Cross-Site Request Forgery Attack Lab

Assignment 4

CMPT 380 – Computer Software Security

Name: Rushabh Prajapati

Id : 3083048

Task 1: Observing HTTP Request

HTTP GET REQUEST

Click Samy's Profile from Alice, that will generate an HTTP GET request, here we can see that all the data (parameters) of a GET request are attached to the URL.

Similarly, we can see that when an add friend request is generated all parameters such as friend are attached to the URL itself.

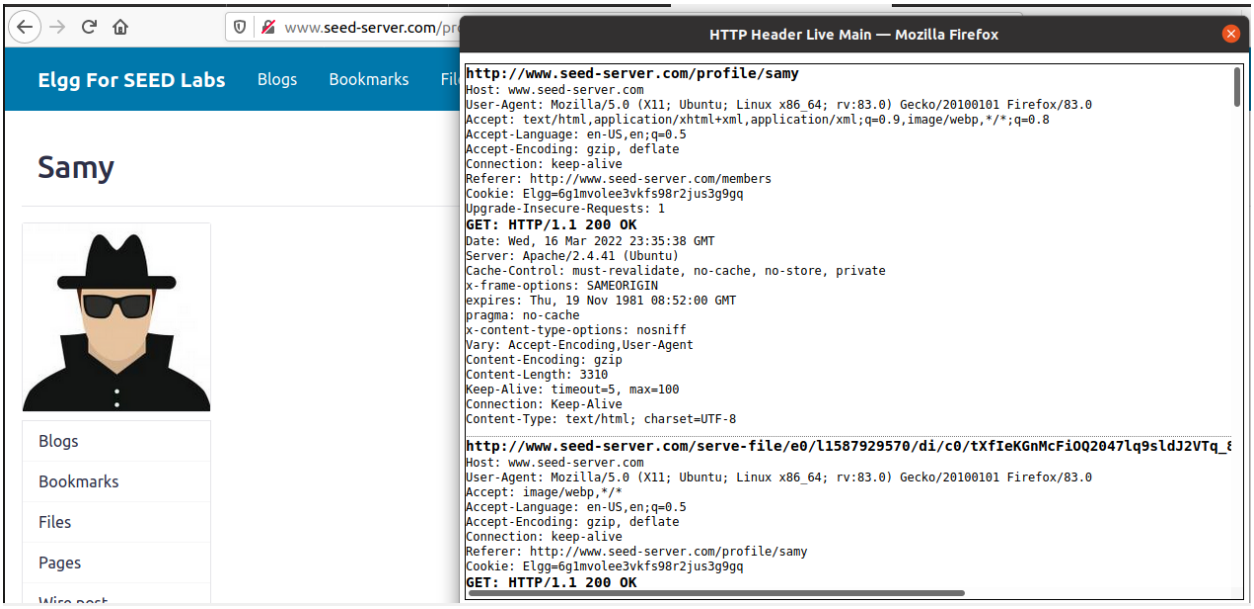


Figure 1 View Profile GET Request

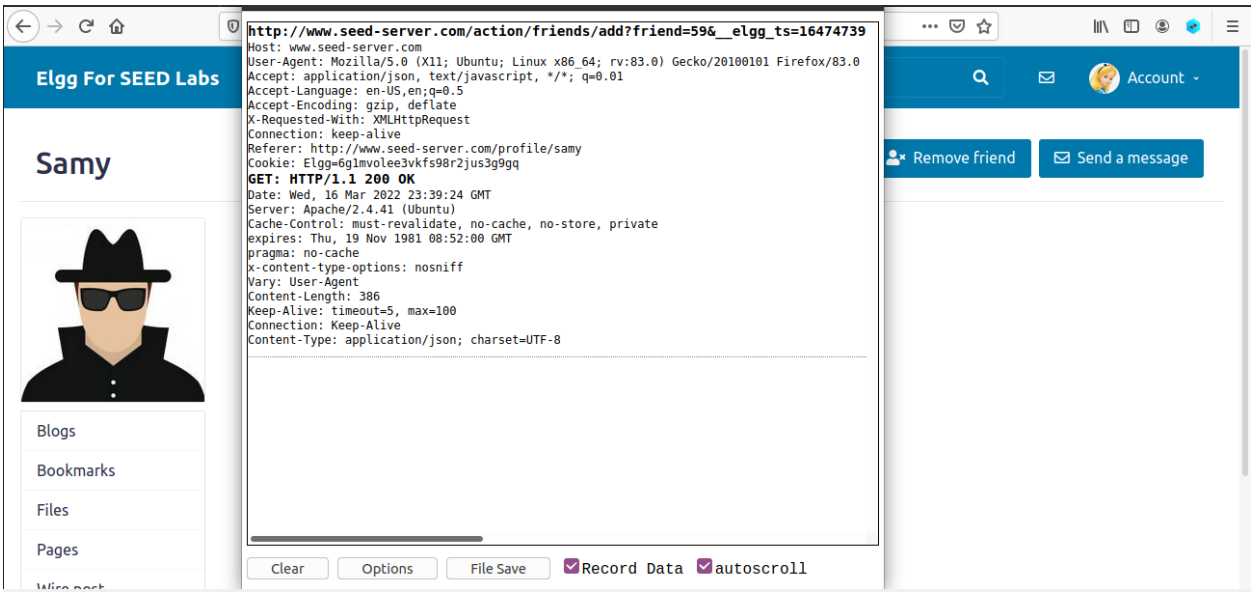


Figure 2 Add Friend HTTP GET Request

HTTP POST REQUEST

As compared to the GET request, in a POST request, the parameters are sent as a body of the request, after the headers.

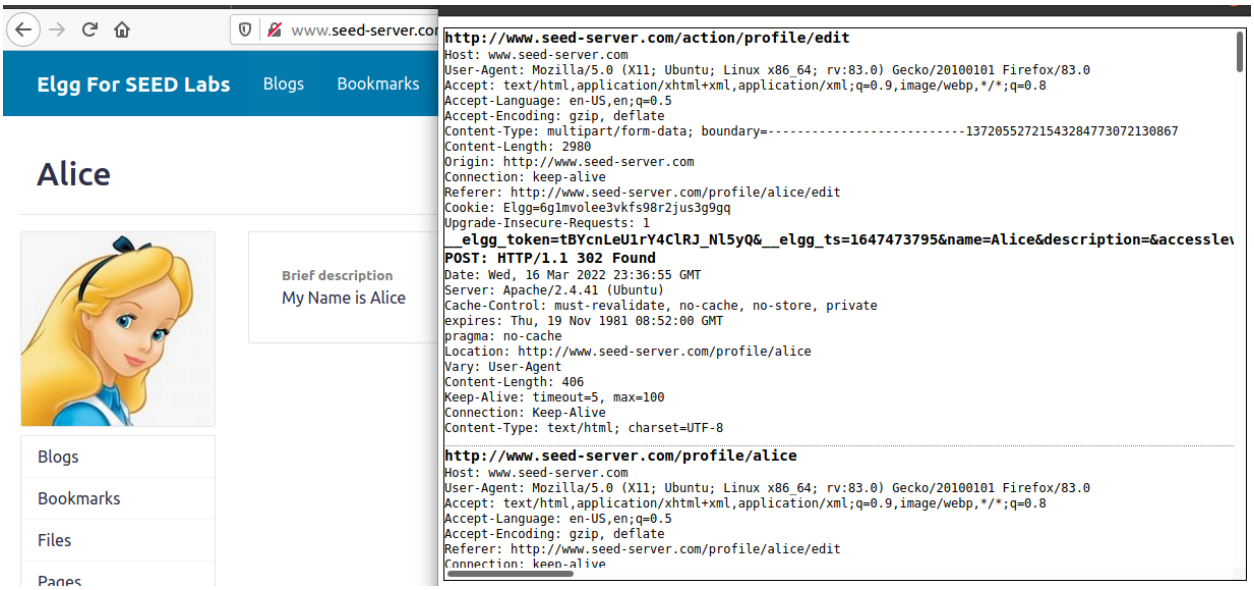


Figure 3 HTTP POST REQUEST

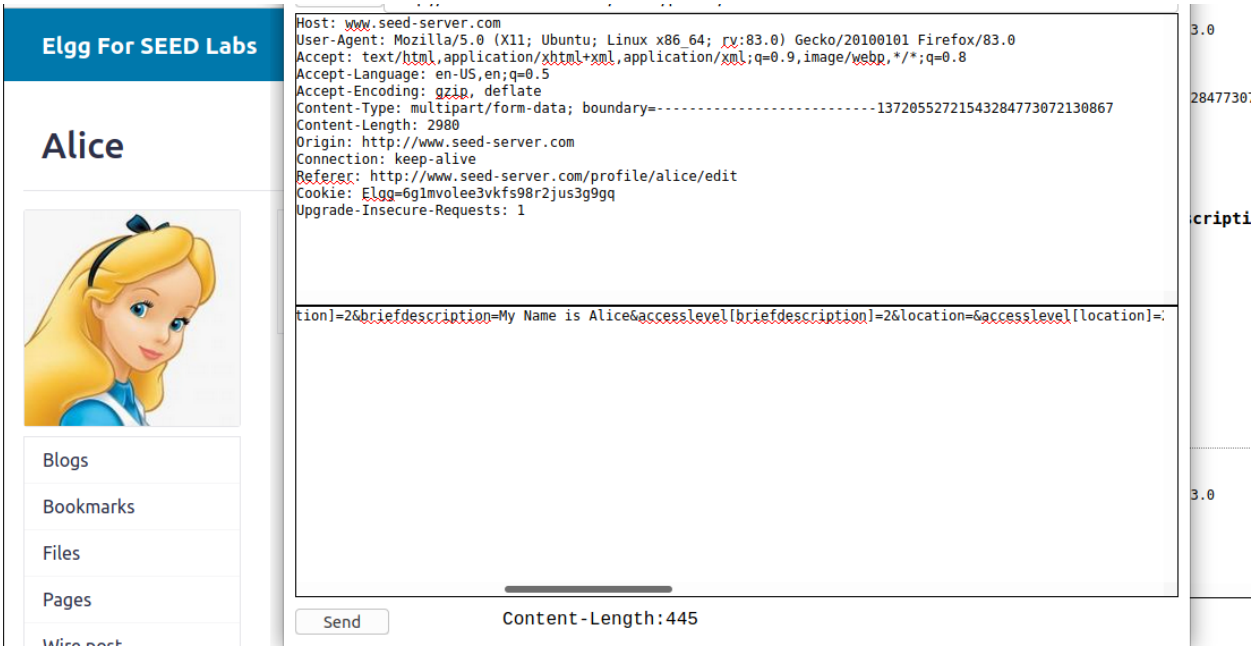


Figure 4 Body part of the POST Request

Task 2: CSRF Attack using GET Request

Get Familiar with the add friend request

In order to do the attack using GET request, we need to understand the how an add friend request is structured. Therefore, Samy's uses his second account Charlie to send a add friend request to Samy.

Below we can see the GET request, in which we can see

friend=58, this is Charlie's GUID.

```
1 http://www.seed-server.com/action/friends/add?-
  friend=58&__elgg_ts=1647475913&__elgg_token=vAJlHlabdtoIVq4O0iQ0RQ&__elgg_ts=1647475913&__elgg_t
2 Host: www.seed-server.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 X-Requested-With: XMLHttpRequest
8 Connection: keep-alive
9 Referer: http://www.seed-server.com/profile/charlie
10 Cookie: Elgg=vg4h5stmrs4bdhm5372vim7l71
11
12 GET: HTTP/1.1 200 OK
13 Date: Thu, 17 Mar 2022 00:12:00 GMT
14 Server: Apache/2.4.41 (Ubuntu)
15 Cache-Control: must-revalidate, no-cache, no-store, private
16 expires: Thu, 19 Nov 1981 08:52:00 GMT
17 pragma: no-cache
18 x-content-type-options: nosniff
19 Vary: User-Agent
20 Content-Length: 392
```

Figure 5 Add Friend Request from Samy to Charlie

URL -> http://www.seed-server.com/action/friends/add?friend=59

Similarly, in Charlie's account, Samy clicks the add-friend button to add himself to Charlie's friend list. When Charlie add's Samy as his Friend, we can catch the HTTP Get request from which we can get the GUID of Samy – 59

This is the URL of Elgg's add-friend request. In the forged request, we need to set the target URL to

http://www.seed-server.com/action/friends/add.

In addition, the add-friend request needs to specify what user ID should be added to the friend list. The friend parameter is used for that purpose. In the captured request, we can see that the

value of this parameter is set to 59; that is Samy's ID (it is called GUID in Elgg).

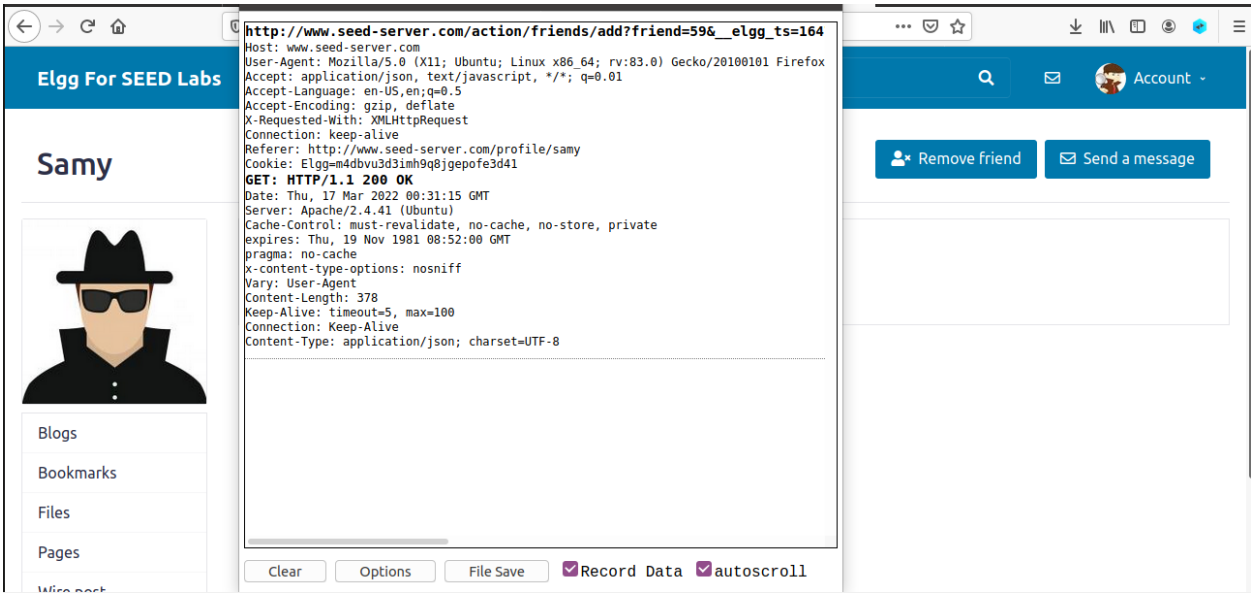


Figure 6 Add Friend GET request

Using the URL and parameters identified from the legitimate add friend GET request, we create the following web request and put it into Samy's profile's About Me field.

```
<img src="http://www.csrflabelgg.com/action/friends/add?friend=59"

alt="image" width= "l" height="l" />
```

The <img> tag will automatically trigger an HTTP GET request. The tag is designed for including images in web pages. When browsers render a web page and see an <img> tag, it automatically sends an HTTP GET request to the URL specified by the <src> attribute. This URL can be any URL, so the request can be a cross-site request, allowing a web page to include images from other websites. We simply use the add-friend URL, along with the friend parameter.

Samy needs to get the victim Alice to visit his malicious web page. Alice does need to have an active session with the Elgg website; otherwise Elgg will simply discard the request. To achieve this, in the Elgg social network, Samy sends a private message to Alice, inside which there is a link to the malicious web page. If Alice clicks the link, Samy's malicious web page will be loaded into Alice's browser, and a forged add-friend request will be sent to the Elgg server. If the attack is successful, Samy will be added to Alice's friend list.

So when alice Visting Samy's malicious link the GET request is triggered automatically which add's Samy to Alice's Friend list.

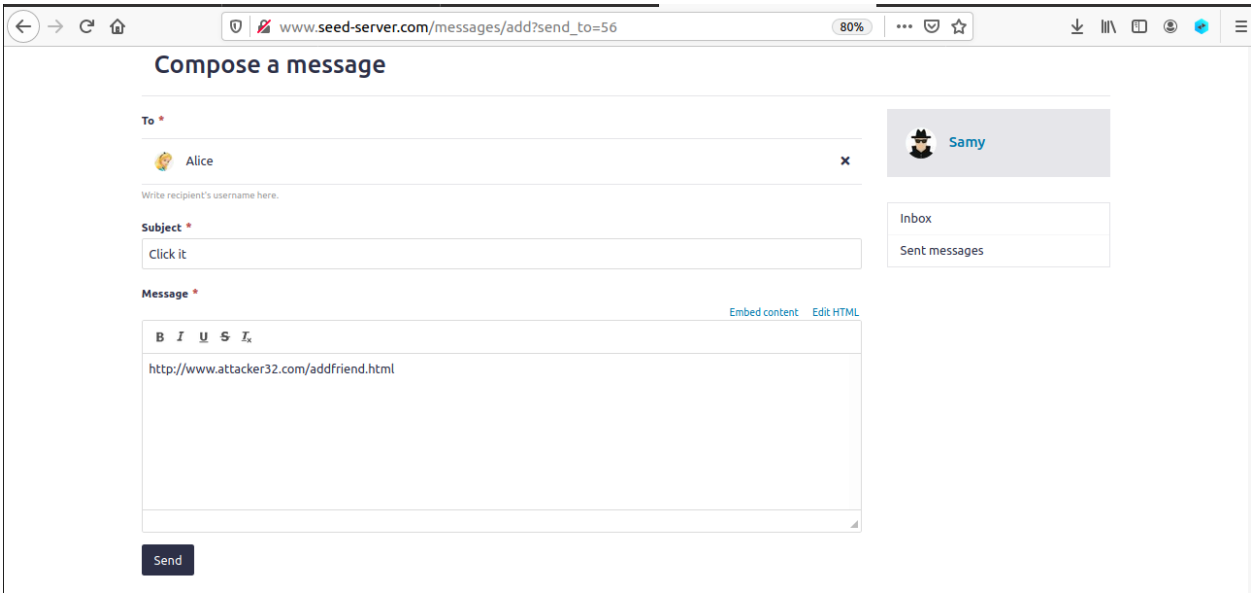


Figure 7 Sending Message to Alice

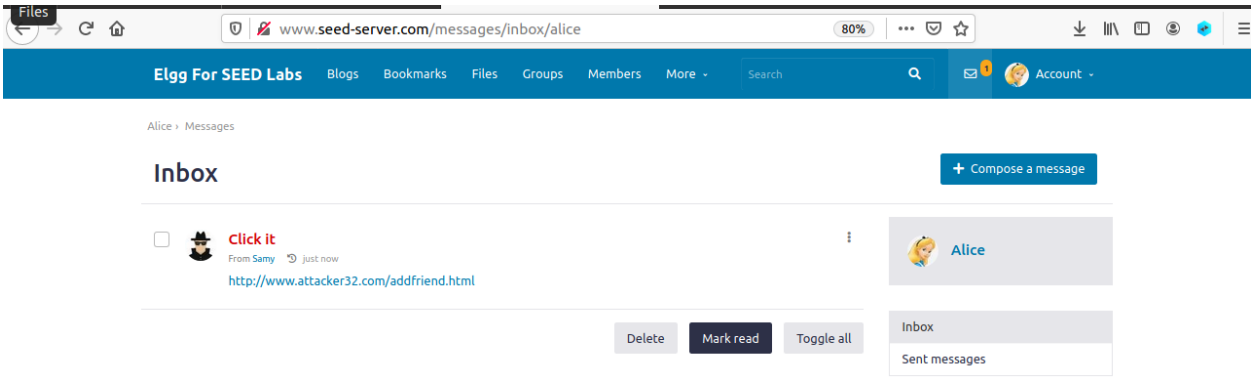


Figure 8 Alice Opening the malicious Link sent by Samy

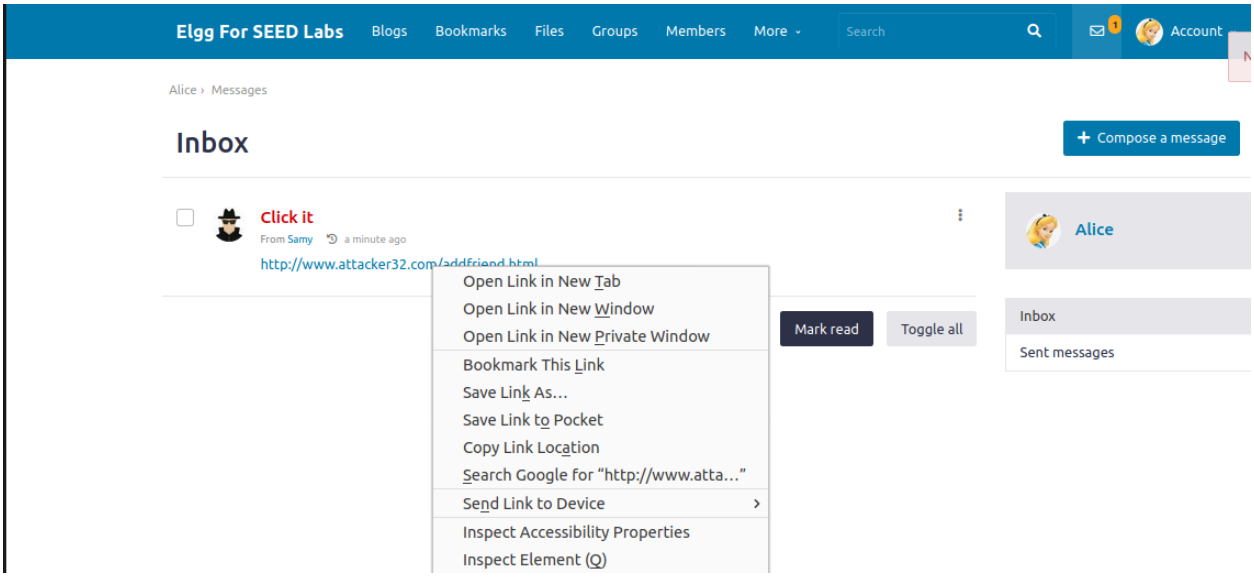


Figure 9 Malicious Link

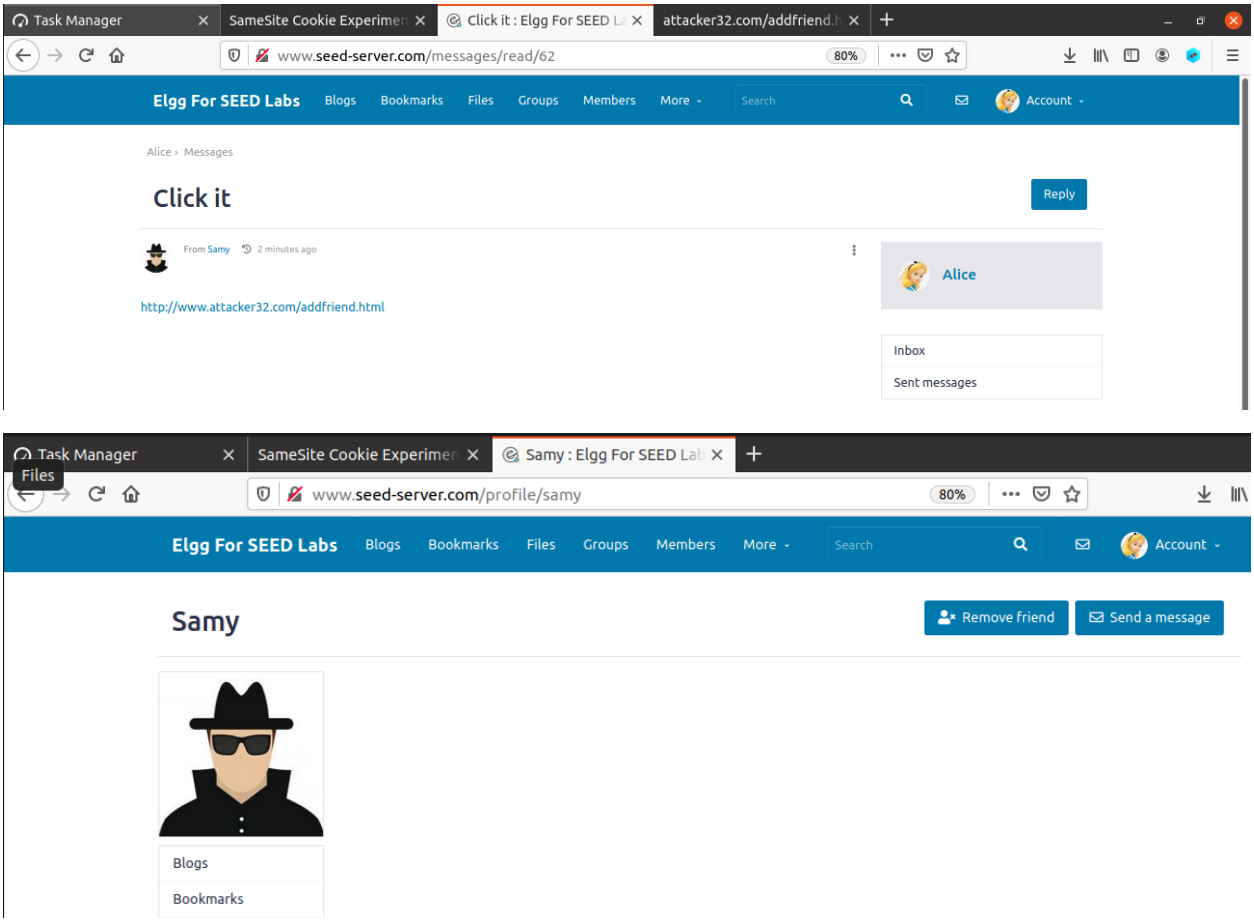


Figure 10 Alice is added to Samy's friend list

Task 3: CSRF Attack using POST Request

POST Request

Elgg's edit-profile service is a POST service, and we are going to use this service as our target.

We will use www.attacker32.com web page to conduct a CSRF attack. When a Alice visits this page while he/she is active in Elgg, a forged HTTP request will be sent from the malicious page to the edit-profile service, on behalf of the victim. If the attack is successful, the victim's profile will be modified with the statement "SAMY is MY HERO" in the victim's profile.

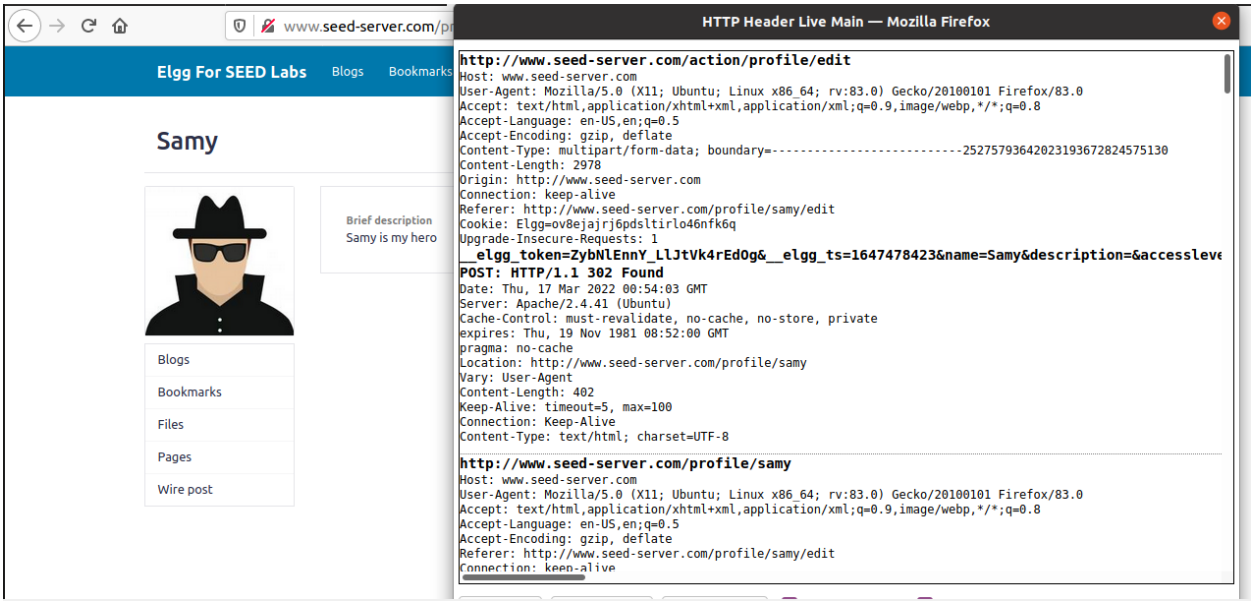


Figure 11 EDIT PROFILE POST REQUEST

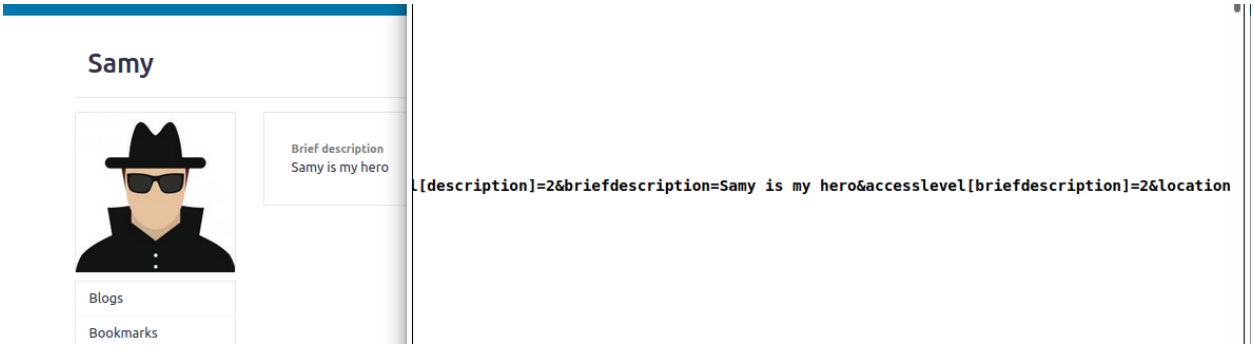


Figure 12 Checking Accesslevel set to 2

Each edit-profile request should include a GUID field to indi cate which user's profile is to be updated. From "HTTP Header Live", we see the value is 59, which is Samy 's GUID. In our attack, this

value should be changed to the victim 's (Alice) GUID. We can find this value by visiting Alice's profile from any account. Once her profile is loaded inside a browser, we can look at the page's source and can see that Alice's GUID is 56, through page_owner:guid.

[},"page_owner":{"guid":56,"type":"user","subtype":"user","owner_guid":56,"container_guid":0,"time

Figure 13 Alice's GUID

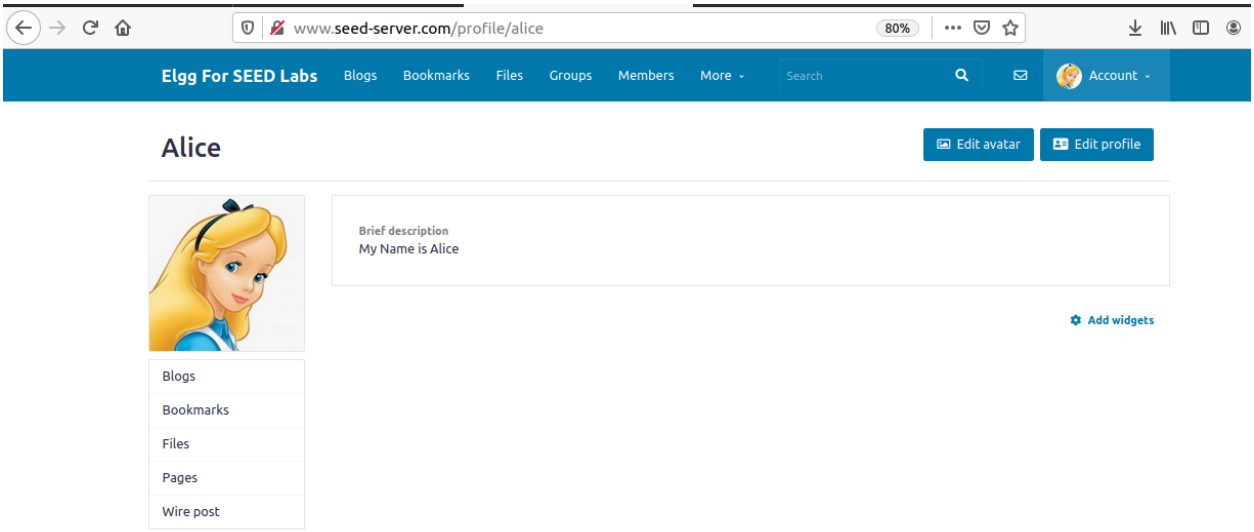


Figure 14 Alice's Current Profile, without the ATTACK

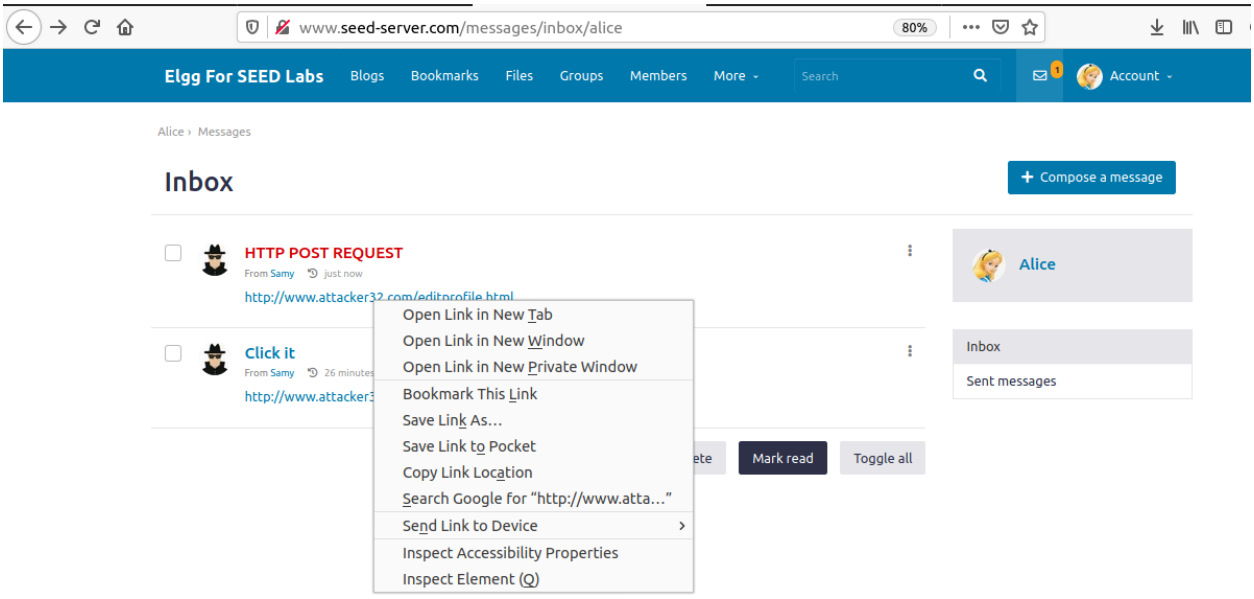


Figure 15 POST Request malicious URL sent to Alice

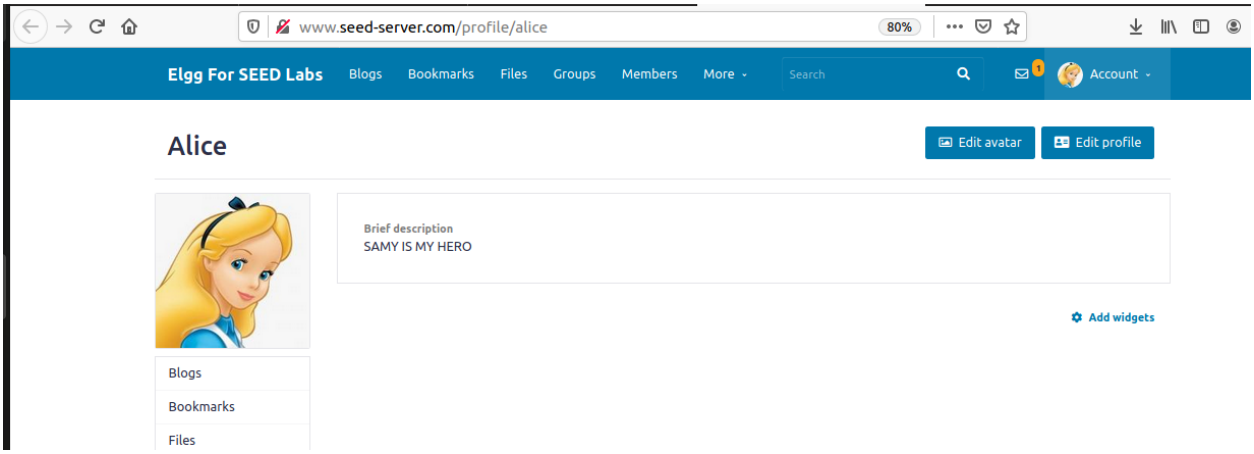


Figure 16 When Alice clicks the Malicious link, the POST request is triggered and her profile is Updated with the new Description

Answer 1:

We can get Alice's GUID by visiting Alice's profile and do View Page Source in which he can find the page_owner key from which he can find Alice's GUID. Thus, solving the problem of getting Alice's user id (guid) to work properly in the forged HTTP request.

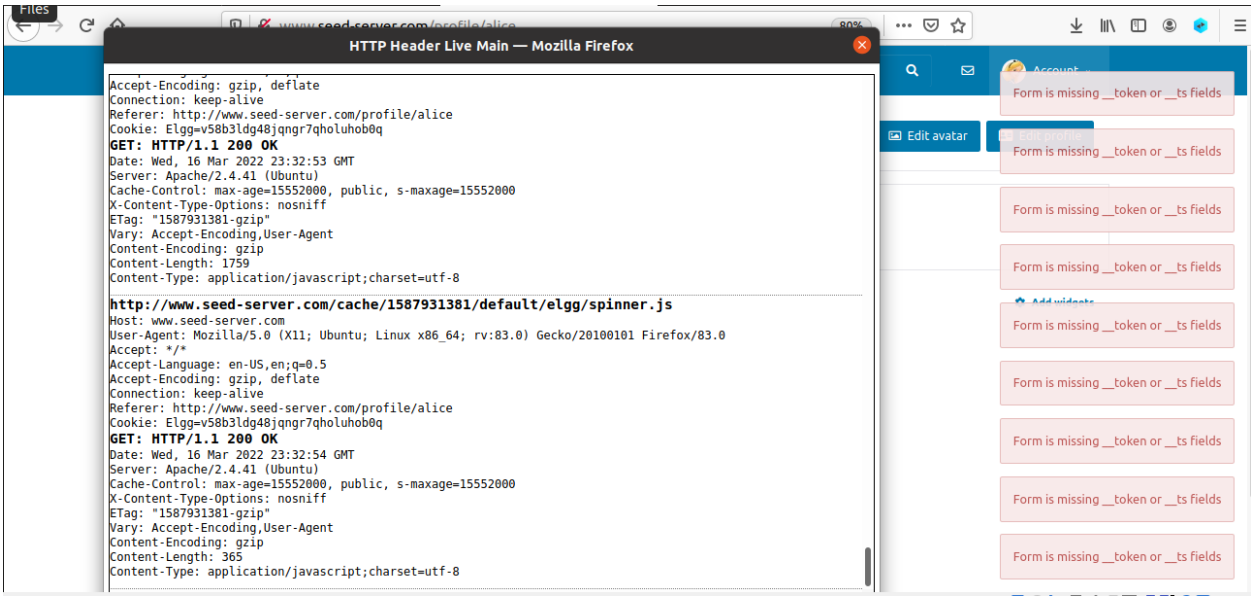
Answer 2:

No, Boby cannot launch the CSRF attack to modify the victim's elgg profile, because in this current attack, we modified our POST request HTML file with Alice's name and Alice's GUID, as this will be different for every user, for which Boby does not know their name and GUID beforehand therefore he cannot launch the CSRF attack to modify the victim's Elgg profile.

4 Lab Tasks: Defense

Task 4: Enabling Elgg's Countermeasure

To turn on the countermeasure, we need to go to the /var/www/CSRF /Elgg/vendor/elgg/elgg/engine/classes/Elgg folder and find the function validate () in the Csrf.php file.



Attack will be successful or not.

As you can see, Alice's profile can no longer be changed due to the validation cookie. And because the request fails, the web page will be refreshed, and after the refresh is requested again, the web page is frantically refreshing.

To succeed, attackers need to know the values of the secret token and timestamp embedded in the victim's Elgg page. Unfortunately, browser's access control prevents the JavaScript code in attacker's page from accessing any content in Elgg's pages.

Task 5: Experimenting with the SameSite Cookie Method

1) During Link A, when we perform GET and POST requests, it's showing that all cookies were displayed (cookie-normal, cookie-lax, cookie-strict) for both requests (GET, POST) because the request is a same-site request!
   In Link B, there are differences, when we perform a GET request or POST request, for the GET request the cookies what will be displayed is (cookie-normal, cookie-lax), but for the POST request, the cookies that will be displayed is one kind of cookie which is (cookie-normal), and that was because the request is a cross-site.

2) Cross-site request forgery (CSRF) attacks rely on cookies attached to any request to a specific origin, regardless of who makes the request. For instance, if you go to a malicious website. Then it can make requests to your blog, for example. Your browser will gladly accept the cookies linked with it. If your blog doesn't validate those requests carefully, then it isn't good. The example could prompt them to do things like delete posts or add their own.

3) When you set SameSite to Strict, your cookie will only be delivered in the context of a first-party website. In user terms, the cookie will only be sent if the cookie's target site is the same as the one now displayed in the browser's URL bar. If the promo_shown cookie is set to the following value: Set-Cookie: promo_shown=1; SameSite=Strict