**Kaleb Alstott**

**LAB 5**

**11/01/2024**

**LAB SET UP**

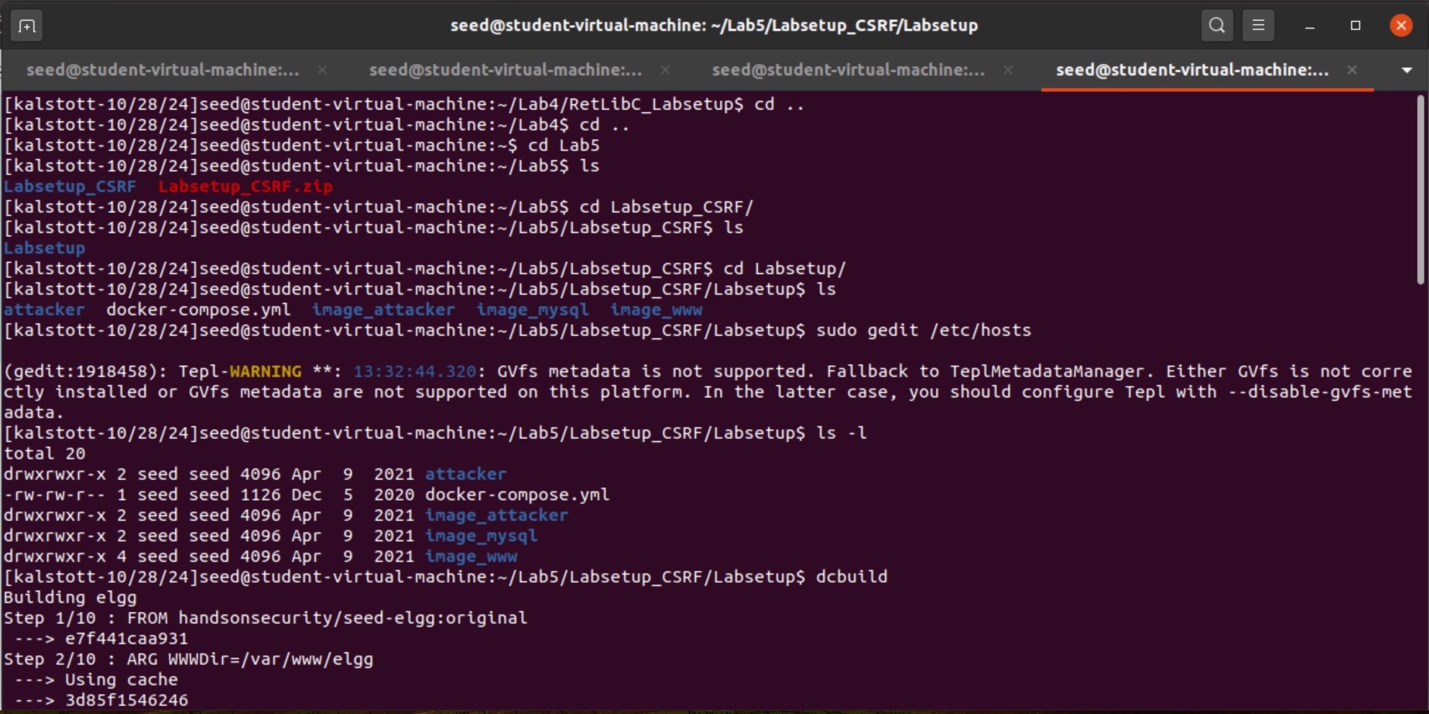
A screenshot of a computer

Description automatically generated

The first website is the vulnerable Elgg site accessible at www.seed-server.com. The second website is the attacker’s malicious web site that is used for attacking Elgg. This website is accessible via www.attacker32.com. The third website is used for the defense tasks, and its hostname is www.example32.com. It is important to note here that we will be using containers to set up this lab environment.

DNS Configuration: First, we had to map our IP addresses to the host websites used in the lab. We use 3 specific websites (hosts) which are:

10.9.0.5 [www.seed-server.com](http://www.seed-server.com) - vulnerable Elgg site  
10.9.0.5 [www.example32.com](http://www.example32.com) - attacker’s malicious website  
10.9.0.105 [www.attacker32.com](http://www.attacker32.com) - used for the defense tasks



In order to start the lab, we must create our container using Docker Compose commands. We use a shortcut for docker-compose build, dcbuild, which builds the container images specified in the docker-compose.yml file. After this step is done we want to use the shortcut for docker-compose up, dcup, which starts up all the containers specified in the docker-compose.yml

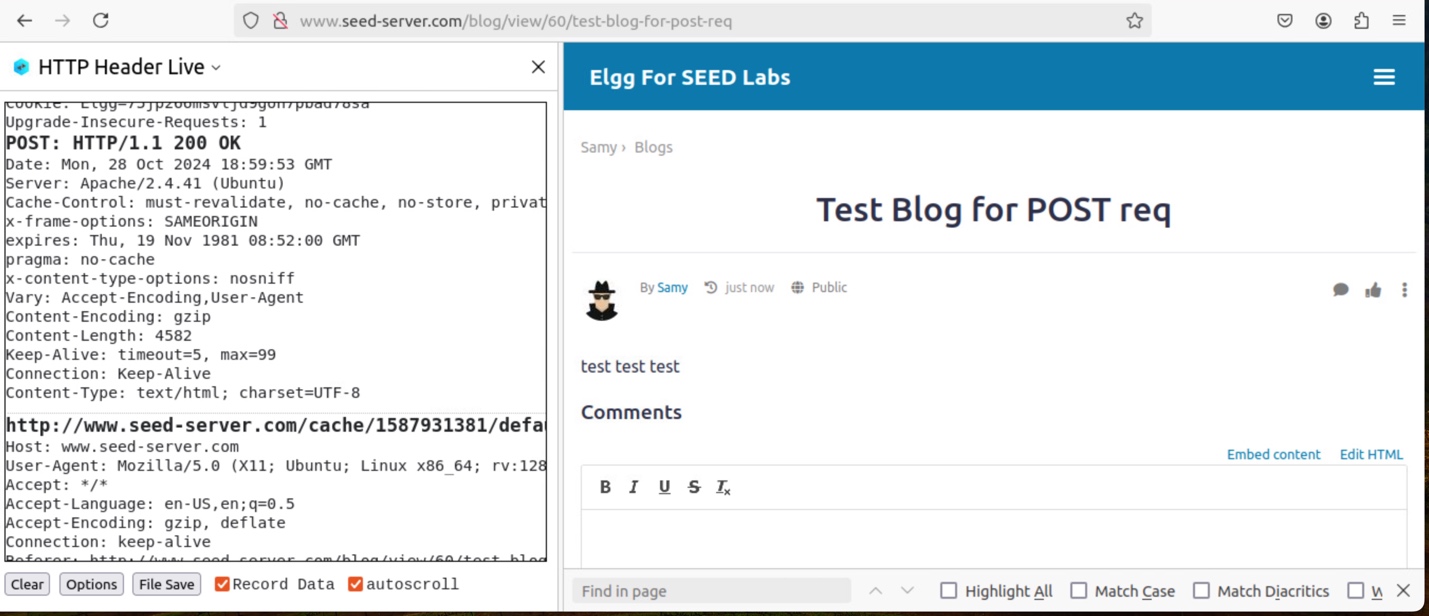
// Aliases for the Compose commands above  
$ dcbuild # Alias for: docker-compose build  
$ dcup # Alias for: docker-compose up

We are now ready to rock and roll for this assignment!

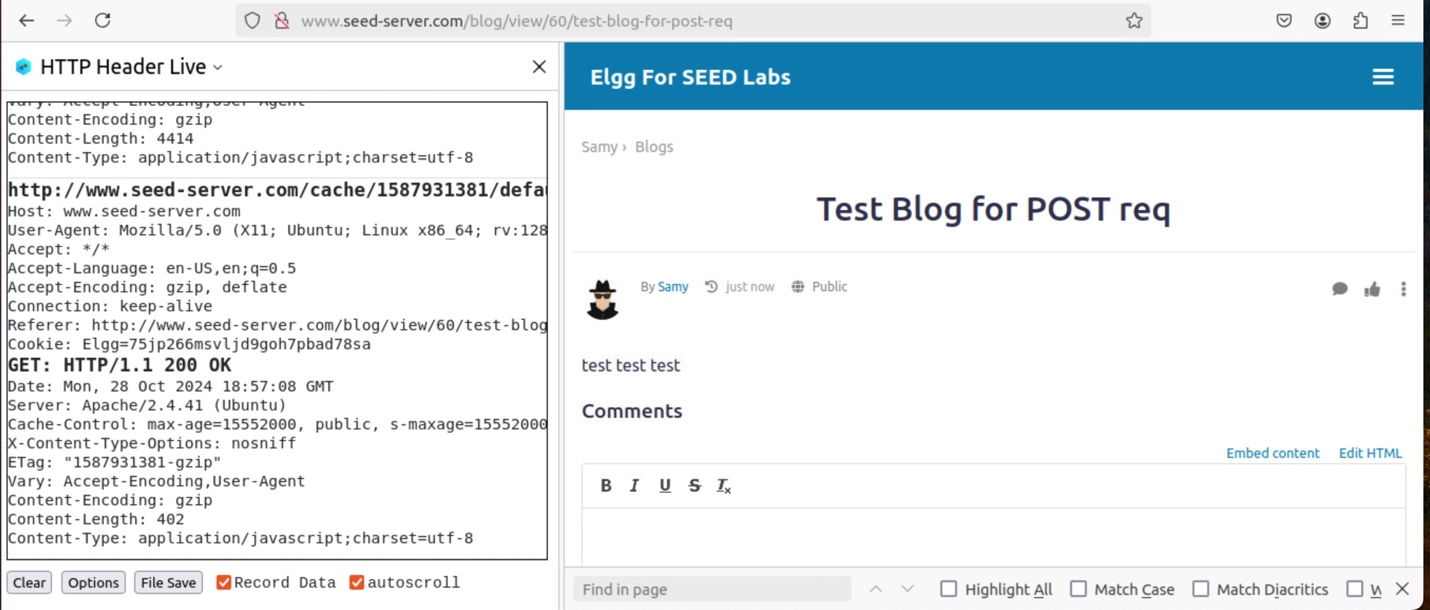
A screenshot of a computer

Description automatically generated

**TASK 1 (Optional)**



**POST:** In this optional task we were asked to get familiar with the HTTP Header Live that we downloaded. To get familiar with the tool we had to capture an HTTP GET request and an HTTP POST request in Elgg. For my POST request I had simply edited Samy’s profile to say, “test test test” and saving this text to Samy’s profile. By doing so our POST request was accepted and as you can see above had posted to Samy’s profile as expected. By editing the profile and adding text to the blog we can form the post request due to us adding content to the webpage. Hints why it is a post request because we are posting to the webpage.



**GET:** For our GET request this one was super easy. Most things you do in the webpage is a GET request. Switching to a different tab such as the blog, settings, messages, etc. will for a GET request. The reason on why this happens because we are requesting to access something on the webpage forming the GET request. If we are not adding to the webpage with a POST request, we are using GET to traverse through the website as well as traversing through profiles. A GET request is us simply asking to move from the current page we are on without adding information to it.

**TASK 2 CSRF Attack using GET Request**

A screenshot of a computer program

Description automatically generated

In this task we were challenged with conducting a CSRF attack using a GET request. Our task is for Samy to add himself to Alice’s friend list on the Elgg website, which is our vulnerable social network site, without Alice’s permission directly accepting our friend request.

To start this task, we first had to find our addfriend.html file which is apart of our malicious webpage which is [www.attacker32.com](http://www.attacker32.com). When you go to this webpage you are accessing our malicious webpage which will host both the GET and POST request. For our example and task here we are going to be using the first link in our malicious webpage which is the “Add-Friend Attack” link.

A screenshot of a computer

Description automatically generated

By using our HTTP Header Live extension, we had to do some playing around and research into what the actual request sent was when Alice tries to add someone as a friend on Elgg. By using this tool, we were able to record the actual request sent when Alice tried to add someone as a friend. This URL is reused and modified so that when the user clicks the link in our malicious website the user will have a new friend named Samy added to their profile. The modifications we did to the URL was the /action as well as the actual value that is used for add?friend. The /action in the URL is used for task to update, modify, etc. what is in the URL. The value we used for the add?friend is actually Alices user ID. By doing so we had formed the URL to add Samy as a friend to Alices account without her supposedly knowing that she had done so by clicking on our link in our website.

A screenshot of a mail

Description automatically generated

In order to get Alice to go to our malicious website to click on our link we had to construct some type of phish or smish message to the user Alice. I had chosen to use Charlie as the attack vector and entrance to this situation. Logged in as Charlie I constructed a possible phishing email that is bullet proof to get Alice to click on our URL to our malicious website.

Per usual Alice had fallen for the phishing attack and clicked the link to get to our malicious website.

A screenshot of a computer

Description automatically generated

We need user to click on “Add-Friend Attack”.

Once Alice was in our website, she had clicked the link “Add-Friend Attack” which is our super-secret link to perform the GET request attack.

Like discussed in class this attack did not work due to modern web browsers having built in countermeasures and extra protection for these types of attacks.

To show that I had the correct URL for this attack and that it would have been successful I had copied the URL to add Samy as a friend to Alices profile into the webpage and then had to refresh Alices page showing that without the modern web browsers protection this attack would have worked.

To clarify I did some research and wanted to add that with “recent version of Firefox, <img> tags won’t trigger the request”.

A screenshot of a computer

Description automatically generated

**TASK 3 CSRF Attack using POST Request**

**A computer screen with white text

Description automatically generated**

In this task we were challenged with conducting a CSRF attack using a POST request. Our task is for Samy to be able to modify Alice's profile by adding the phrase "Samy is my Hero!!!!" to Alice’s profile description. Ideally, we want this to happen without any type of action from Alice besides going to our malicious website and clicking on our attack link.

To start this task, we first had to find our editprofile.html file which is apart of our malicious webpage which is [www.attacker32.com](http://www.attacker32.com). When you go to this webpage you are accessing our malicious webpage which will host both the GET and POST request. For our example and task here, we are going to be using the second link in our malicious webpage which is the “Edit-Profile Attack” link.

**A screenshot of a computer

Description automatically generated**

By using our HTTP Header Live extension, we had to do some playing around and research into what the actual request was when Alice tried to edit her profile on Elgg. By using this tool, we were able to record the actual URL and request sent when Alice tried to modify her profile. This URL is reused and modified so that when the user clicks the link in our malicious website the user will have a new cool text in their brief description about whatever we want. The modifications we did to the URL was the /action which allows us to perform the task such as editing or updating the profile, as well as/profile/edit which we had to edit Alice specifically out because this attack doesn't require a specific user ID in the URL. We also had to record the values used her such as Alices GUID as well as the text we wanted to enter in Alices webpage and finally the name of who we are attacking which is Alice. It is important to note here that the value of 2 that is being used in line 14 is to makes the description public.

In this code we create a form with hidden inputs that will act as Alice filling out her profile. When she visits the page the form submits automatically sending the POST request to Elgg to update Alices profile.

**A screenshot of a computer

Description automatically generated**

In order to get Alice to go to our malicious website to click on our link we had to construct some type of phish or smish message to the user Alice. I had chosen to use Charlie as the attack vector and entrance to this situation. Logged in as Charlie I constructed a possible phishing email that is bullet proof to get Alice to click on our URL to our malicious website.

Per usual Alice had fallen for the phishing attack and clicked the link to get to our malicious website.

**A screenshot of a computer

Description automatically generated**

We need user to click on “Edit-Profile Attack”.

Once Alice was in our website, she had clicked the link “Edit-Profile Attack” which is our super-secret link to perform the POST request attack.

**A screenshot of a computer

Description automatically generated**

As you can see here when Alice clicked on our link, we were successfully able to conduct the attack and modify the brief description for Alices profile without her direct consent to do so. We were successful in our POST request attack.

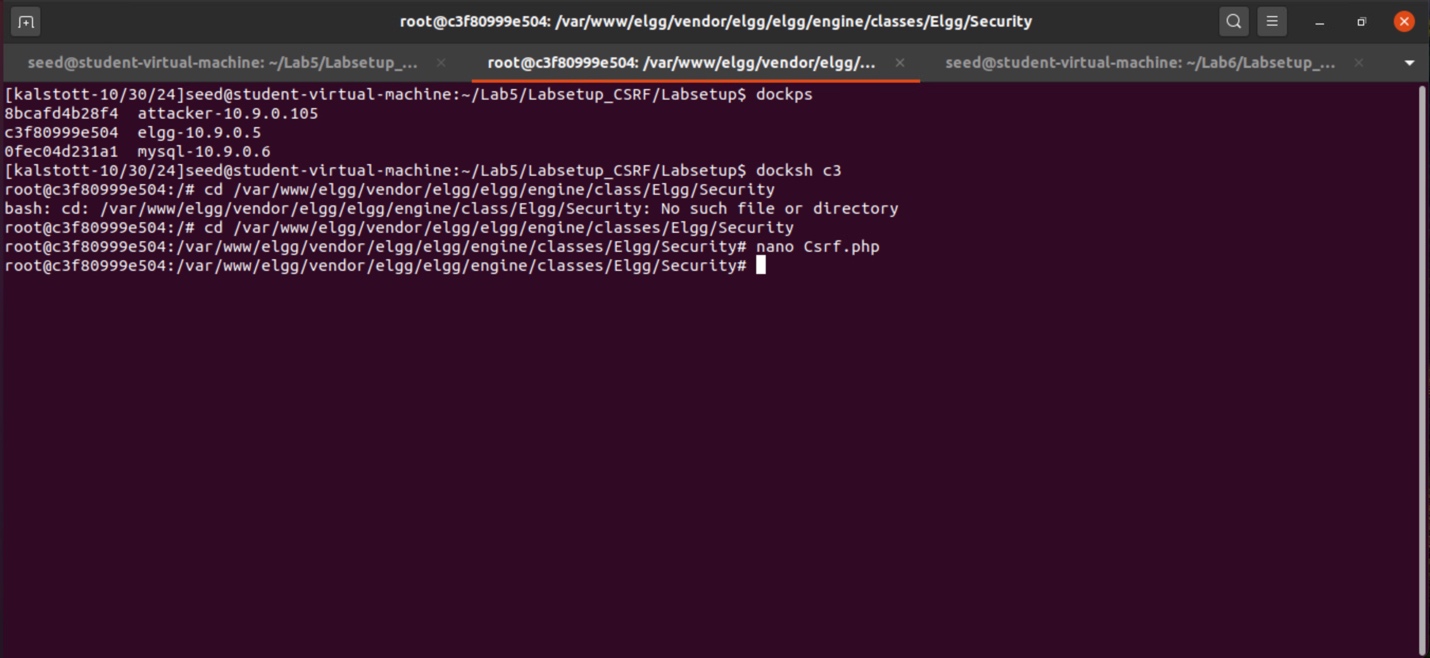
**Question 1: The forged HTTP request needs Alice’s user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice’s user id. Boby does not know Alice’s Elgg password, so he cannot log into Alice’s account to get the information. Please describe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

Boby could determine Alice’s GUID without logging into her account by checking Alice's public profile URL on Elgg. It is possible that some websites still use this, and the URL would show Alices GUID. It is also possible that some websites will also show the GUID value of a user in other specific webpages that are open to all users. With trial and error of being on Alices profile and clicking through the specific webpages to Alice you may be able to record the GUID value for her account.

**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.**

Yes, Boby will be able to the conduct and launch the CSRF POST request attack to anybody who visits his malicious web page. When a user visits the malicious page their browser will include the session cookies with the request needed for this attack. Elgg will authenticate the request as it will be coming from a user who is currently logged in. This will allow Boby to conduct this attack and modify the profile of any user who is logged into Elgg and visits our malicious site. This will allow Boby to attack users without identifying the GUID beforehand like we did in our attack above.

**TASK 4 Enabling Elgg’s Countermeasure**

****

In this task our goal was to see what our attack would look like with the CSRF countermeasure enabled. To do so we first had to follow the instructions and correctly get into the Elgg container. We had to use the command dockps to find out the ID of the container. After doing so we had to use the command docksh to start a shell on the Elgg container. Our next step was directing ourselves into the correct container folder which is /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security folder. Once in we remove the  
return statement from Csrf.php to ensure that we have enabled the CSRF countermeasure.

**A screenshot of a computer

Description automatically generated**

In this step we had to use the command-line text editor nano to properly comment out or remove the return function we had initially added in our setup (in our case it was already there). By doing so this enables the CSRF countermeasure for our attack.

**Note:** In our setup, we added a return at the beginning of this function, essentially disabling the validation.

**Note:** I asked in class if we could use gedit for this part of the task and I think I was incorrect and that we would have to use nano. I think with us being in the container gedit is not supported since it is used for mostly GUIs. In comparison to nano which is a command-line text editor.

**A screenshot of a computer

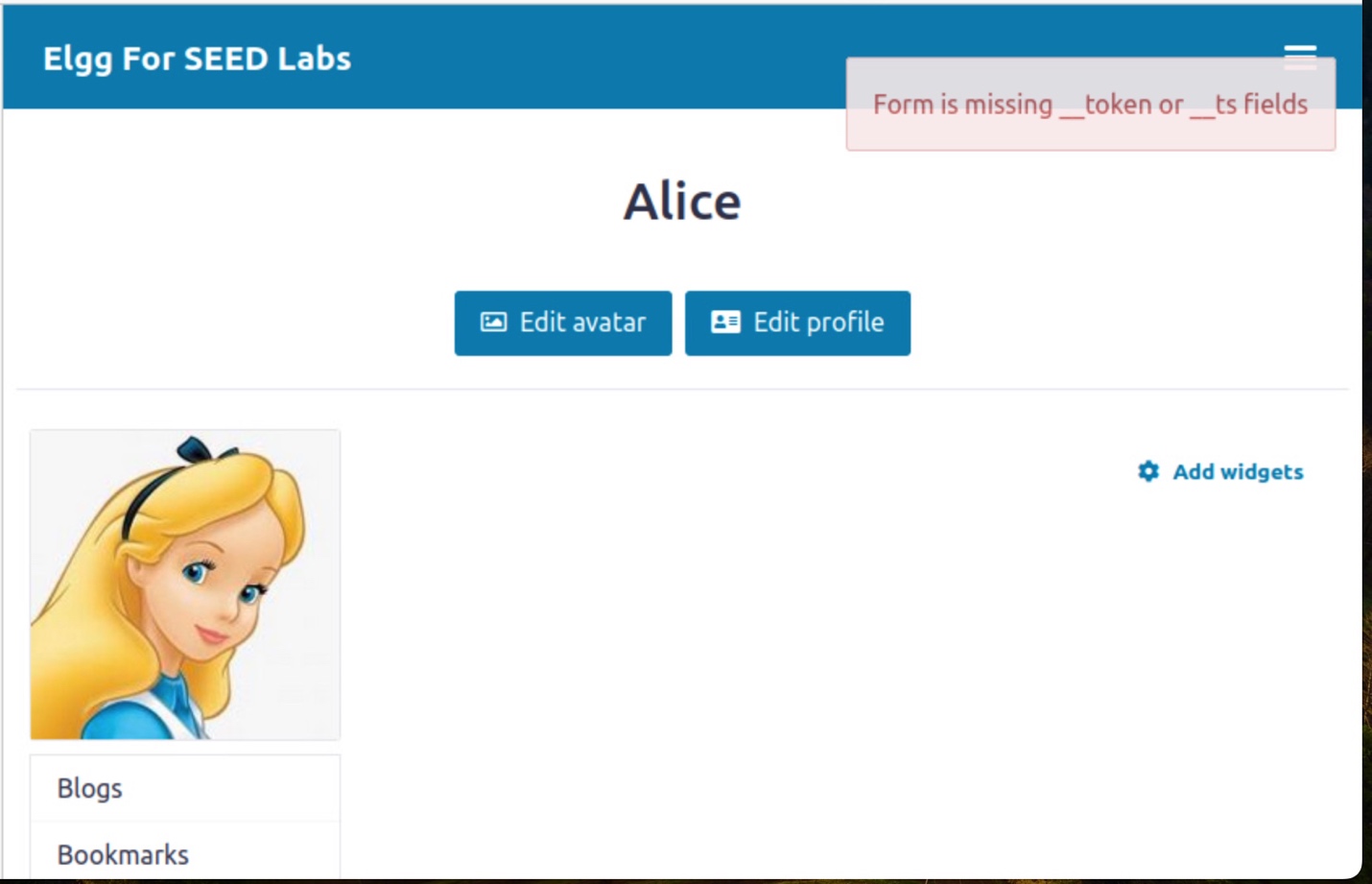
Description automatically generated**

Here we reused our same attack message from the prior POST request attack. Which Alice falls for and clicks on.

**A screenshot of a computer

Description automatically generated**

After clicking we would need her to click on the “Edit-Profile Attack” link which will perform our attack to modify her profile.

****

As we can see above our attack to edit Alices profile failed due to the CSRF protection being enabled. Due to turning on the CSRF countermeasure we were unable to complete the attack due to not being able to recreate the correct security token and timestamp request.

**FUNNY JOKE TIME & COOL PICS**

**What’s a CSRF attack’s favorite music genre?**

Anything with heavy POST-punk!

