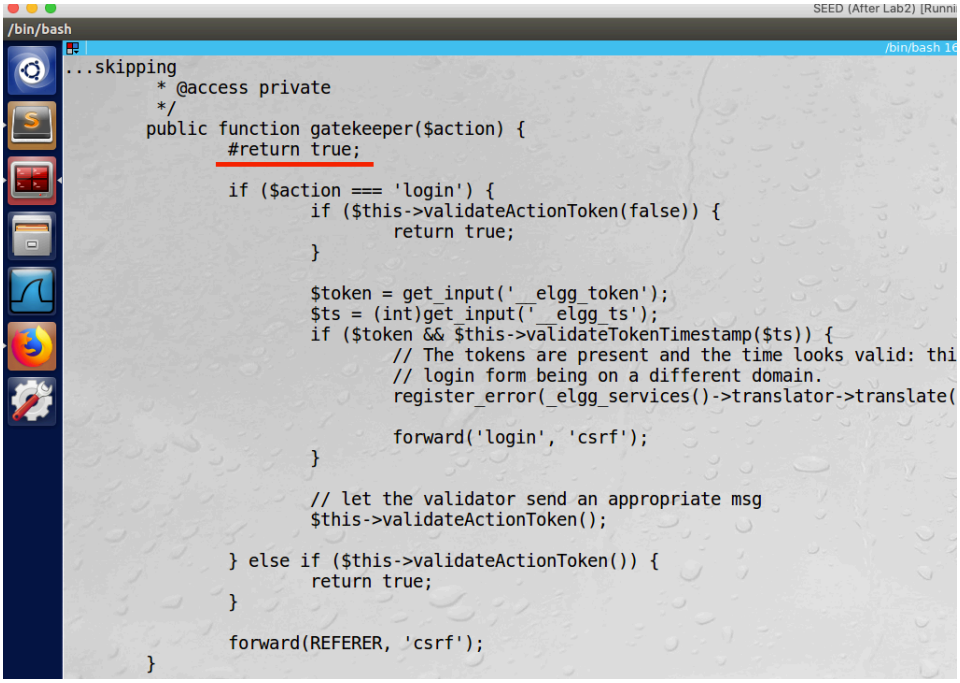


Fig: Turned ON the Elgg CSRF countermeasures

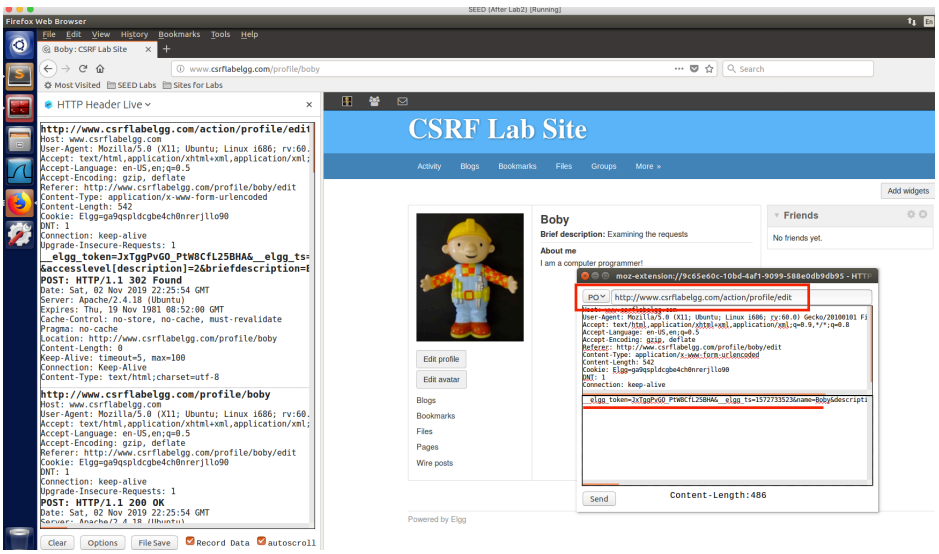


Fig: Examining the POST request

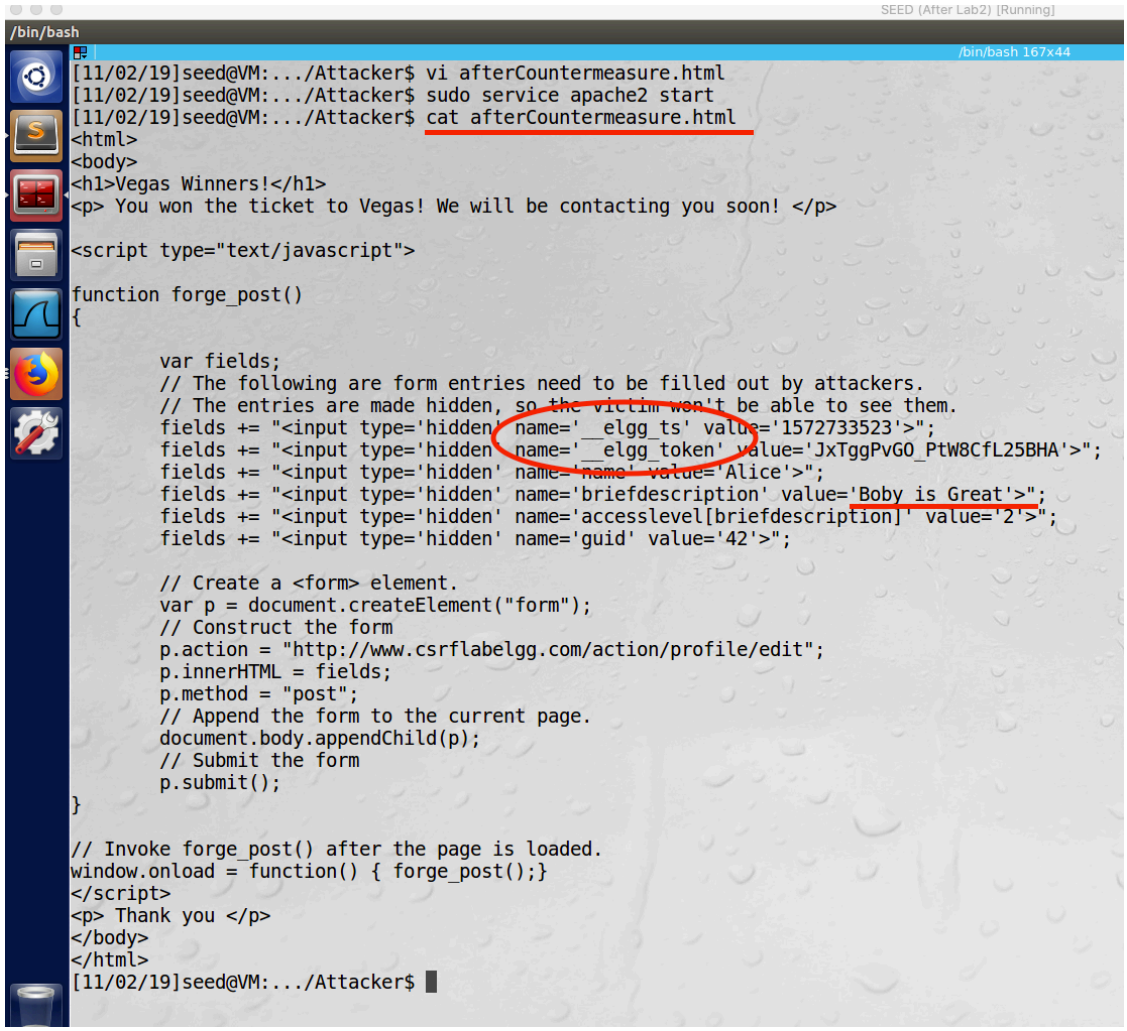


Fig: Created a new malicious page with updated description and secret tokens

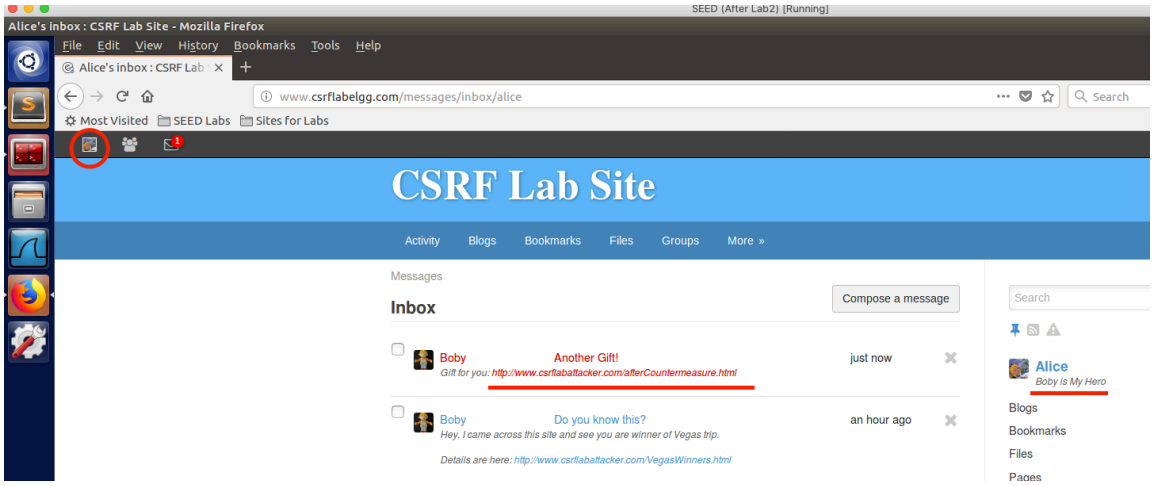


Fig: Boby sent a new malicious webpage to update the Alice profile description

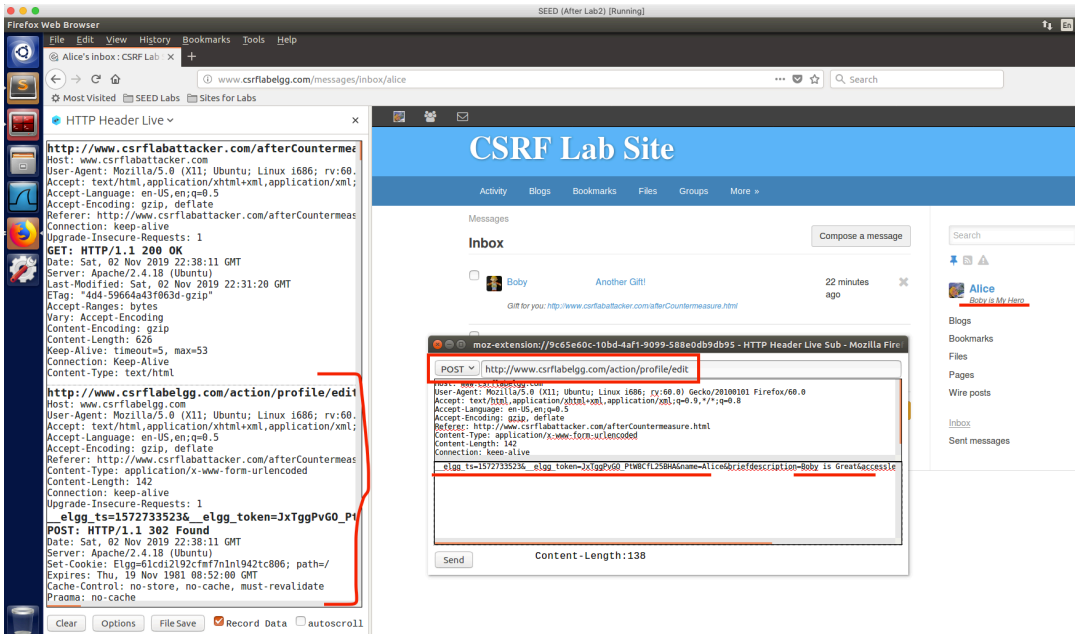


Fig: Alice clicked the URL, but no impacted. *Attack Failed.*

Observation: After turning ON the CSRF countermeasures, again observed the HTTP headers for updated post request for attacking and saw the __elgg_ts and __elgg_tokens. So updated the malicious page with new headers and performed the attack to update the Alice Profile description as done similar to Task 3, but the attack was failed.

Explanation: Elgg now has the timestamp and secret token validations that are tied to Alice session and when Boby tried to attack the Alice with new malicious page even by updating the POST request including the __elgg_ts and __elgg_token, it failed. Because the tokens are not matched with the tokens that are with Alice session. Thus, the CSRF attack was defended by the web server and this is a countermeasure for CSRF attacks. To succeed, attackers need to know the values of the secret token and timestamp embedded in the Alice's Elgg page. Unfortunately, browser's access control prevents the Javascript code in attacker's page from accessing any content in Elgg's pages.