Matthew Kachar

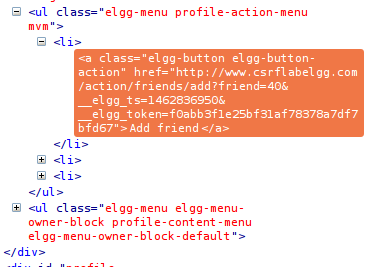
COSC4931 Project 9

Lab : Cross-Site Request Forgery Attack

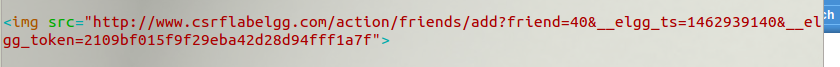
Objective — Understanding the cross-site request forgery attack, including the components involved, the necessary access for the attacker, and the countermeasure for defending users against manipulations of their personal data stored on a given web application.

The first step to constructing the CSRF attack is developing a malicious site from which we can inject some forged HTTP request. To perform the GET request we need to observe what code the legitimate site is either executing or looking for certain functions are call. The first task is asking us to use the malicious site to add Bobby to Alice’s friends list.

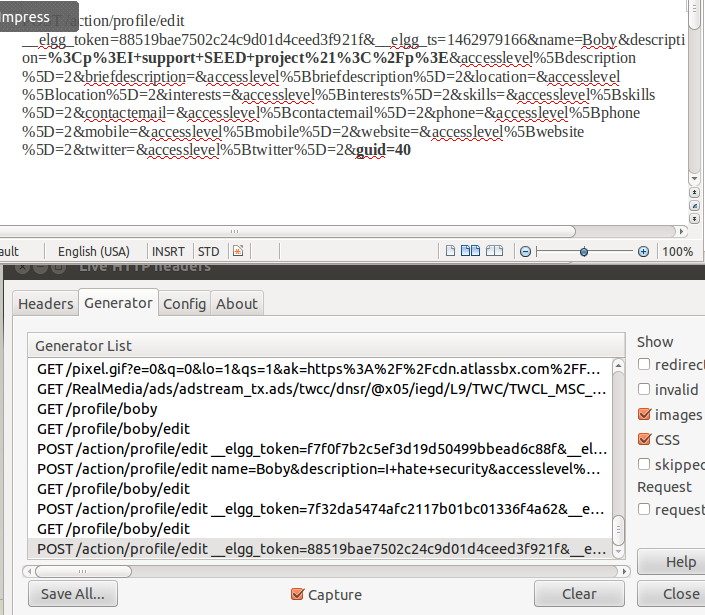
Inspecting the action associated with the button “add friend” we can see the http which actually adds the desired friend to the user’s list



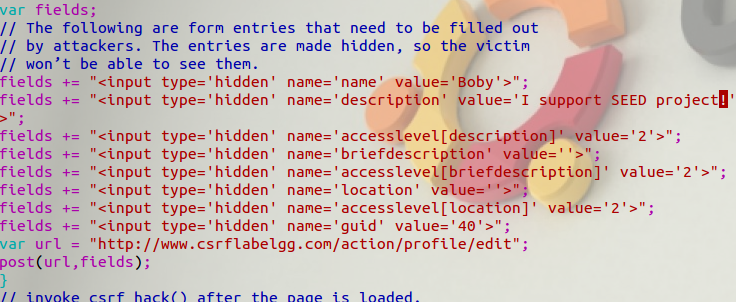
We can run this same action in our malicious site so when Alice clicks on the link Bobby is automatically added to her friend list. When following script is added to /var/www/CSRF/Attacker/index.html file it adds Bobby to the friends list of whomever clicks on the link



As for the POST request, first we need to view the POST call using LiveHTTPHeaders plug-in which allows us to look at the format of the HTTP POST. As the following image shows, using the plug-in when the save button is clicked on the profile page a POST call is made to the server. The bolded text is the information we are interested in passing specifically for this task.



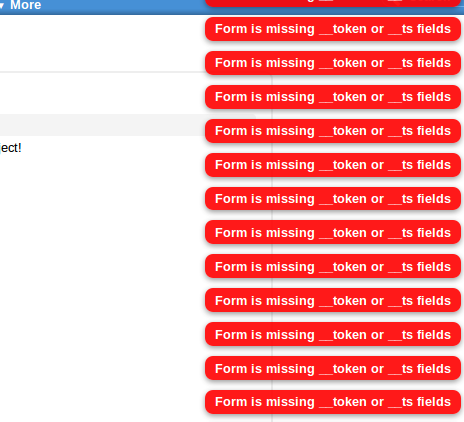
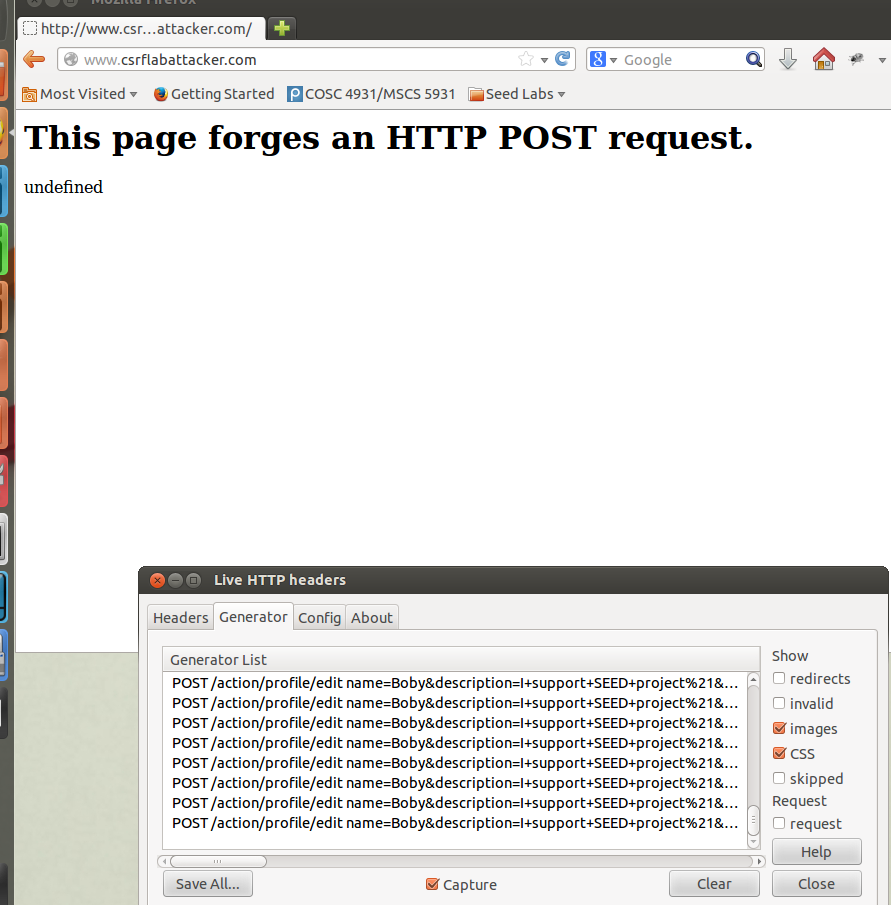
We can replicate this POST action with the desired information using the following script with some minor specific changes. The first description field is where we add the text Alice wants to post on Bobby’s profile. The other field that needs to be verified is the ‘guid’ which needs to be set to the target user. For Bobby we change that to 40.



If the guid field is not set an error will occur which states profile edit failed. If Alice did not immediately know who the target was going to be or if she wanted to target multiple users she could modify the attack to run a GET action first to acquire the user’s guid. For instance Alice could add the user as a friend first and store the guid from the GET action or use the send message action which also has the user ID in the ‘send\_to’ field.



As for implementing countermeasures, the Elgg application uses a secret token method to identify legitimate cross-site requests. The secret token a hash value of several components including a secret value from the server database, a timestamp, a random session string, and the user session ID. The token is checked and verified by the Elgg application upon request.



When the token function is turned on and we run the same attacks from the first part, the malicious site seems to get hung up waiting for the POST or GET request to pass through the legitimate site as seen on the left side of the image. On the right, the legitimate site is returning an error for the missing token. Thus, the request seems to never get through the server and the user’s data is never changed. Also when calling functions on the legitimate site we can look at the token using the LiveHTTPHeaders plug-in which completely changes with every action.