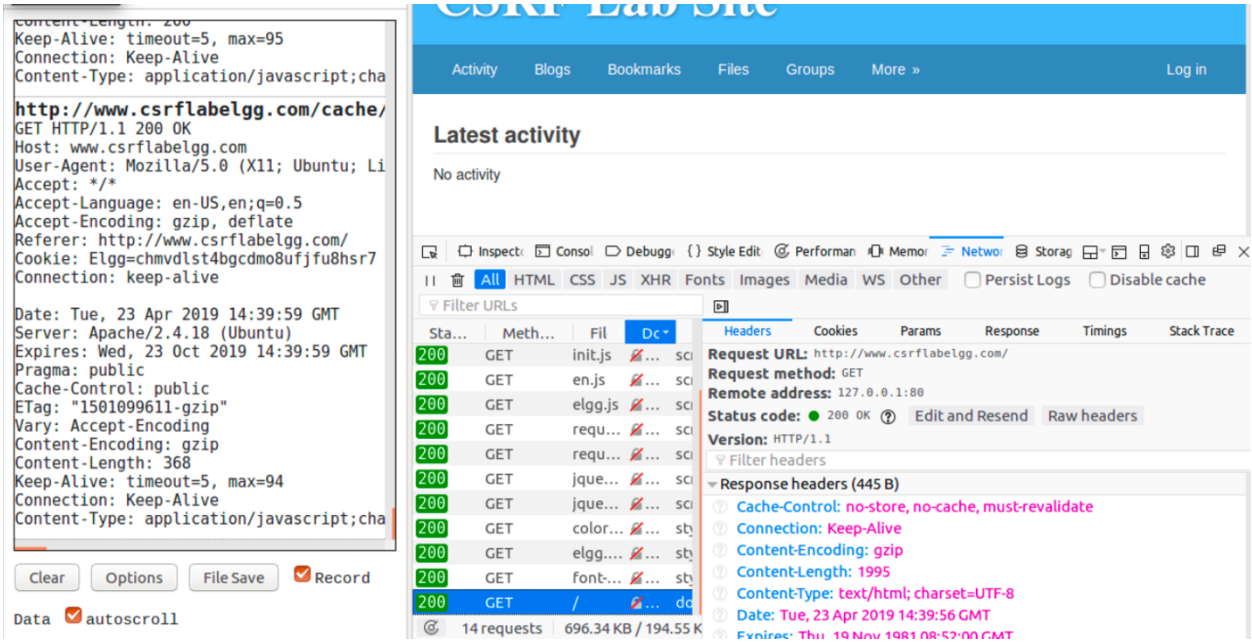
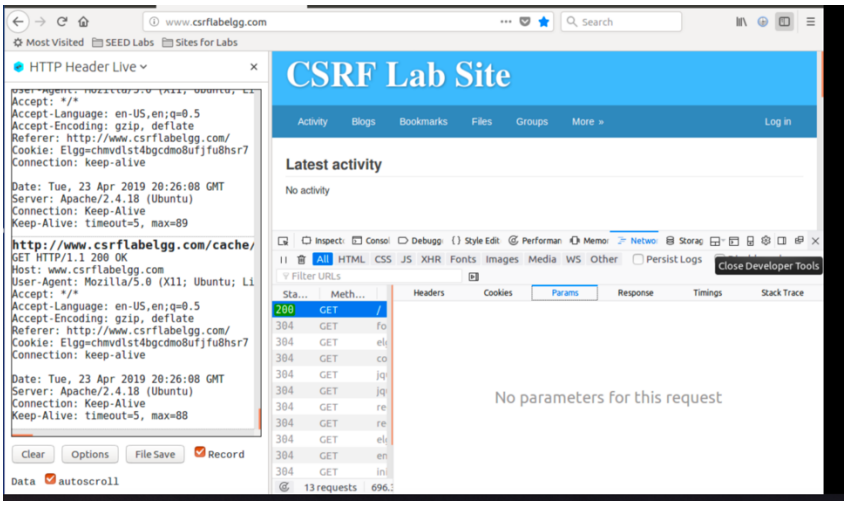
Gabriel Naples, Dylan Bathurst, Rachel Toler

Cross-Site Request Forgery Lab

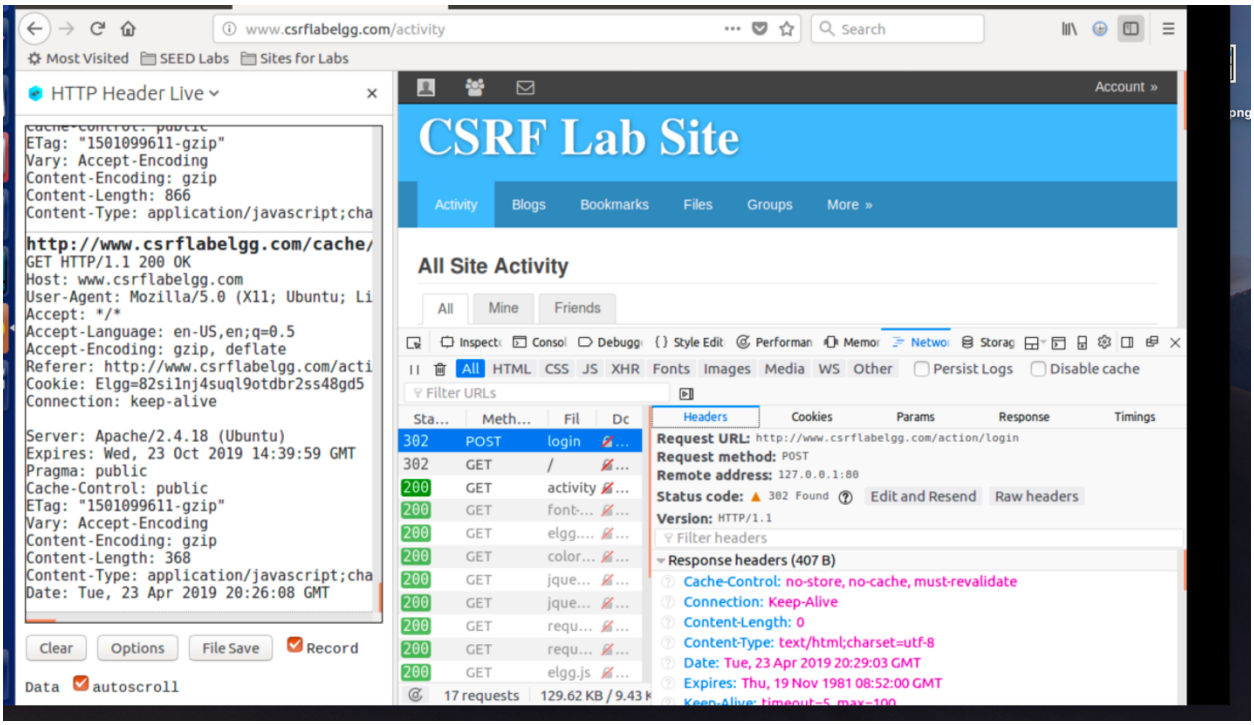
**Task 1: Observing HTTP Request**




In both the left panel (HTTP Header Live) and the web inspect element inspecting the network requests is an example of a GET request to the CSRF lab website.



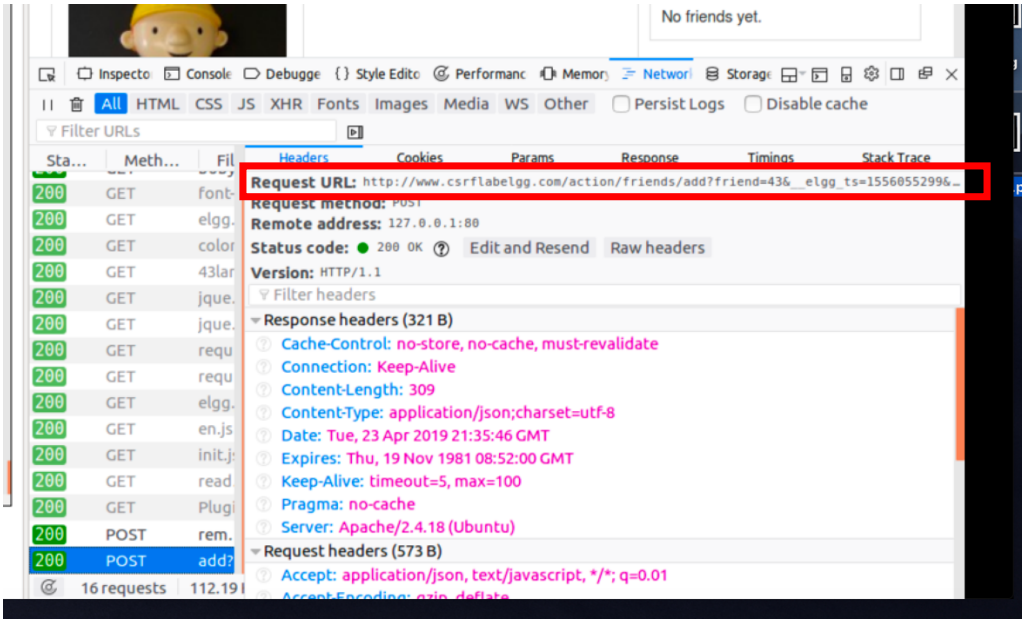

The GET request has no parameters.



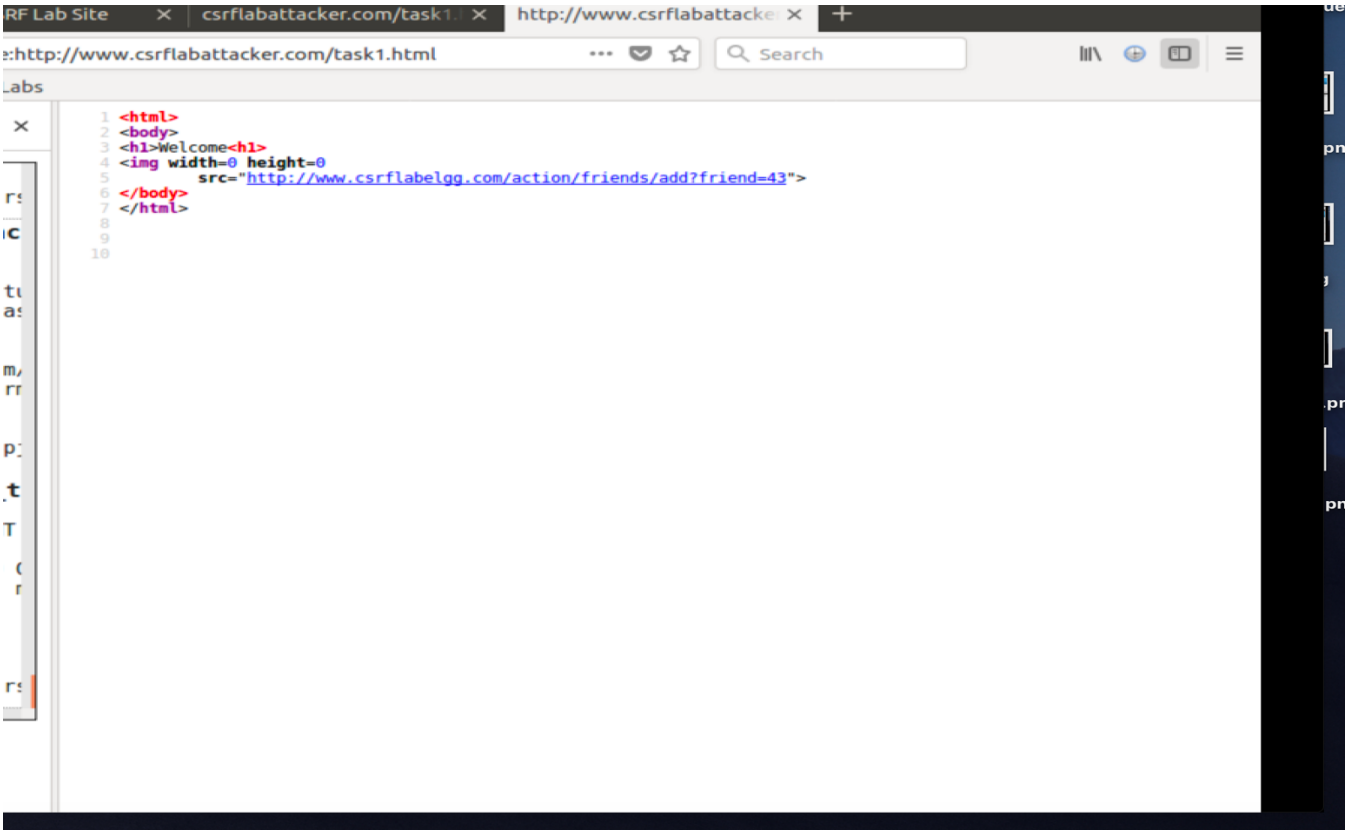

This is the similar to above, but the POST request is only being analyzed in the network section of the web inspector. We preferred the layout of the web inspector since the organization was easier to read.


The POST request did have parameters. We were logged into the admin account on the site and this request appeared when interacting with the Login form. The token and timestamp parameters are the ones that were disabled for this lab to function properly, as they are the countermeasures to the CSRF attack.

**Task 2: CSRF Attack Using GET Request**

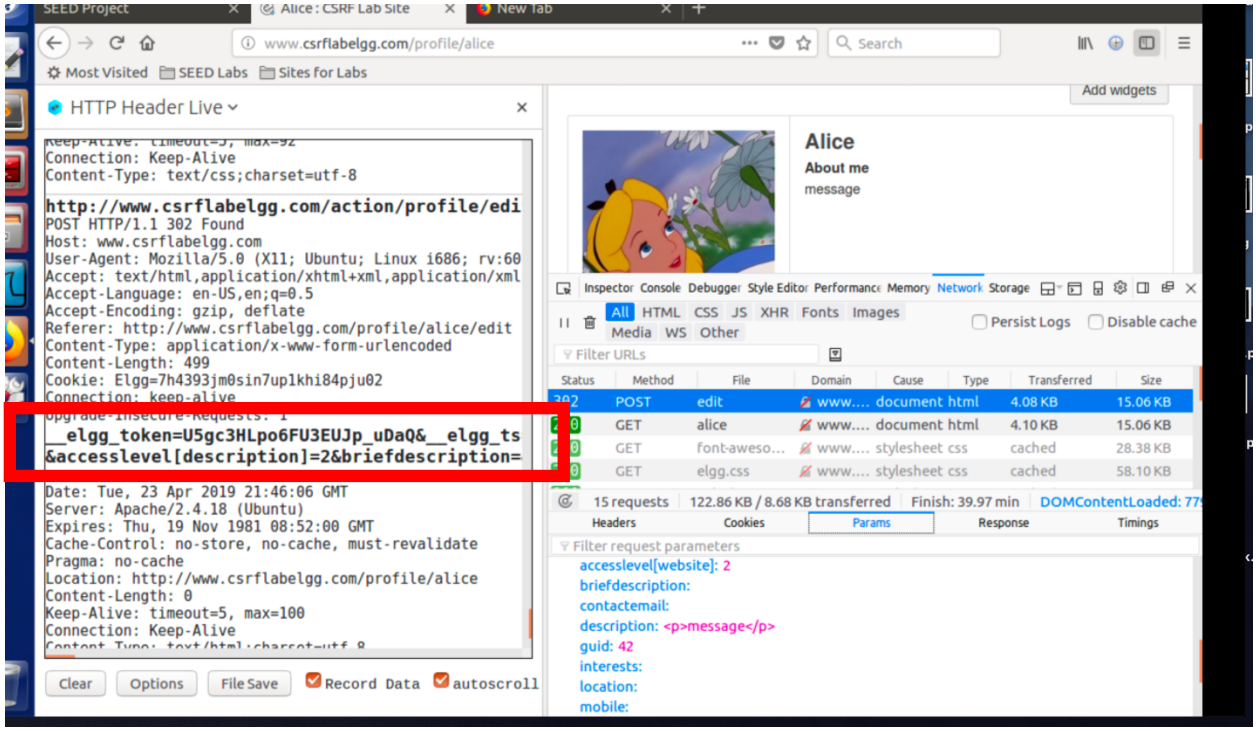



This screenshot is me analyzing a GET request sent by Alice when we manually added Bobby the legitimate way using the website. This way we were able to determine what the malicious GET request should look like and also how we found the ID number that was assigned to Bobby which is necessary for the request. Outlined in red is the URL we looked at to get the information. Though we were analyzing a POST request, it still contained the information we needed to make the GET request.

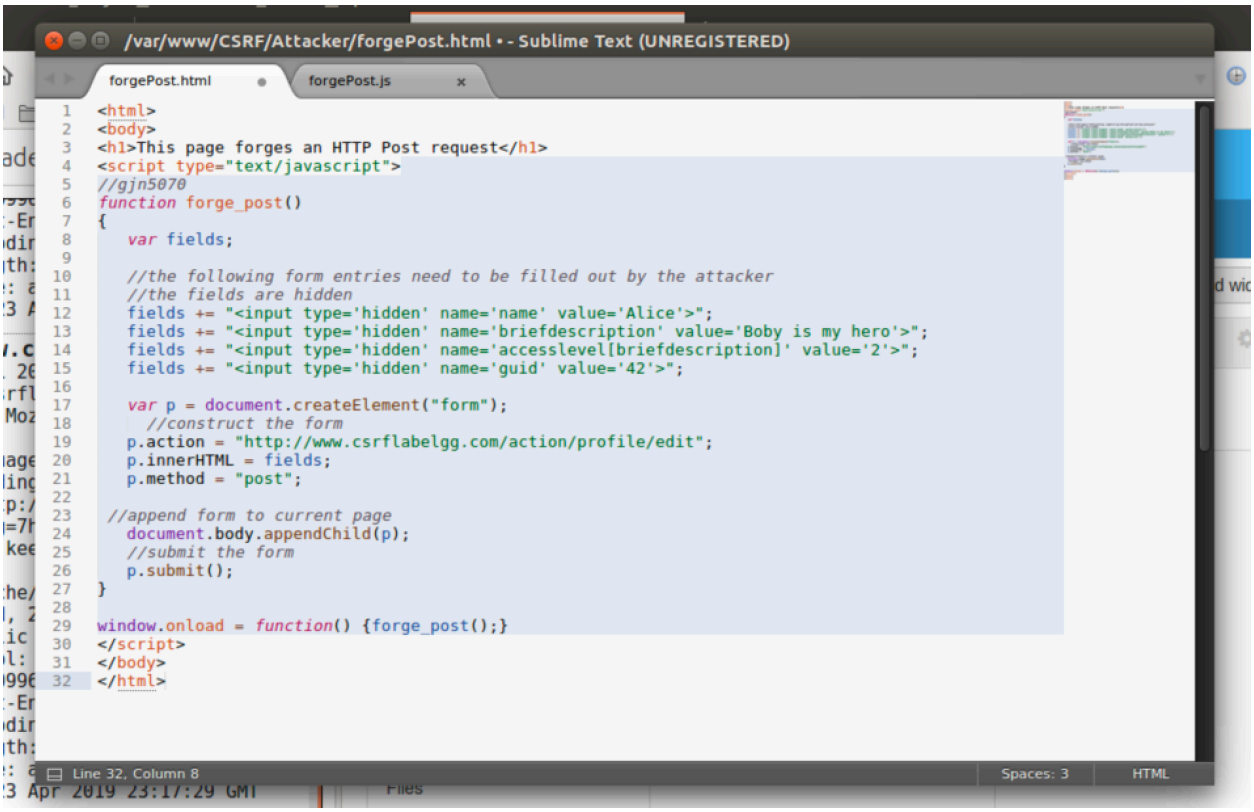



Above is the code on the malicious website page that runs and forces the browser to send the GET request that automatically adds Bobby to Alice's friends list. For this to work properly Alice must be logged in to her account already, as this cross-site attack requires her cookie which establishes communication with the web server. When she clicks the link that leads to this page, the page loads the GET request as an image since images load automatically, thus sending the GET request automatically. Furthermore, this image is not even visible on the page as the dimensions were set to 0. Since Bobby is trying to add himself, the ID in the link (43) must be his own.

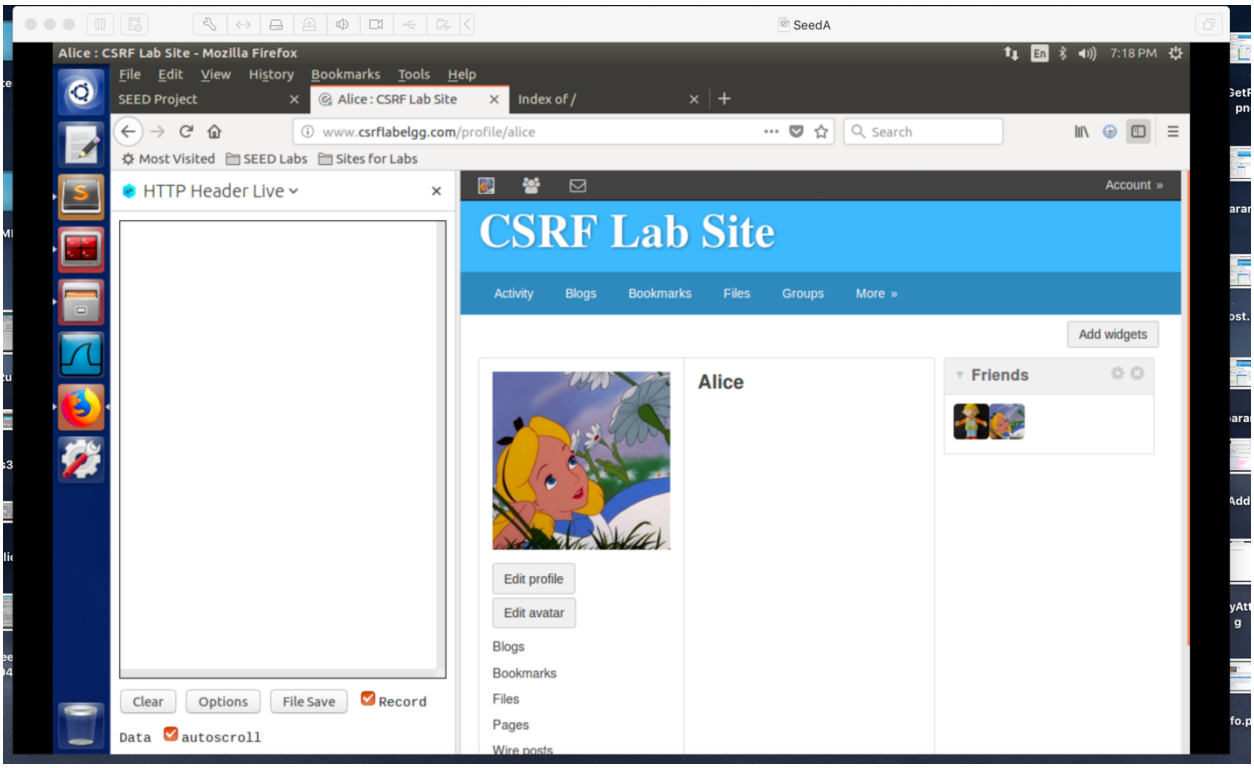
## Task 3: CSRF Using POST Request



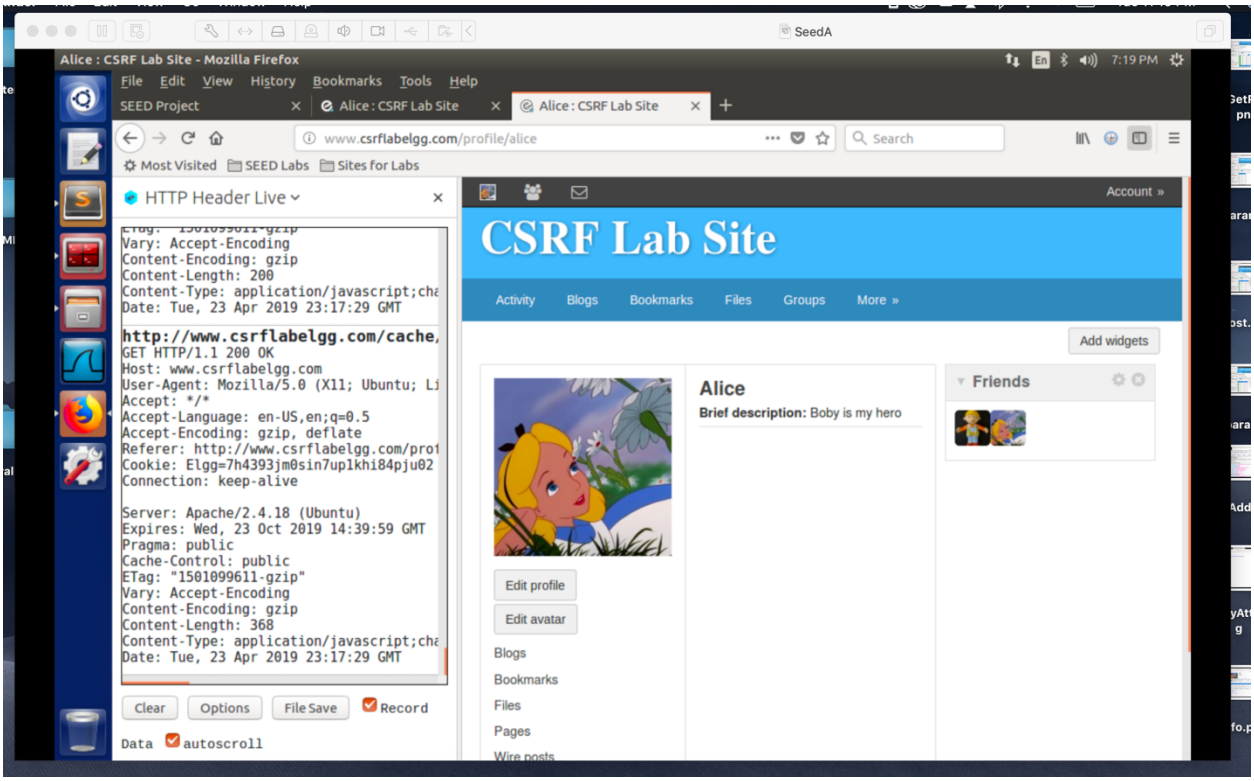

For this attack, we had to figure out what the POST request looked like when it was sent out the proper way. In the HTML Header Live tool (the left column) it shows the bulk of the information we needed within the red box. This is a legitimate request that shows how the elements of the form are sent to the server. In the web inspector it is showing the parameters of the same request which shows the information in a more readable format. The description with “message” in it is the line we added to test this request.




Above is the code we made on an html page. When Alice loaded the page, this code ran automatically and forced a POST request that added ‘Bobby is my hero’ to her description. This had to be created using JavaScript and had to be sent in the format of a form. This code creates a form and fills in the information that has to be changed. The line at the bottom ‘window.onload’ is what makes the code run right as the window loads. The url that the form is sent to is ‘http://www.csrflabelgg.com/action/profile/edit’ which is directly to the same server that the request gets processed if Alice were to change this information herself.

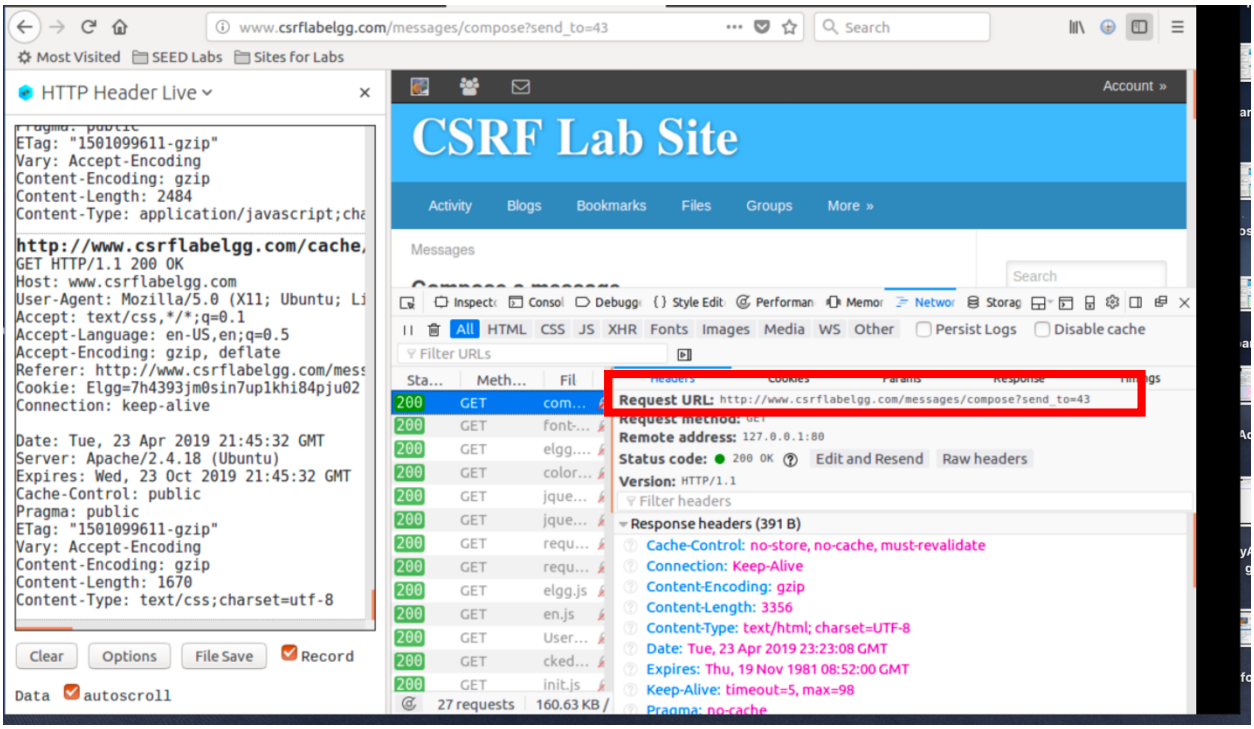



This screenshot is her page before clicking onto the malicious page.




This is her page after clicking onto the malicious website.

Question 1: Since Bobby would need Alice's ID prior to executing this attack he must be creative in his way of finding her ID. A method that could use to find her ID is to send her a message from his account and analyze the get request created when this happens. He doesn't even need to send the message, when he enters the field to compose the message a request is sent that shows who the message is going to be sent to. Below is a screenshot of a composed message to Alice, and within the red box is her ID.




Question 2: Bobby can employ methods to make any user that accesses his website to have their page automatically altered. It would require some clever coding, however. A way he can do this is have the page add him as a friend automatically when they enter it. Once he is on their friends list he can make a program that automatically creates a message to be sent to the newest person on his friends list, this will generate a 'send_to= ' request which would give him the userID of the persons bio he wants to manipulate. The program could pull the get request and scrape out the userID for the attack.

**Task 4: Employing the countermeasure**

After the countermeasure was employed, neither of the attacks worked. The attacks still get sent to the server, but they can't send the secret token that is generated for the user on the csrflabelgg site. These tokens that are sent to confirm with the server the authenticity of the request is algorithmically generated and therefor very hard to break, making it an effective deterrent for the attack. The way these tokens work is by running a hashing algorithm that uses parameters on the client side and creates a unique hash using things like timestamp and

sessionID. The server side also creates a hash and these values are compared to prove authenticity.