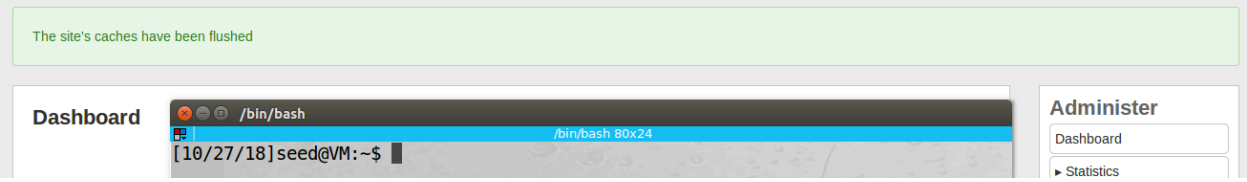
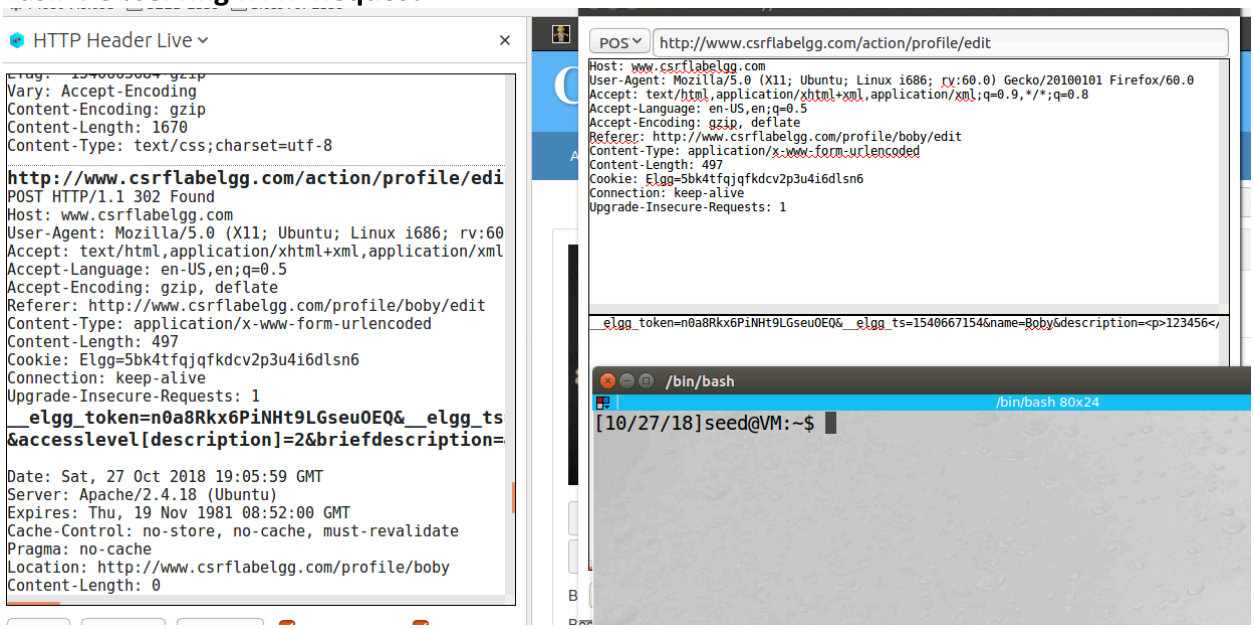
CSE643 Lab9
Yishi Lu
10/29/2018

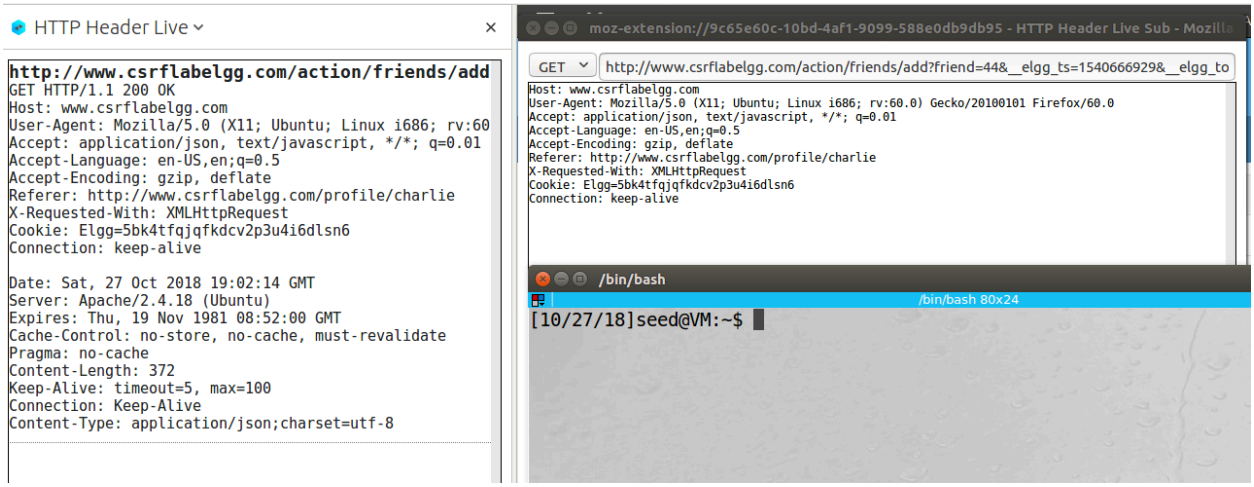


Before we start this lab, we flush the cache of Elgg

Task1: Observing HTTP Request



screenshot1, changing profile of Boby's account which is done by HTTP POST request



screenshot2, we make a HTTP GET request on Elgg by adding a friend to Boby

Observation and Explanation:

In this task, we use a tool "HTTP Header Live" to capture GET and POST request on Elgg. As screenshot1 shows, when we change profile description of Boby's account, this change is done by a HTTP POST request. And parameters are used in this request are shown in following:

```
__elgg_token=n0a8Rkx6PiNHt9LGseuOEQ
&__elgg_ts=1540667154
&name=Boby
&description=<p>123456</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
```

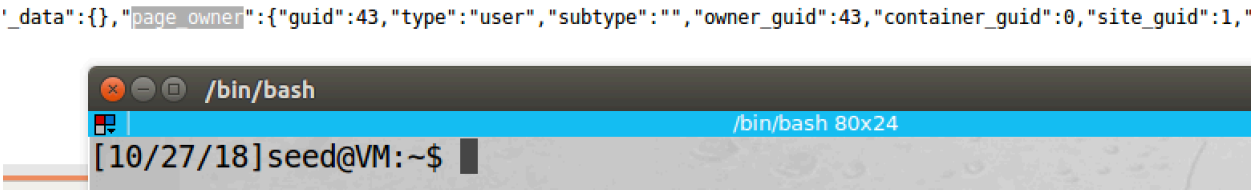
The first two is secret tokens of Elgg, they are used to prevent CSRF attack. The following are profile attributes, which include name, description, location, and so on. If we enter the attributes field in the profile form, the value will be here (after "="). If we do not enter anything, it will be empty. Moreover, the accesslevel controls visibility of the corresponding attribute, 2 means visible to everyone.

As screenshot2 shows, we add a friend to Boby which done by a HTTP GET request. And parameters are used in this request are shown in following:

```
friend=44
&__elgg_ts=1540666929
&__elgg_token=Tma1KD2YOjZPD5St3n4msA
```

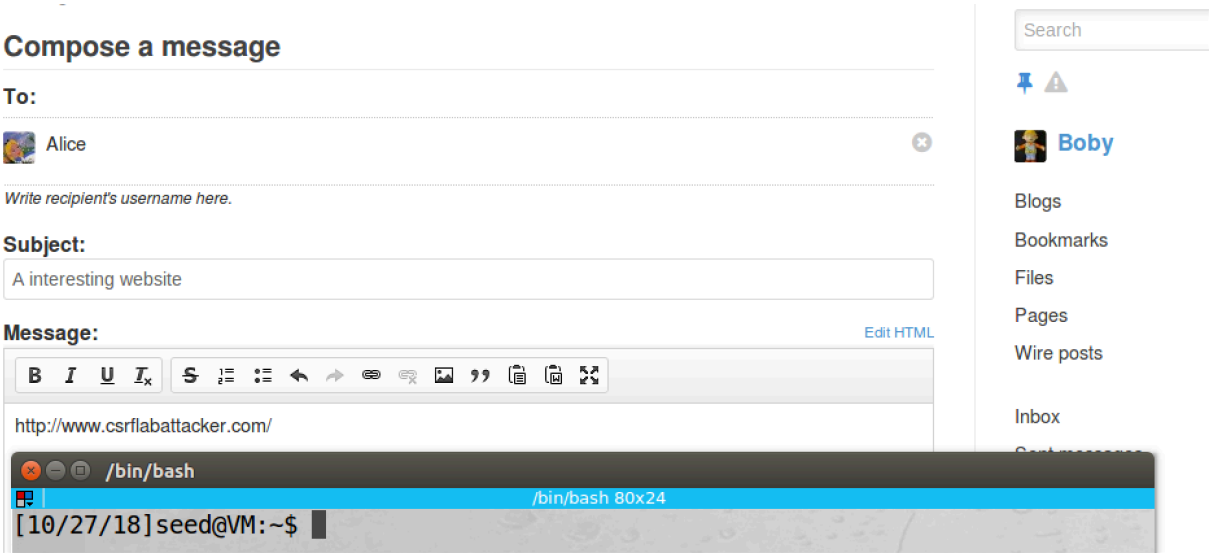
There are three parameters in this request, first is the target guid. And the rest two are the secret tokens of Elgg.

Task2: CSRF Attack using GET Request

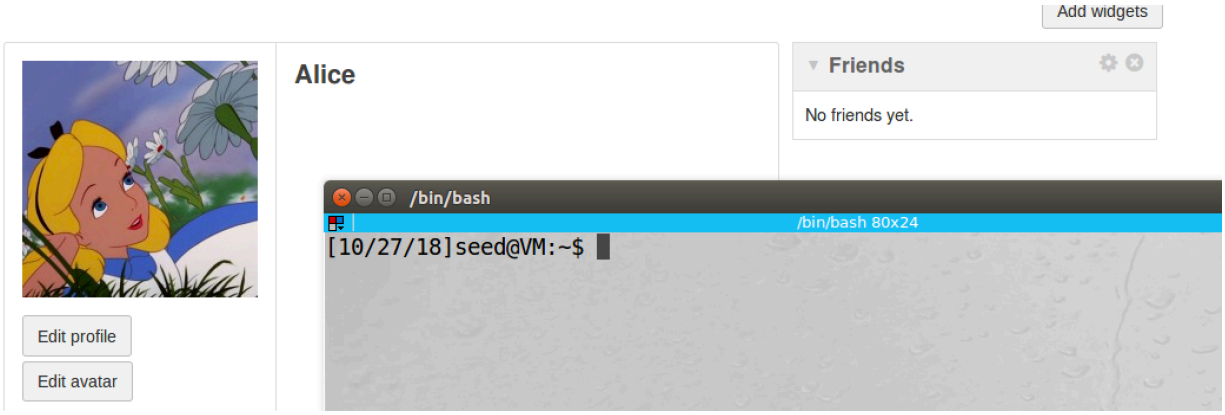


screenshot1, to add Boby as friend to Alice, we need to find out Boby's guid. So we can go to Boby's profile and open the view page source, and then we search page_owner tag, we find Boby's guid which is 43

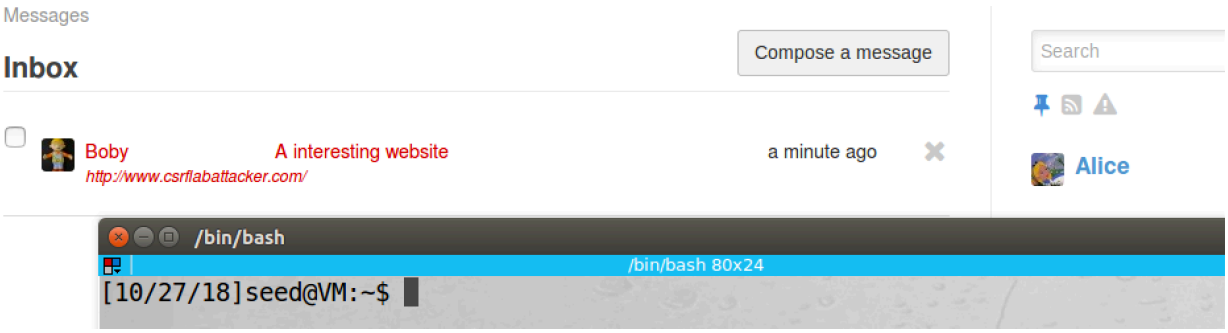
screenshot2, we construct a website, which contains an image tag. When a user visits this website, the image tag will automatically trigger an GET request to Elgg.



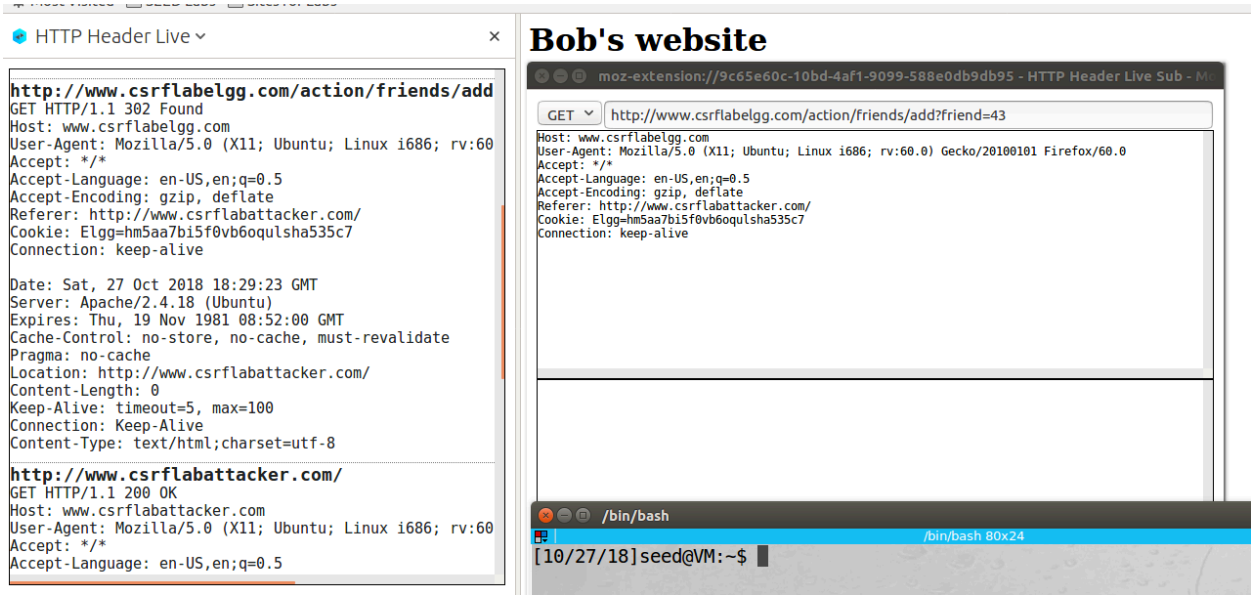
screenshot3, we switch to Boby's account, and we send a message to Alice from Boby's account, and this message contains the malicious website.



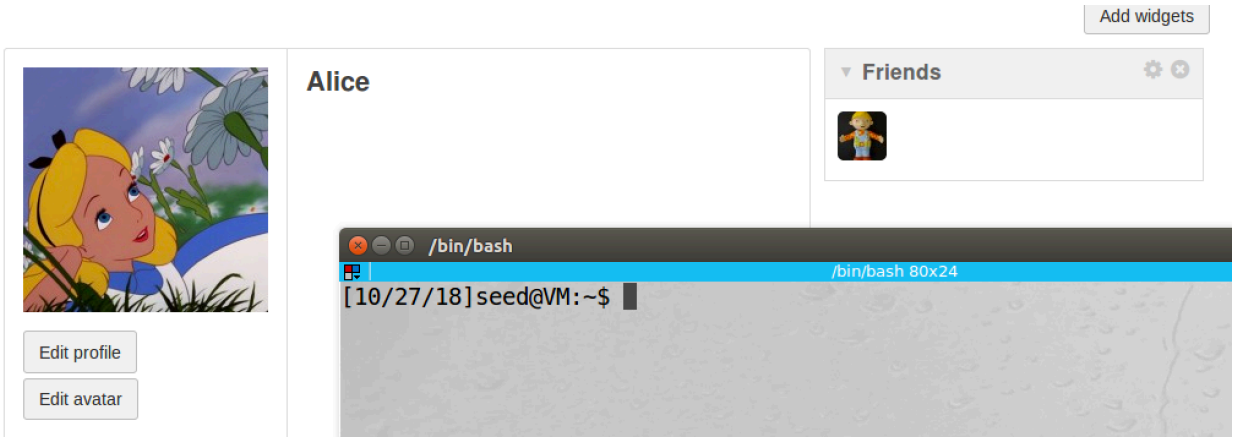
screenshot4, we switch to Alice's account. At this point, Alice has no friend



screenshot5, we see Boby's message on Alice account



screenshot6, after click the link, we go to Boby's website, and we use inspection tool to capture the GET request, which will add guid 43 as friend



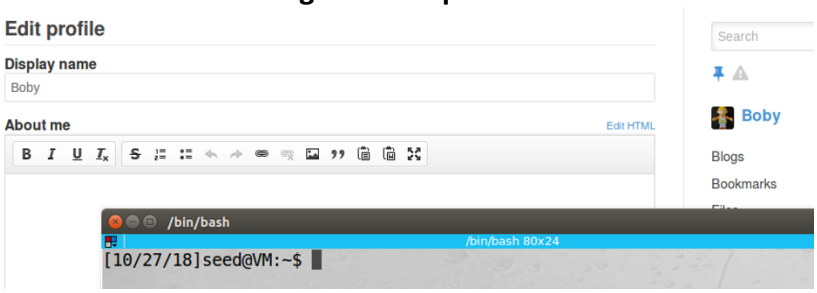
screenshot7, after we come back and check Alice's friend list, Boby is added in the list. So our attack succeeds

Observation and Explanation:

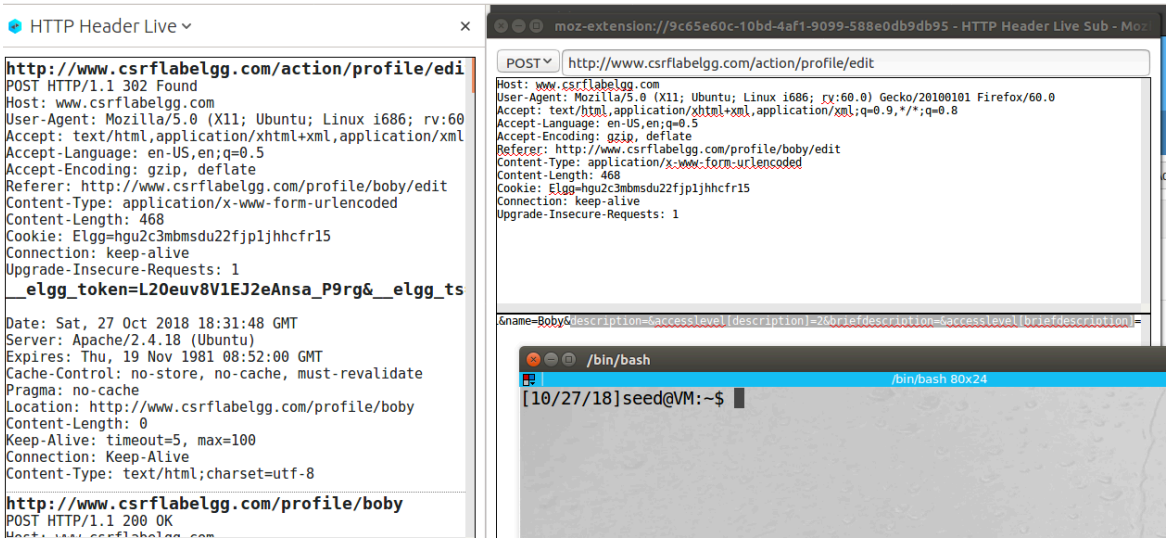
In this attack, we perform CSRF attack by GET request. When a request coming from website A and go to website A, it is same-site request. When a request coming from website A but go to website B, it is cross-site request. Requests coming from same website are trusted. But requests coming from other sites should not be trusted. However, when browser send a request to a server, it will attach the same cookie to both cross-site request and same-site request. So for server, it is hard to distinguish them. If the server treats them as same, then it will cause big problem. Cross-site request can be a forgery request which called CSRF. In this task, we perform CSRF attack by HTTP GET request. GET request attached data on the URL, so if we put the URL in the src attribute of image or frame, then the GET request will be automatically triggered when user visits the page. This attack involves three parties, a victim user, a target website, and a malicious website. When the victim user has an active session on the target website, and the user also visits the malicious website. Then the malicious website can send a forgery cross-site request to the target website. And the target website will process such request if it has no countermeasure. In our case, the victim user is Alice, the target website is Elgg, and the malicious website is csrflabattacker.com (owned by Boby). Our goal is to add Boby (attacker) to Alice friend list without her consent. We first need to construct the malicious website; for the request to add Boby as friend, we need Boby's guid. To find Boby's guid, we can login to Boby's account; and then in his profile page, we right click, there will be a menu, and we click on View Page Source; in View Page Source, we search page_owner, Boby's guid will be there (screenshot1). Then we construct the website. As screenshot2 shows, the website includes an image tag. when the victim user visits this page, the HTTP GET request will be automatically triggered. The URL is specified in the src attribute. This URL will send cross-site request to Elgg, and it will add ID-43 to the victim's friend list. Because this is not an image, we set its width and height to 1, so user is very hard to see it on the page. Then we login to Boby's account, and we send the website address to Alice (screenshot3). Afterwards, we switch to Alice's account, we first check her friend list, it is empty (screenshot4). And then we check the message from Boby (screenshot5). After we click the link, we jump to the malicious website, the inspection tool captures the HTTP GET request, it is same as the one in the malicious

website (screenshot6). After we come back to Elgg, we check Alice's friend list again, Boby is in the list (screenshot7). So our attack succeeds.

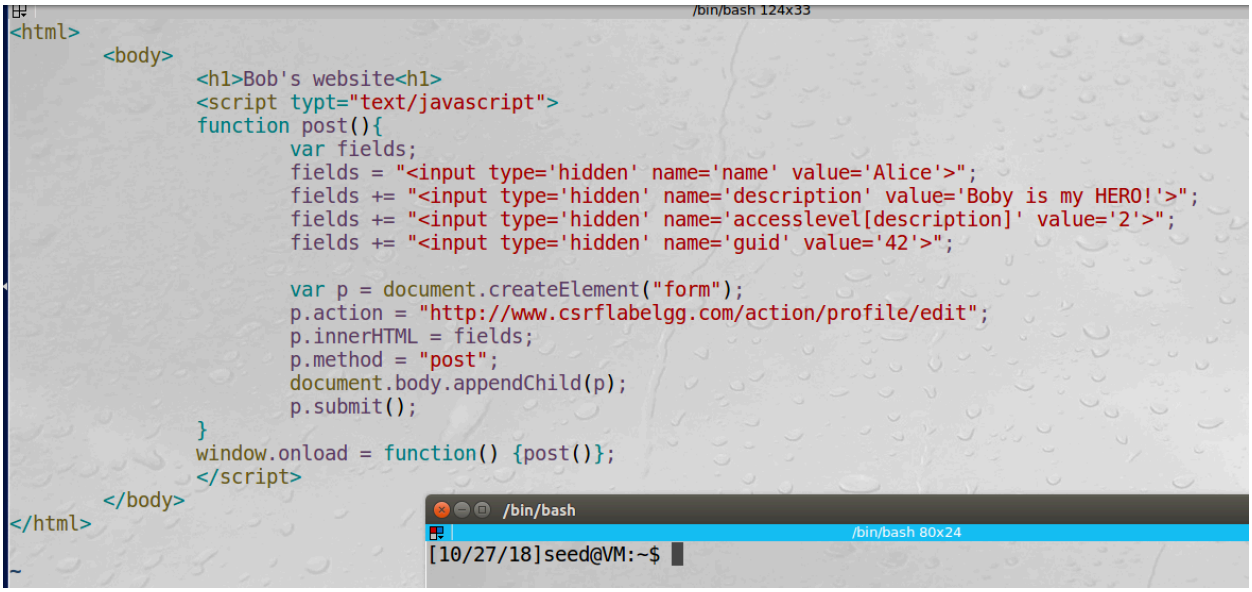
Task3: CSRF Attack using POST Request



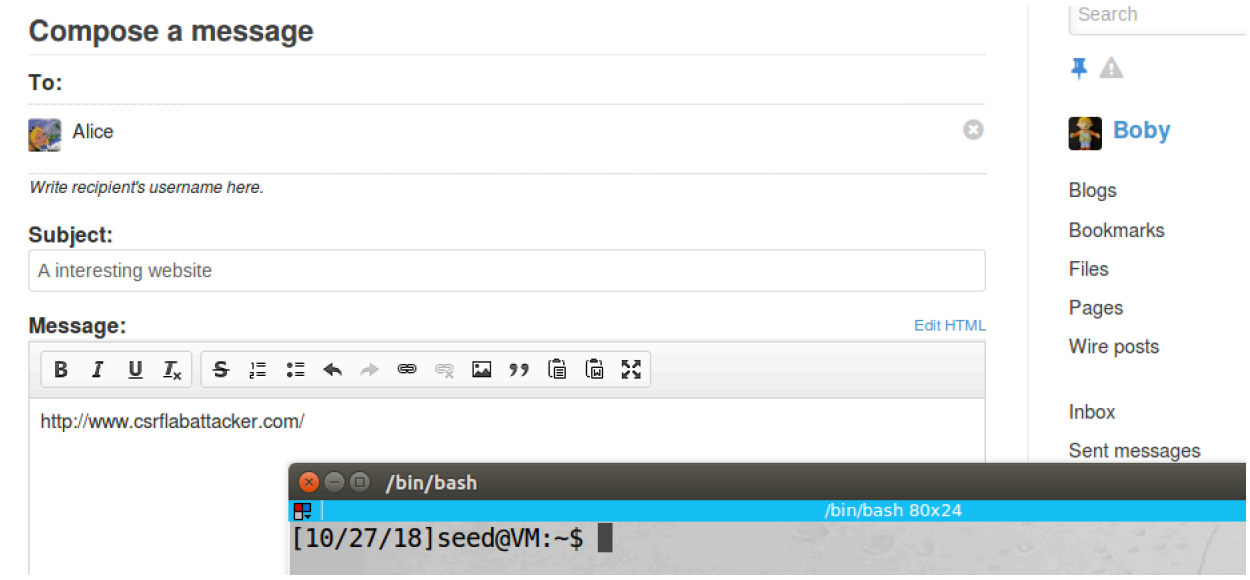
screenshot1, we login to Boby's account



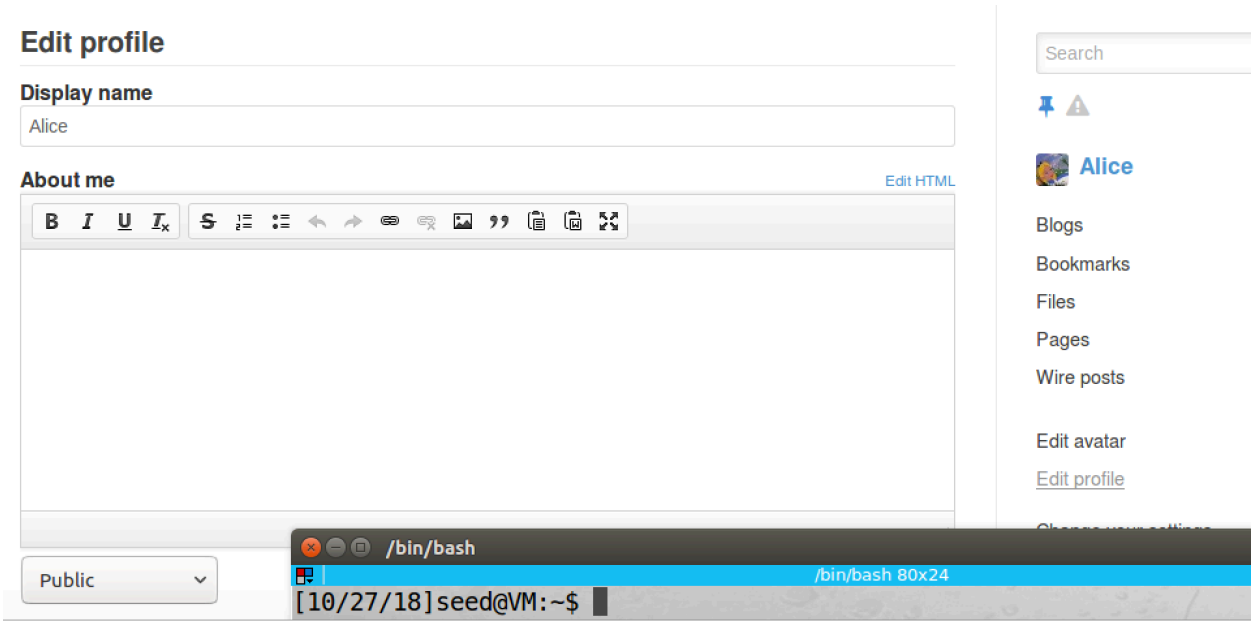
screenshot2, we modify Boby's profile, so we know which attributes need in the malicious website



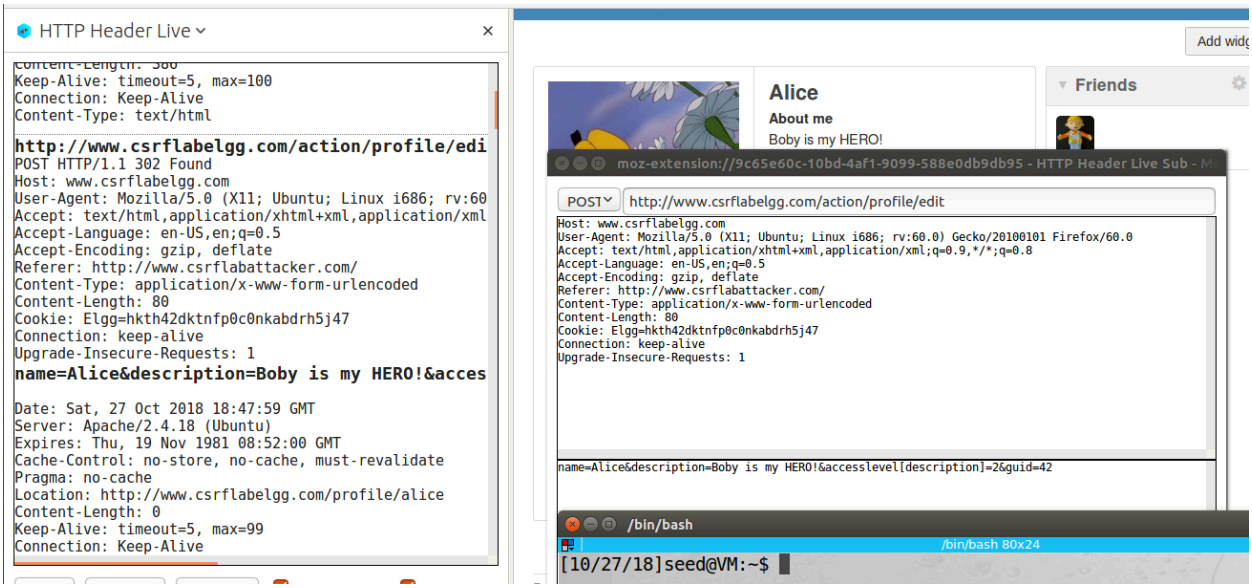
screenshoht3, we construct the malicious website



screenshot4, we send a link of the malicious website to Alice



screenshot5, we switch to Alice account, we first check the description of Alice, it's empty



screenshot6, after Alice open the link, we jump to the malicious website. After we come back, Alice's description becomes "Boby is my HERO!". We also use inspection tool captures the POST request, it contains attributes and parameters provide in the malicious website. So our attack succeeds.

Observation and Explanation:

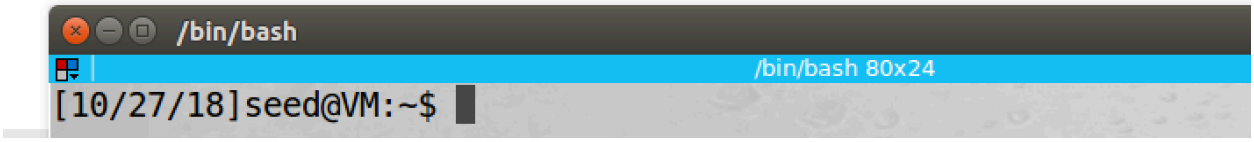
In this task, we perform CSRF by HTTP POST request. The whole mechanism is similar as last attack. The only different is the method, last task we use HTTP GET request, and this task we use HTTP POST request. Because POST request does not attach data on the URL, so we cannot put the request on the src attribute of image or frame. Instead, we write a JavaScript

program, which is included in the malicious website. Once the victim visits the malicious website, the JavaScript program will generate a form which contains the malicious command, and the JavaScript program will send the form to the target website automatically via POST request. In this task, we want to change Alice's Elgg account's description (change to "Boby is my HERO!") without her consent. We first login to Boby's account, and then we change Boby's profile, so we know the attributes in the profile (screenshot 1 and 2). Now we construct the malicious website. The attributes we need which include name, description, accesslevel[description], and guid. We keep the name field as Alice. We change the description value to "Boby is my HERO!". We also change the value of accesslevel[description] to 2, so anybody can see the description, and we also need Alice guid (we can visit Alice profile page and open page source to find page_owner tag, Alice's guid is there which is 42). Moreover, in the JavaScript program, we need to make these input fields to be hidden, so when victim visits the malicious website, he/she will not notice the form. Then we create a from, we set the destination to be Elgg profile edit page; then we add these input fields into the form, we also set the method to be POST. Finally, we also call function submit(), so the form will be submitted automatically when victim visits the website (screenshot3). After constructed the malicious website, we send the link of the malicious website to Alice (screenshot4). Then we switch to Alice's account, we first check her profile, there is no any description (screenshot5). Then we open the message from Boby, and we click the link. Afterwards, we transfer to the malicious website, after we come back, the description of Alice becomes "Boby is my HERO!" (screenshot6); we also use inspection tool to capture the POST request, and it contains the attributes and parameters which are same as those provided in the malicious website. So our attack succeeds.

Question1:

We first login Boby's account, and then we go to Alice's profile. Then we right click, there is a menu, we select View Page Source. In the View Page Source, we search page_owner, then we can find the guid of Alice.

,"page_owner":{"guid":42,"type":"user","subtype":"","owner_guid":42,"container_guid":0,"site_guid":1,"



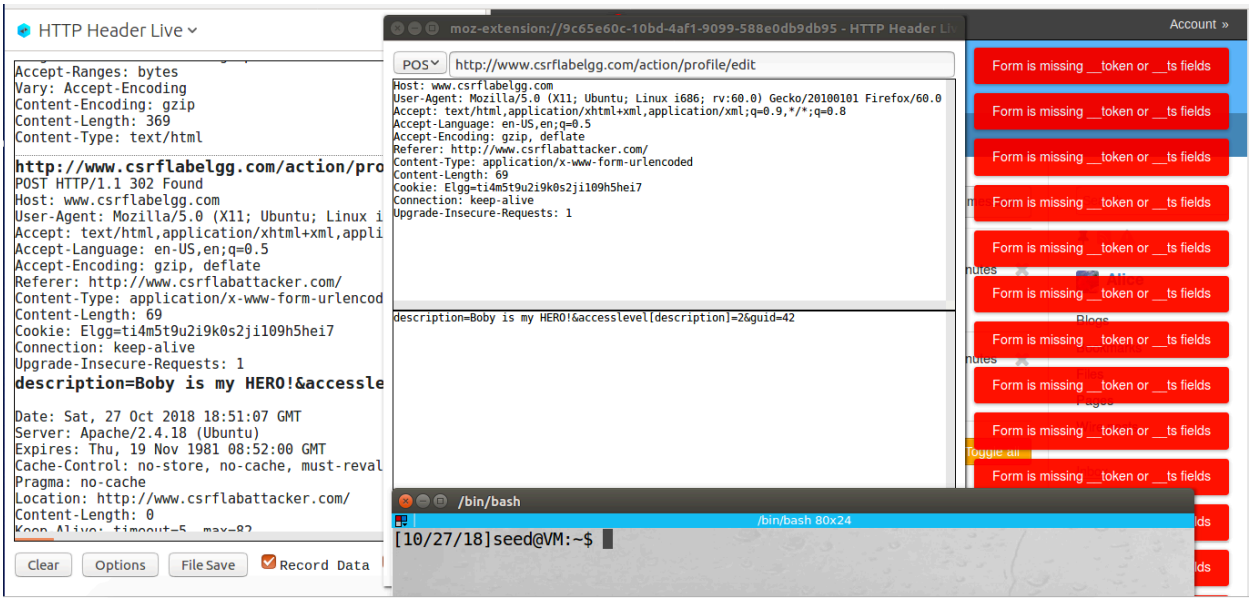
Question2:

If the victim does not have an active session on the target website, then the attack will not work. Because website use session cookies to decide the coming request from a client is trusted or not; if the victim has no active session (no valid session cookies), then the website will discard this request, or it will redirect user to login page and ask for login credential. So in this situation, Boby's attack will fail.

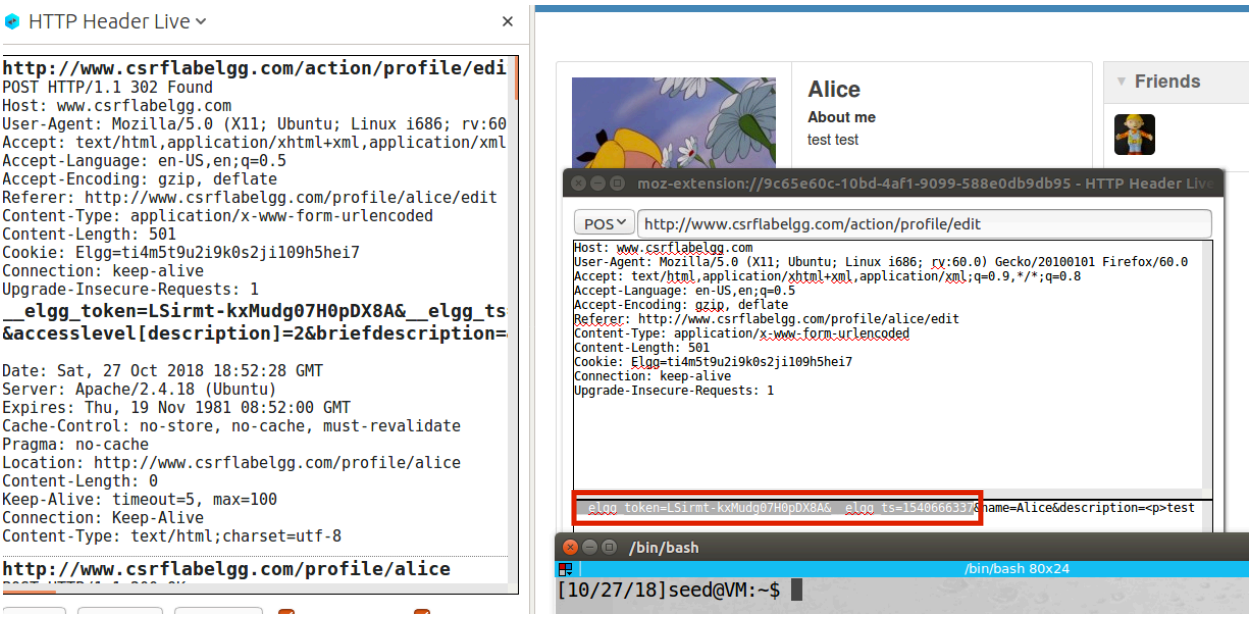
If the victim has an active session on the target website, then Boby's attack will succeed as we did in task2 and 3.

Task4: Implementing a countermeasure for Elgg

screenshot1, we turn on the countermeasure of Elgg



screenshot2, the attack fails, it shows error message: missing token



screenshot3, if we change Alice account's description in a valid way, we are successful to change the description. And we use inspection tool to capture these secret tokens

Observation and Explanation:

There are several ways for server to prevent CSRF. For Elgg, they choose to use secret token. Elgg embeds two secret values, _elgg_ts (time stamp) and _elgg_token (token) in all its page. The secret token of Elgg is a MD5 digest, and it is generated base on four pieces of information: the site secret value, timestamp, user session ID, and a randomly generated session string. For attacker, these information are very hard to guess. Moreover, the browser's access control also prevents the JavaScript code to access any content in Elgg's page from attacker's page. So the attacker cannot get and send these secret tokens to the target website. When there is user action/request, Elgg function call validate_action_token() to validate token. If the token is not present or invalid, the action/request will be denied. As screenshot 1 and 2 show, after we turn on the countermeasure, our attack fails because we do not have valid secret tokens. As a result, our forgery request will be discarded by the server. Then we modify Alice account description is valid way (change the description by login Alice account). We are successful to change the description; and we also use inspection tool to capture the POST request, these two secret tokens: _elgg_ts and _elgg_token are captured (screenshot3).