Lab 5.3. Cross-Site Request Forgery (CSRF) Attack Lab

1. Lab Environment

This lab can only be conducted in our Ubuntu 16.04 VM, because of the configurations that we have performed to support this lab. We summarize these configurations in this section.

The Elgg **Web Application**. We use 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 pre-built Ubuntu VM image. We have also created several user accounts on the Elgg server and the credentials are given below.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

DNS Configuration. This lab involves two websites, the victim website and the attacker's website. Both websites are set up on our VM. Their URLs and folders are described in the following:

```
Attacker's website
    URL: http://www.csrflabattacker.com
    Folder: /var/www/CSRF/Attacker/

Victim website (Elgg)
    URL: http://www.csrflabelgg.com
    Folder: /var/www/CSRF/Elgg/
```

The above URLs are is only accessible from inside of the virtual machine, because we have modified the /etc/hosts file to map the domain name of each URL to the virtual machine's local IP address (127.0.0.1). You may map any domain name to a particular IP address using /etc/hosts. For example, you can map http://www.example.com to the local IP address by appending the following entry to /etc/hosts:

        127.0.0.1   www.example.com

If your web server and browser are running on two different machines, you need to modify /etc/hosts on the browser's machine accordingly to map these domain names to the web server's IP address, not to 127.0.0.1.

Apache Configuration. In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named 000-default.conf in the directory "/etc/ apache2/sites-available" contains the necessary directives for the configuration:

Inside the configuration file, each web site has a VirtualHost block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL http://www.example1.com and another website with URL http://www.example2.com:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application http://www.example1.com can be changed by modifying the sources in the /var/www/Example_1/ directory. After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

$ sudo service apache2 start

2. Lab Tasks

For the lab tasks, you will use two web sites that are locally setup in the virtual machine. The first web site is the vulnerable Elgg site accessible at www.csrflabelgg.com inside the virtual machine. The second web site is the attacker's malicious web site that is used for attacking Elgg. This web site is accessible via www.csrflabattacker.com inside the virtual machine.

2.1. 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 is given in the Guideline section (§ 4.1). Please use this tool to capture an HTTP GET request and an HTTP POST request in Elgg. In your report, please identify the parameters used in this these requests, if any.
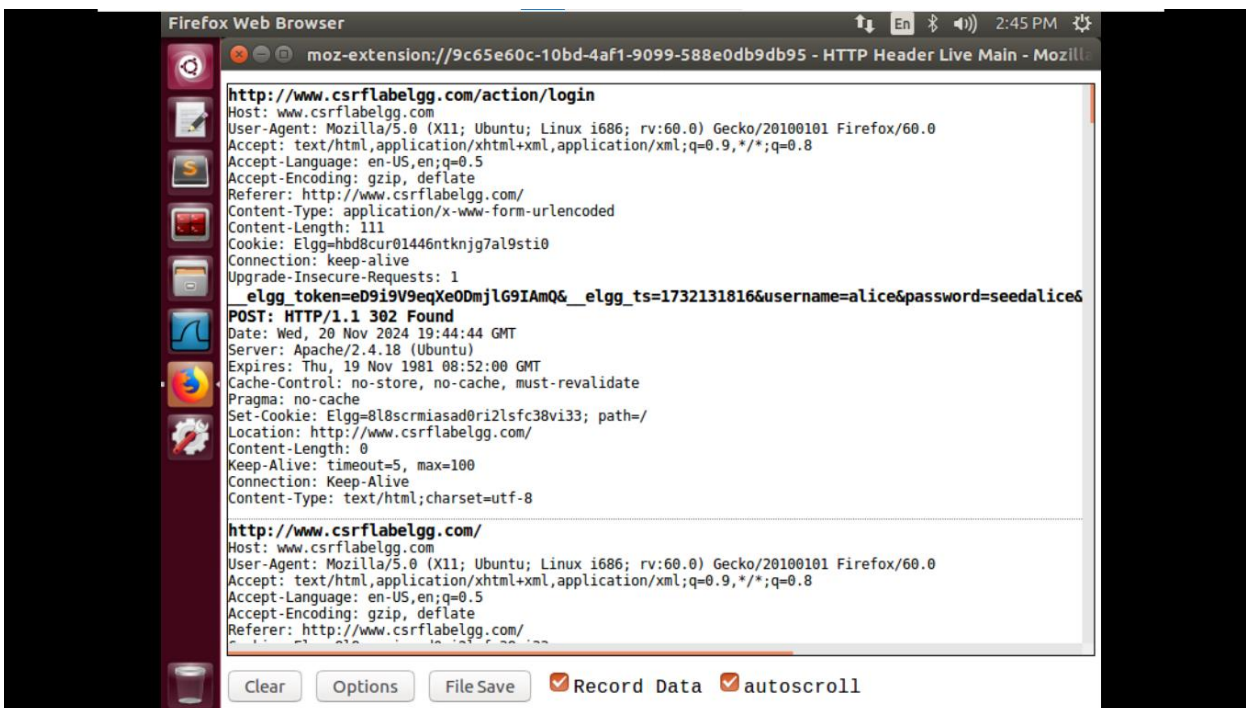
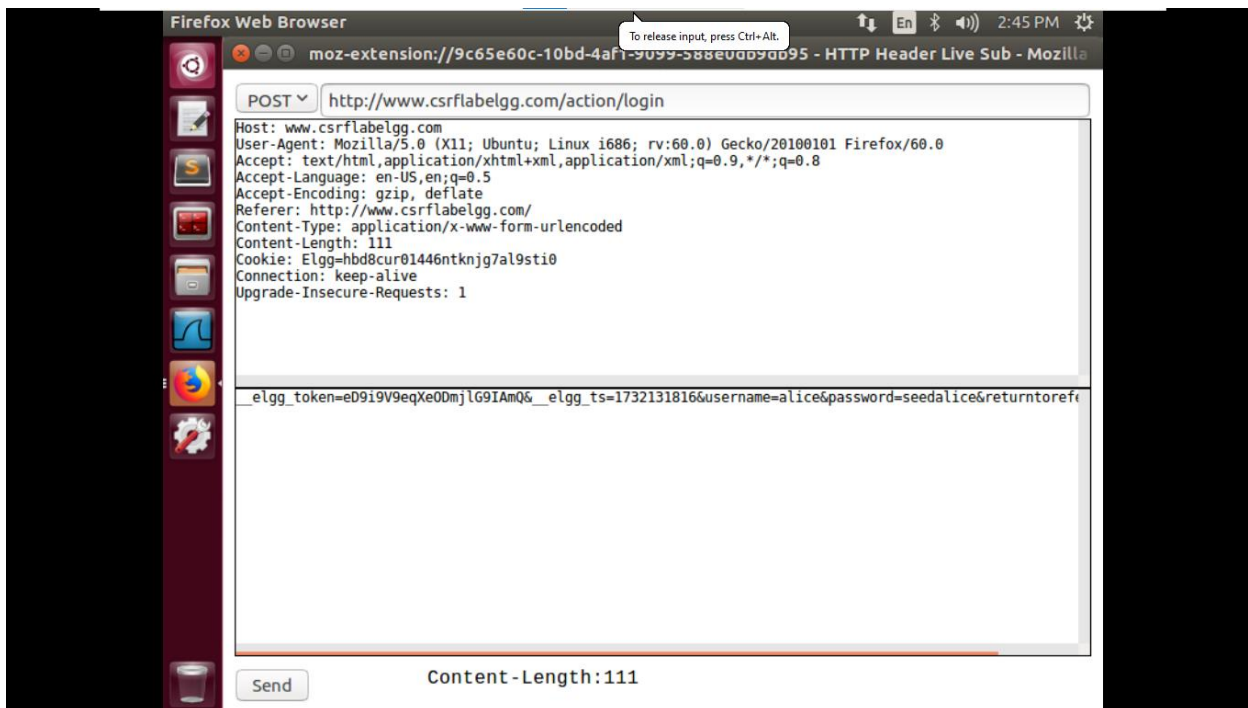**Các bước thực hiện:**

Đăng nhập với User Alice

Bật HTTP Header Live để xem thông tin khi các gói yêu cầu GET và POST được gửi đi. Sau đó nhập Username và Password rồi nhấn Log in
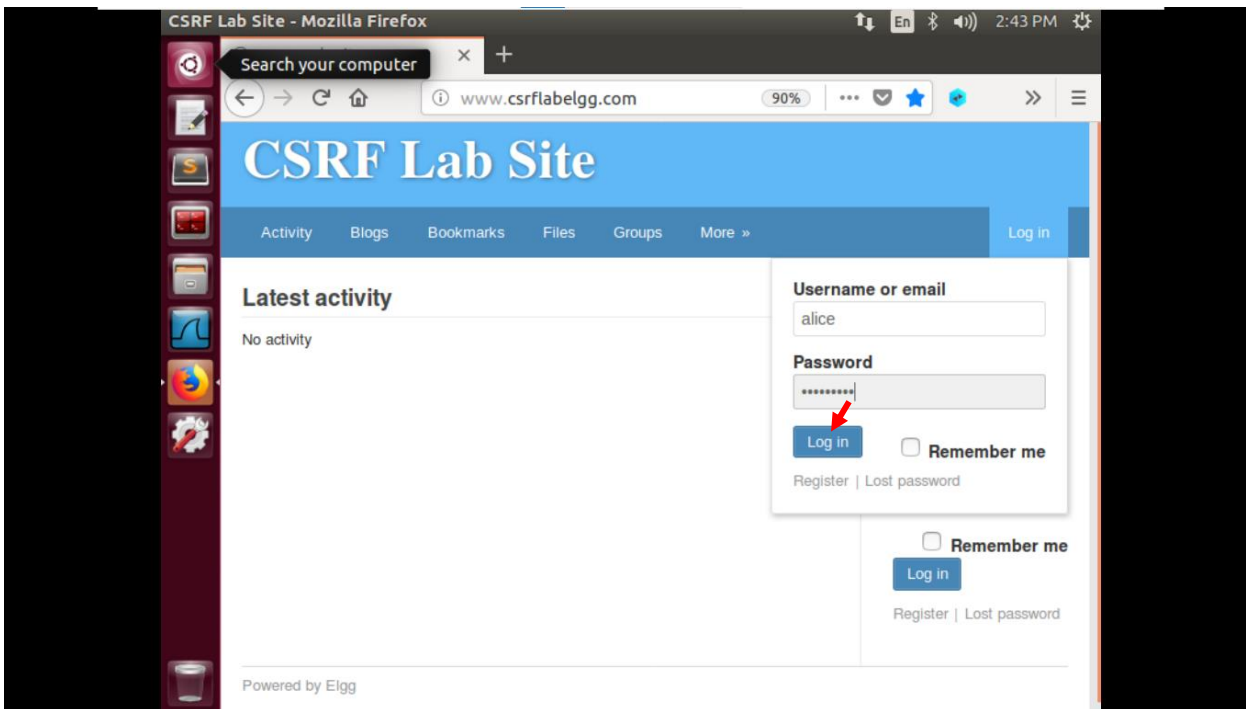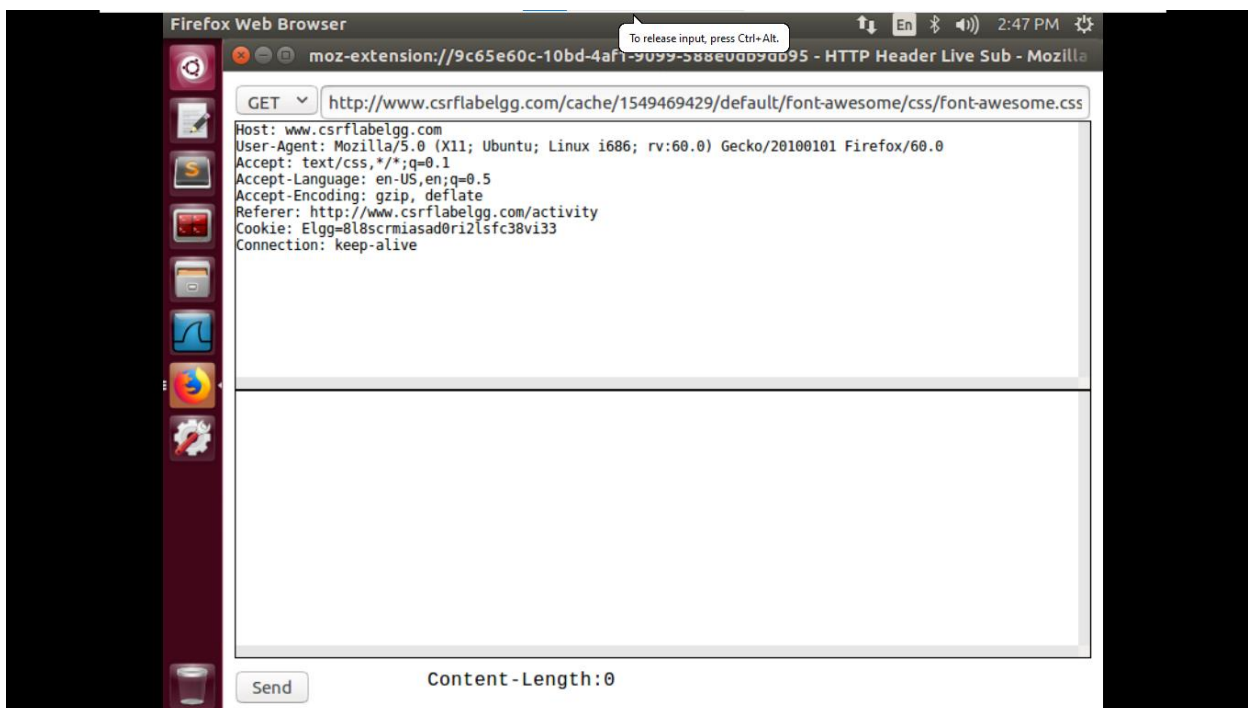

Kiểm tra kết quả:

Yêu cầu POST:

Yêu cầu GET:

2.2.Task 2: CSRF Attack using GET Request

In this task, we need two people in the Elgg social network: Alice and Boby. Boby wants to become a friend to Alice, but Alice refuses to add him to her Elgg friend list. Boby 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 Boby's web site: www.csrflabattacker.com. Pretend that you are Boby, describe how you can construct the content of the web page, so as soon as Alice visits the web page, Boby 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 wired-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.
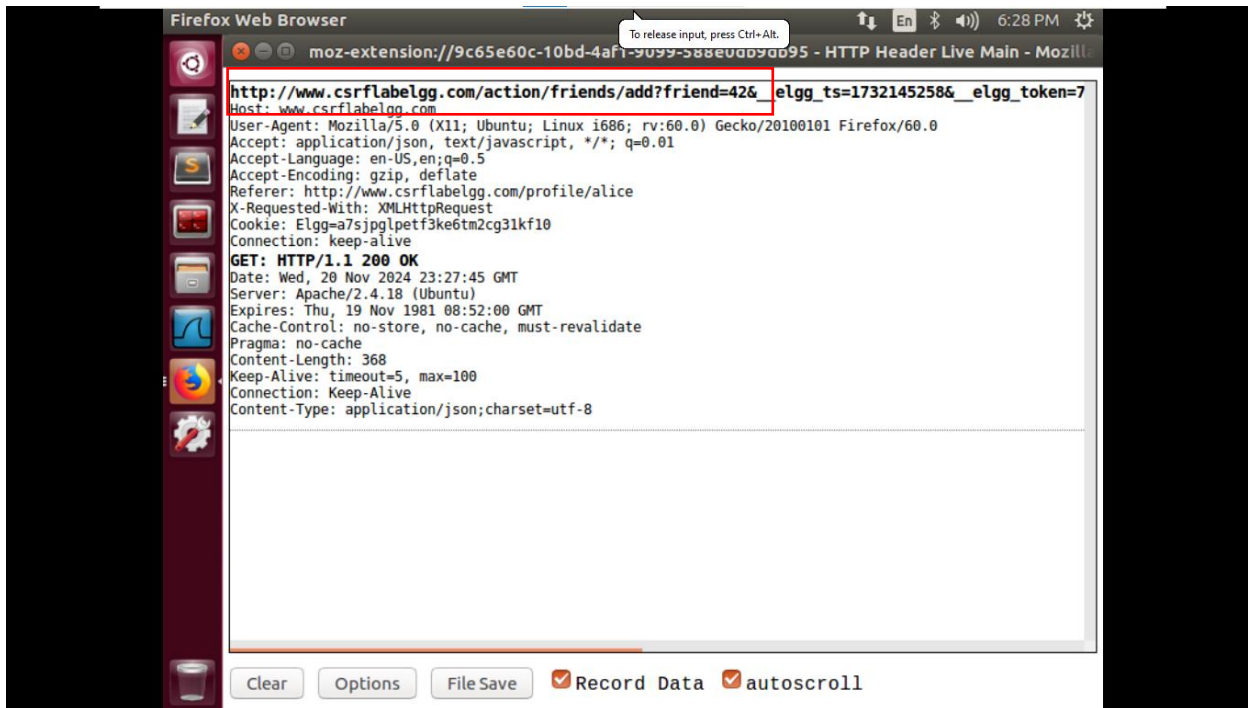
Các bước thực hiện:

Đăng nhập với User Boby

Boby add friend với Alice. Sau đó mở HTTP Header Live để xem thông tin các yêu cầu được gửi đi
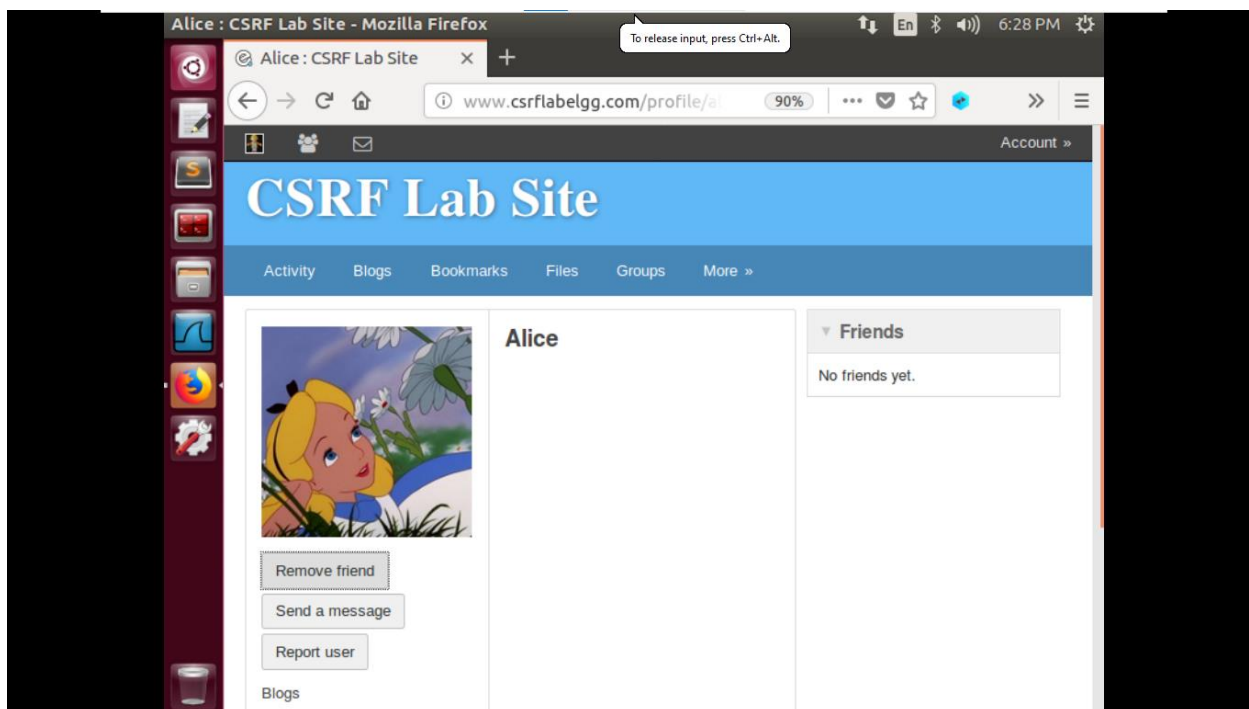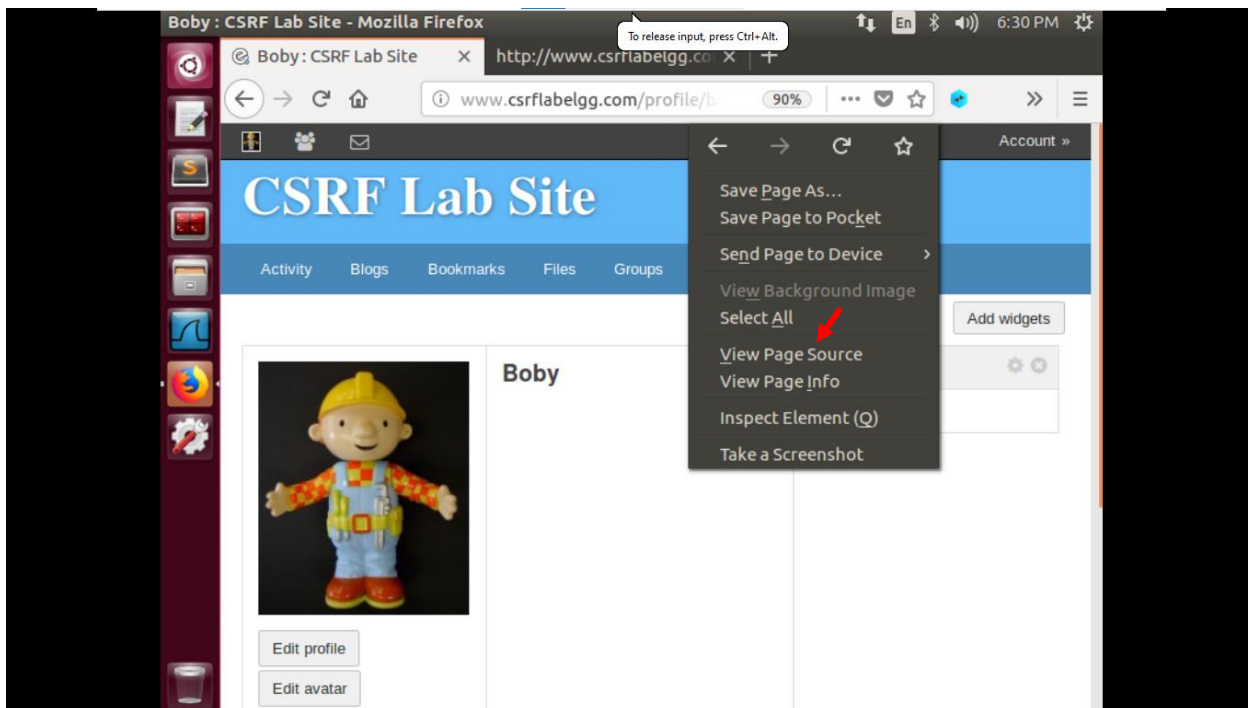
HTTP Header Live

Dòng đầu tiên là url của yêu cầu add friend Alice. Ta sẽ chỉnh sửa url của yêu cầu add friend Alice thành url của yêu cầu add friend Boby bằng cách thay 42 thành guid của Boby
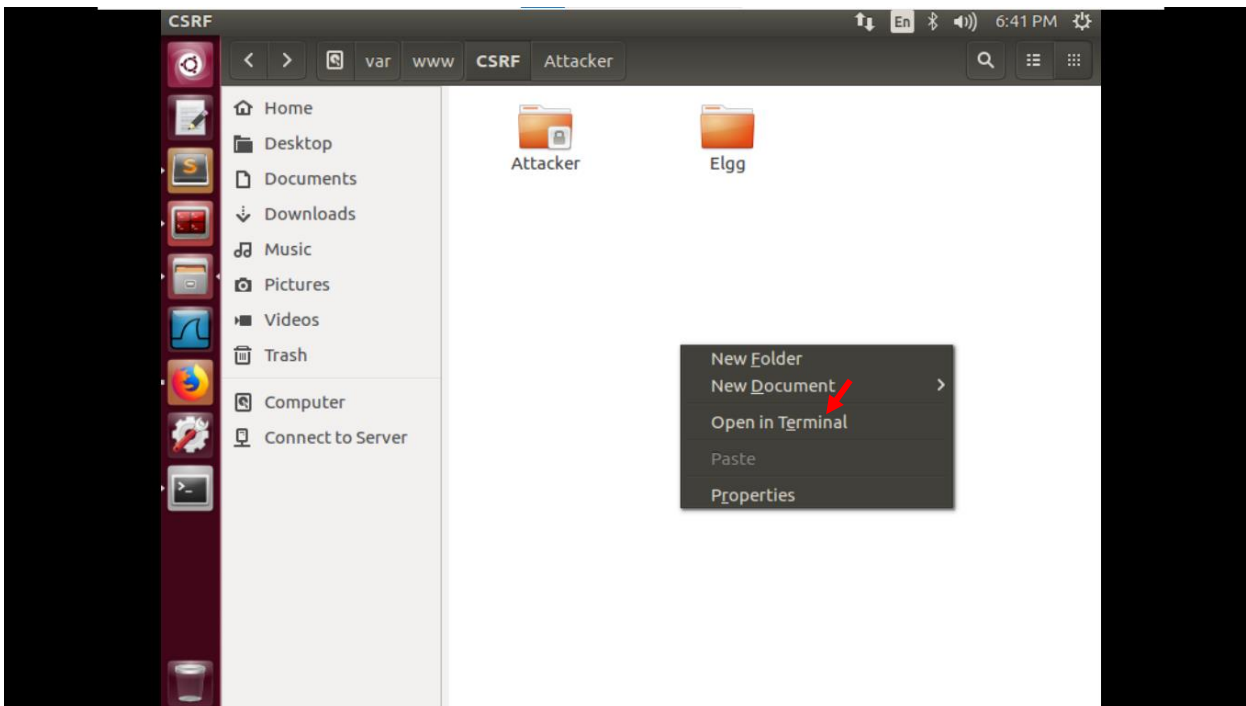
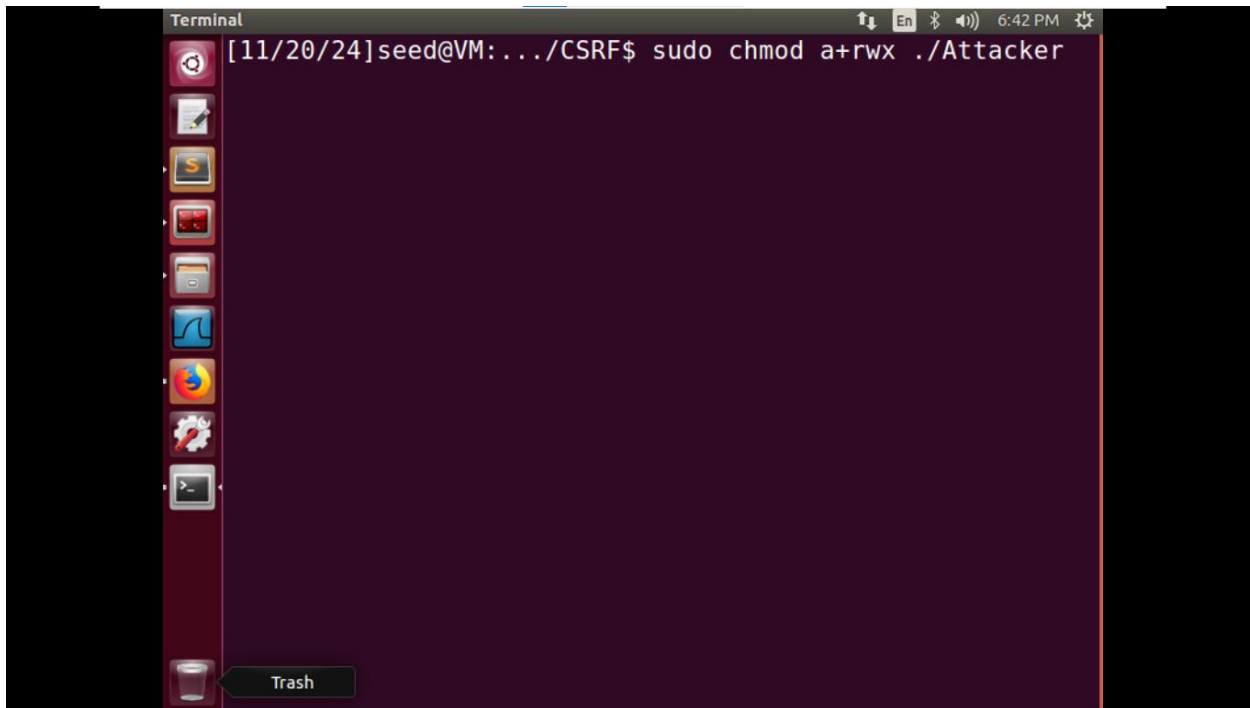Ta thực hiện cách dưới để tìm guid của Boby

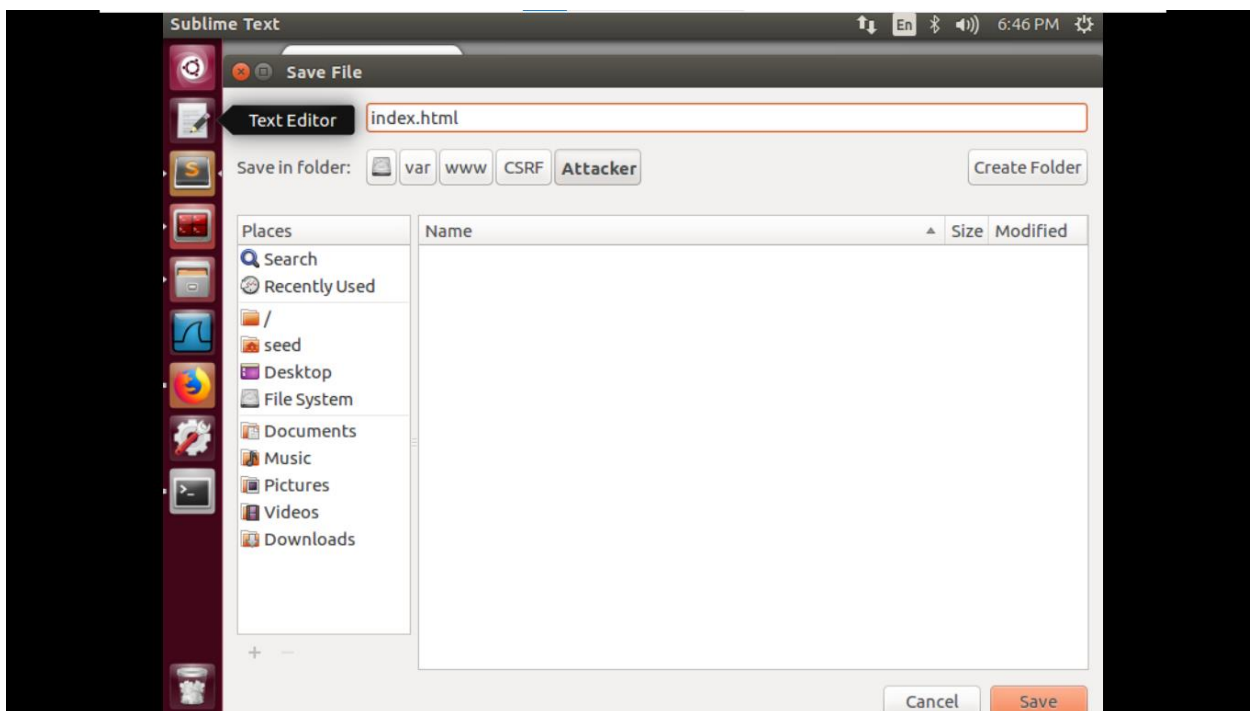guid của Boby là 43

Vậy url của yêu cầu add friend Boby là:

http://www.csrflabelgg.com/action/friends/add?friend=43

Vào thư mục /var/www/CSRF/Attacker
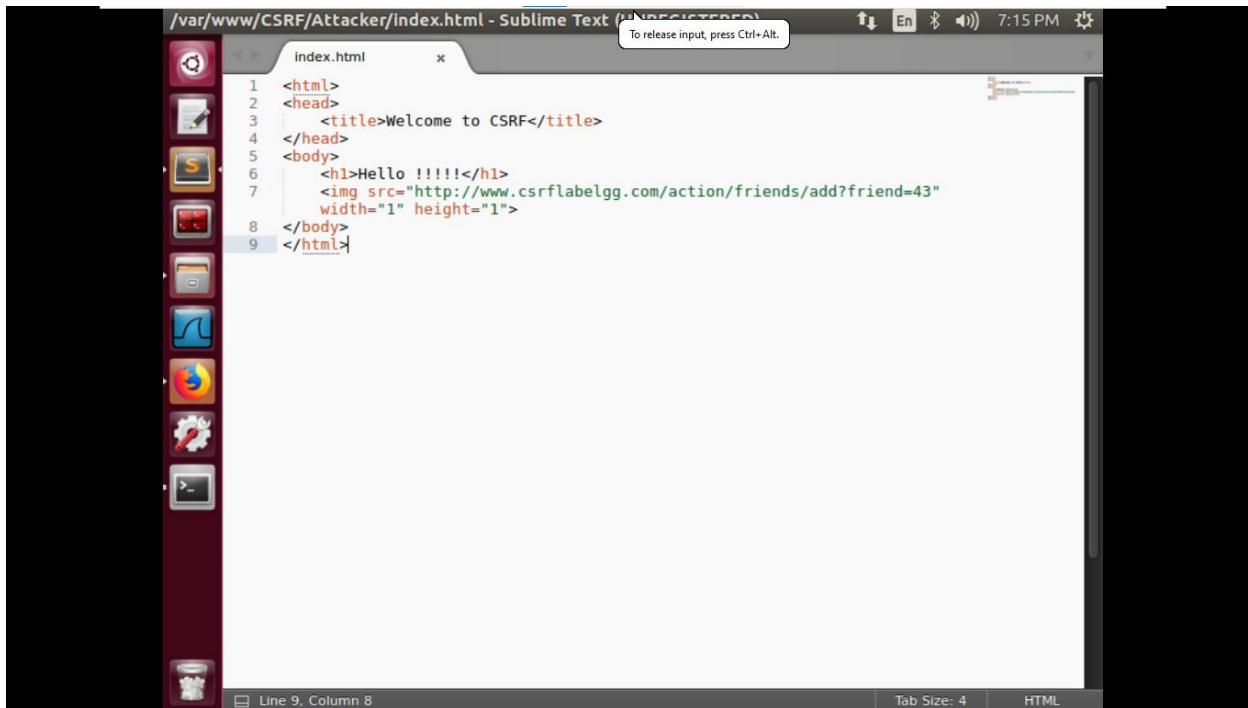
Mở khoá thư mục Attacker

Tạo index.html trong /var/www/CSRF/Attacker

Viết chương trình trong file index.html vừa tạo cho URL : http://www.csrflabattacker.com

Tạo blog và cài URL: http://www.csrflabattacker.com vào body của blog để người dùng khác nhấn vào.
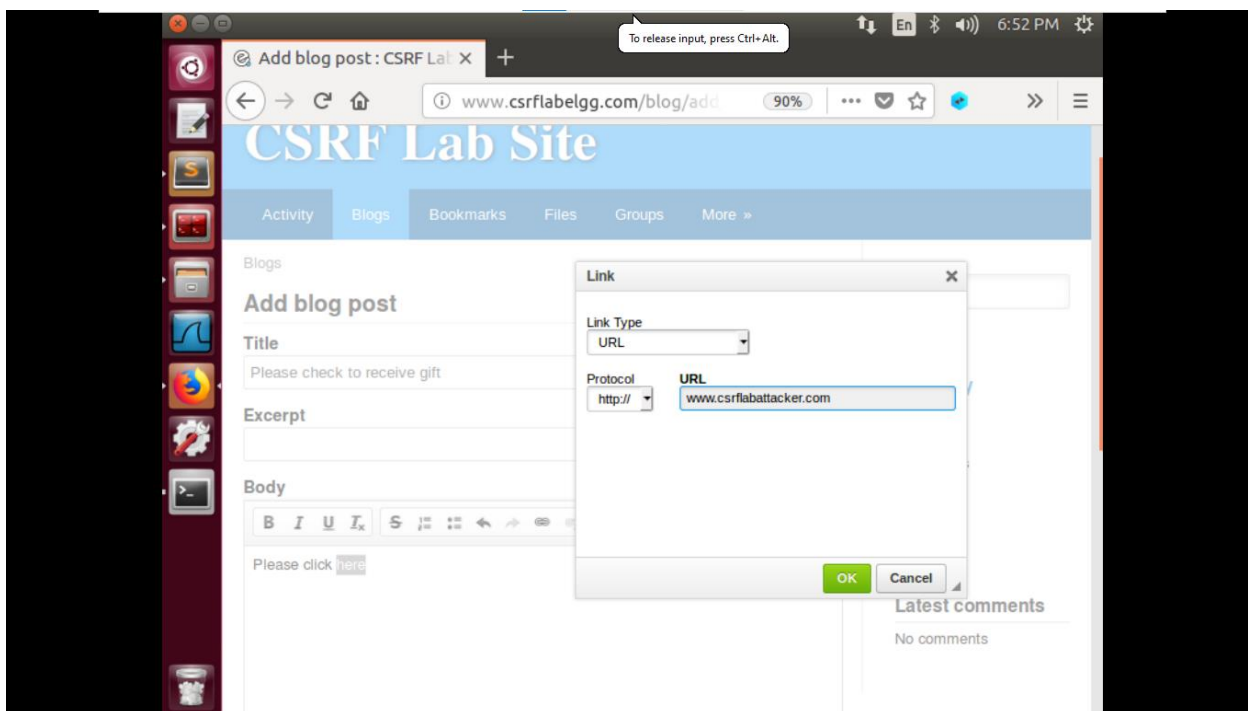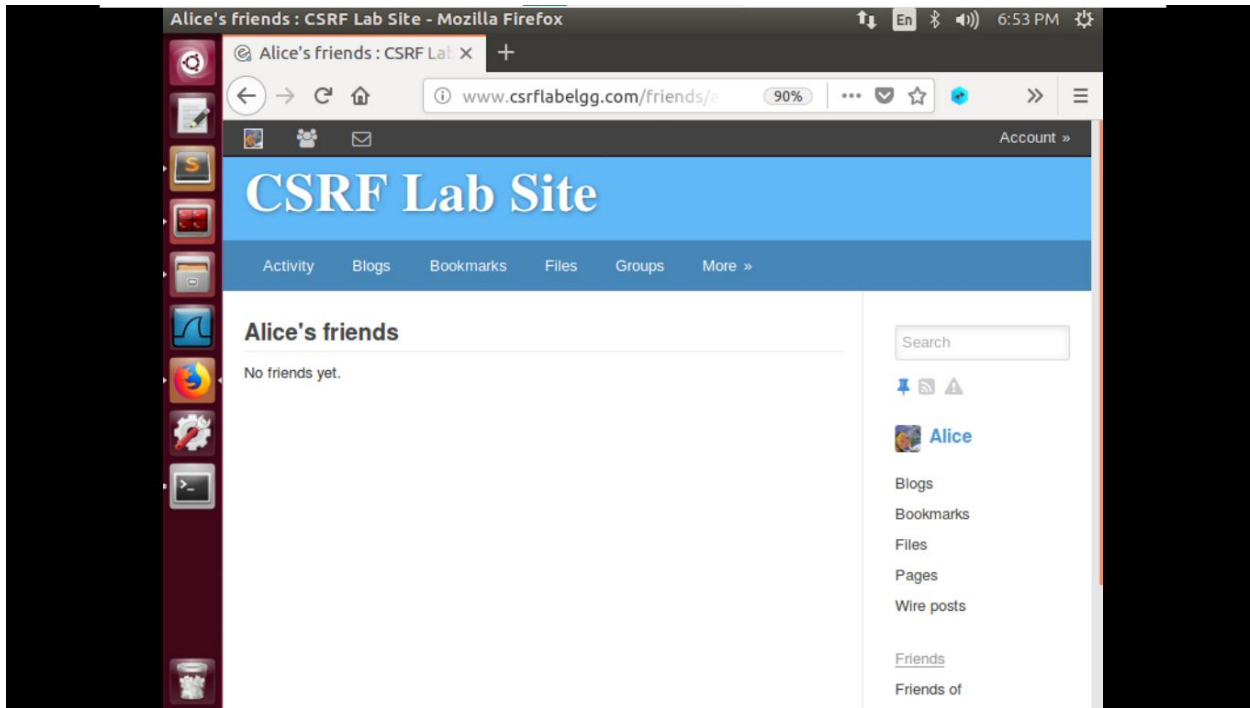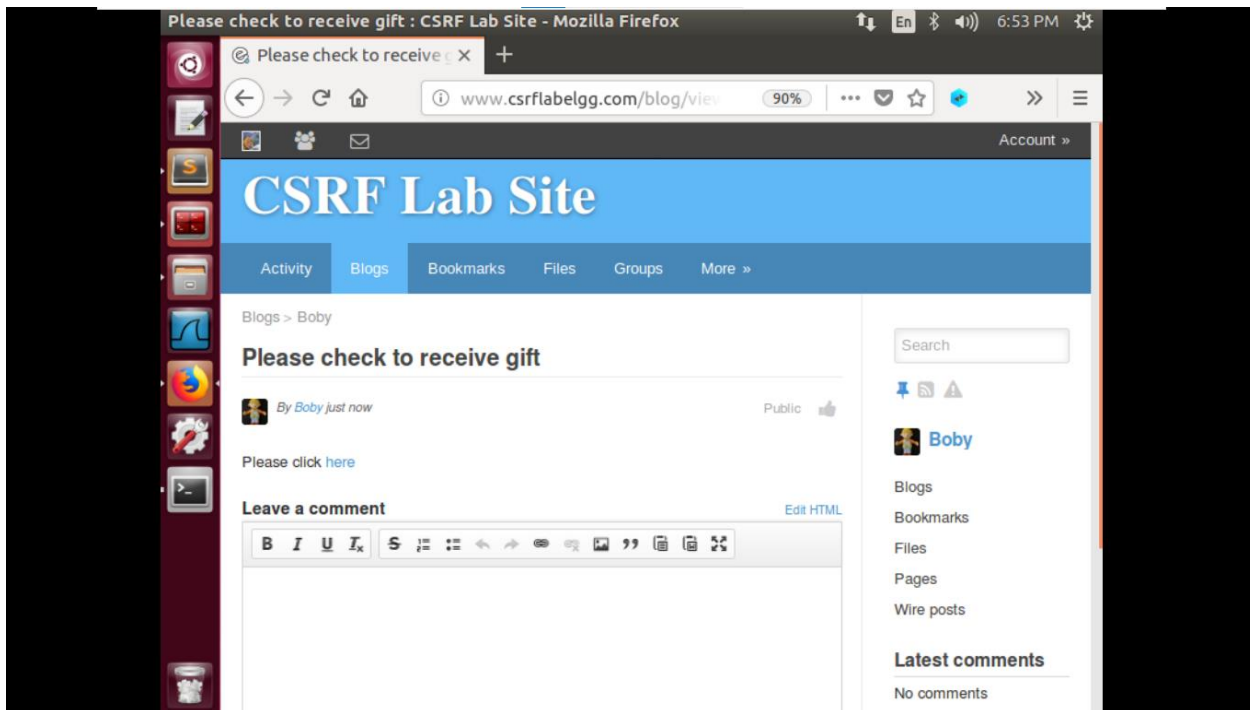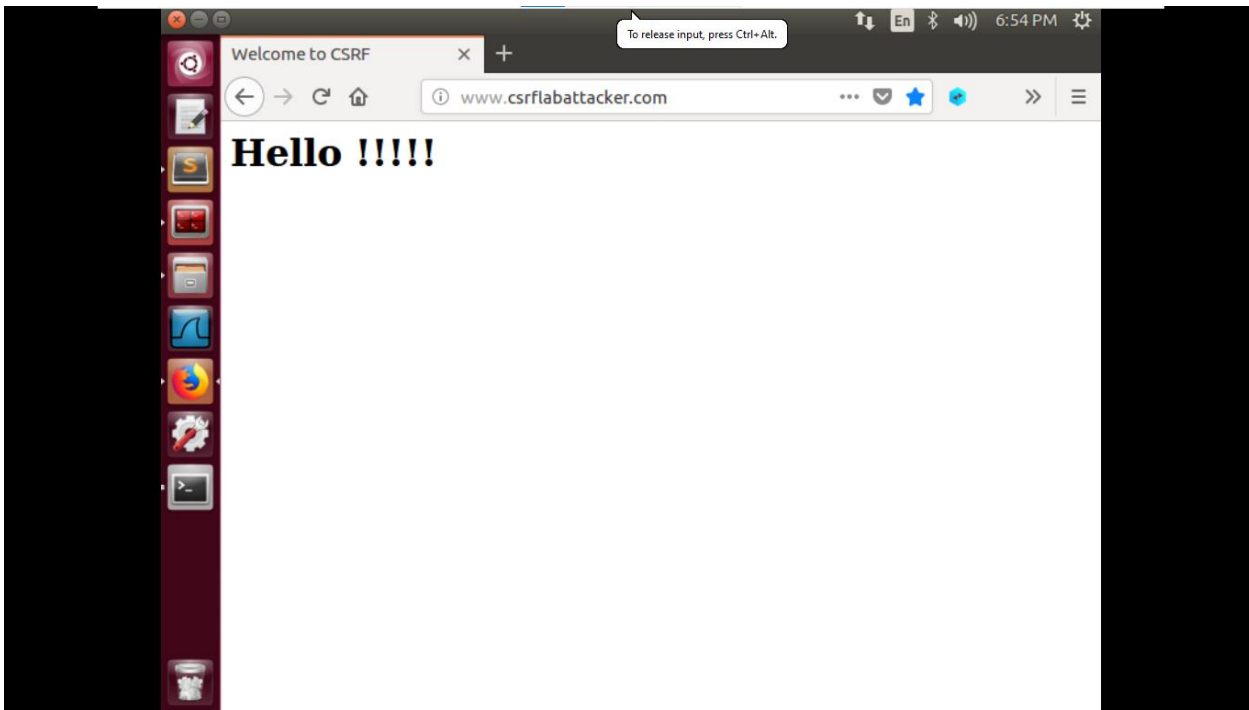
Kiểm tra:

Đăng nhập với tài khoản Alice
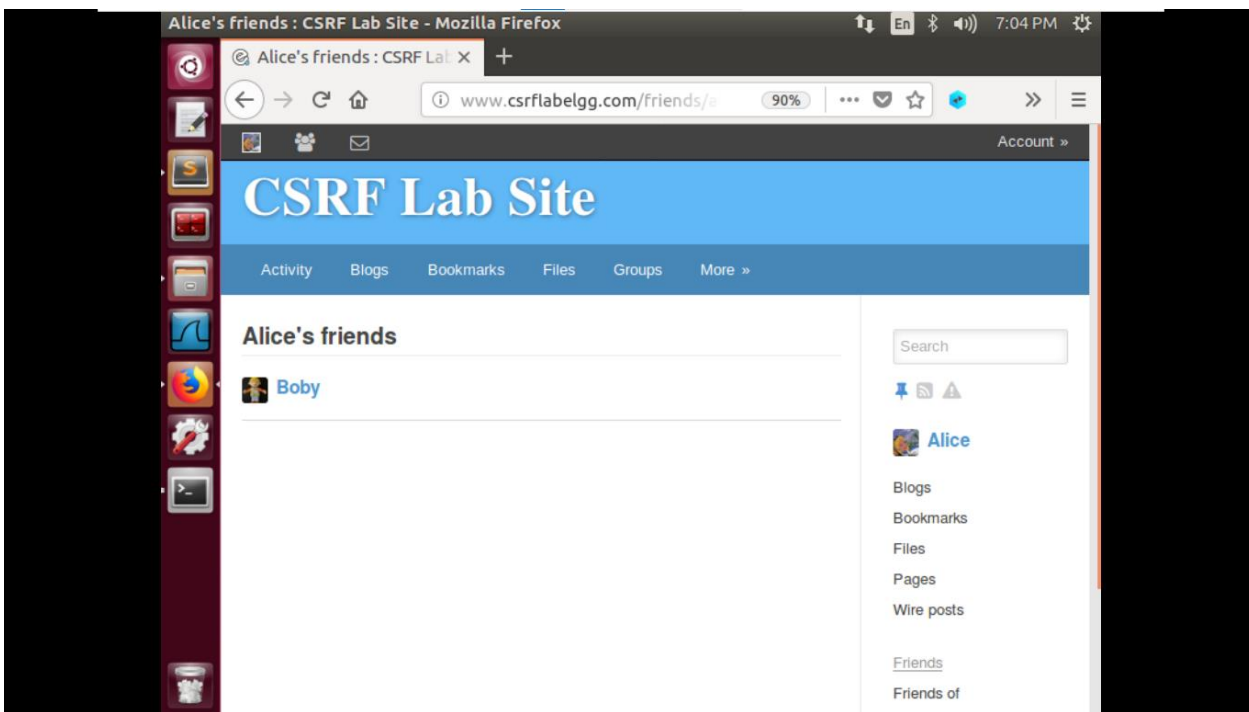
Người dùng Alice chưa add friend với ai

Khi người dùng Alice xem blog và nhấn vào here . Alice sẽ được dẫn đến url mà Boby đã cài sẵn.

Sau khi người dùng Alice nhấn vào here trong blog. Trong friend của Alice đã tự động add friend với Boby.

2.3.Task 3: CSRF Attack using POST Request

After adding himself to Alice's friend list, Boby wants to do something more. He wants Alice to say "Boby is my Hero" in her profile, so everybody knows about that. Alice does not like Boby, let alone putting that statement in her profile. Boby 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., Boby's) malicious web site www. csrflabattacker.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 Task 1 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 P symbols):

```
http://www.csrflabelgg.com/action/profile/edit

POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.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.csrflabelgg.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 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>
```
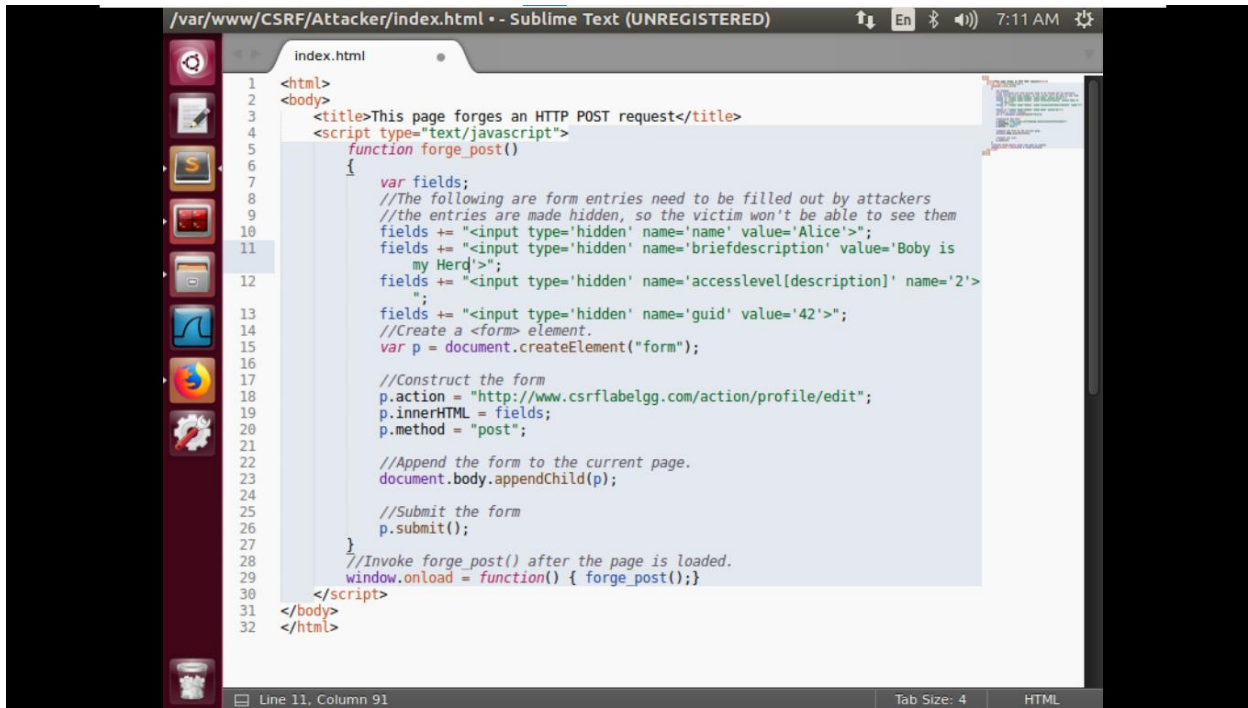
In Line ①, the value 2 sets the access level of a field to public. This is needed, otherwise, the access level will be set by default to private, so others cannot see this field. It should be noted that when copy-andpasting the above code from a PDF file, the single quote character in the program may become something else (but still looks like a single quote). That will cause syntax errors. Replace all the single quote symbols with the one typed from your keyboard will fix those errors.

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.
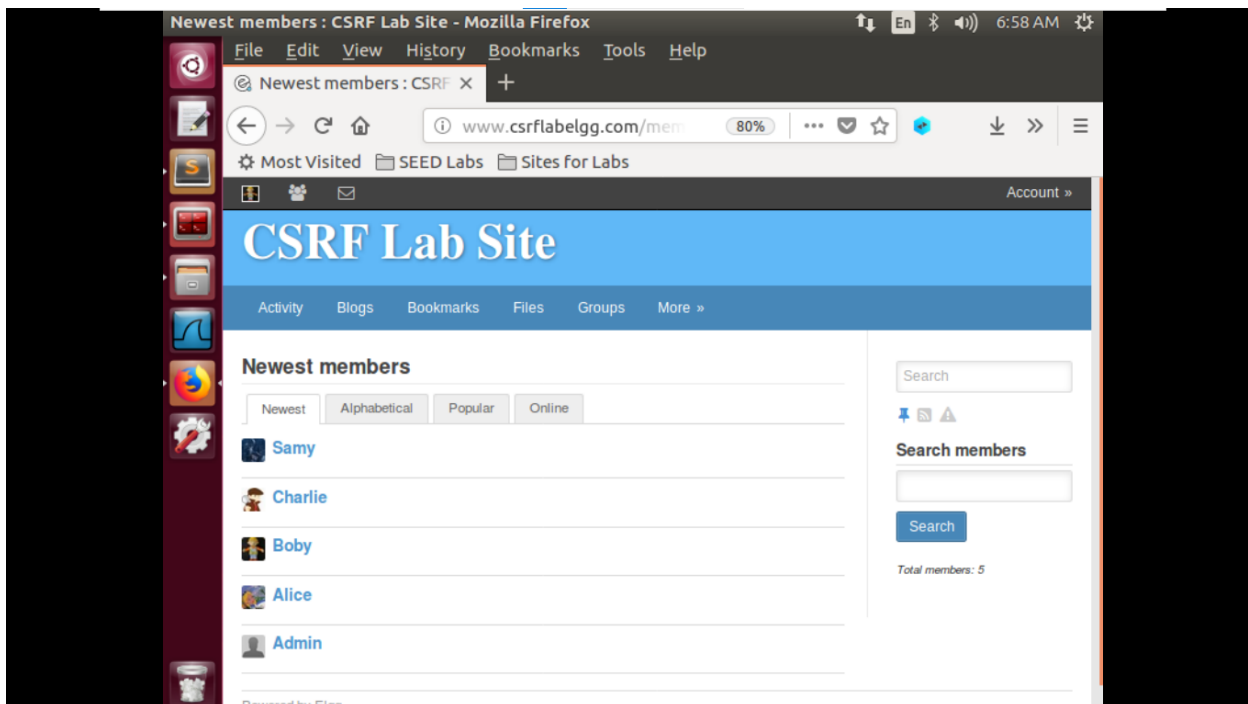
Các bước thực hiện:

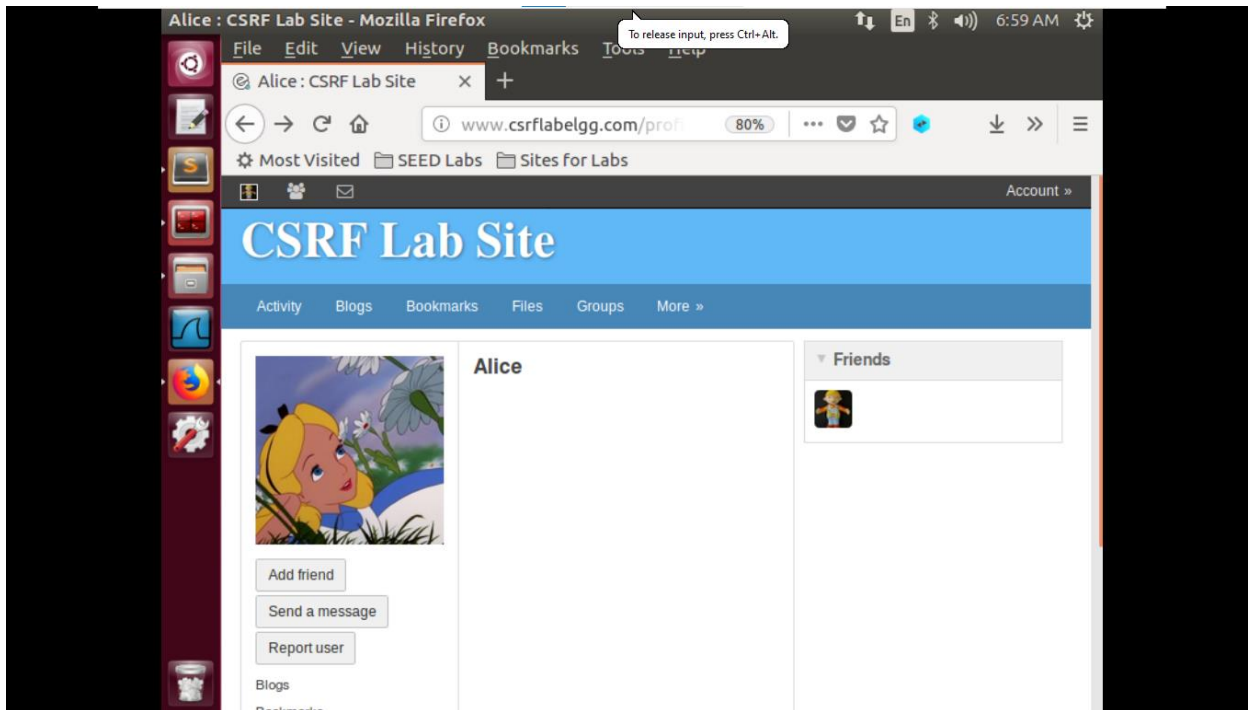Đầu tiên sửa chương trình trong index.html đã được tạo ở task 1:
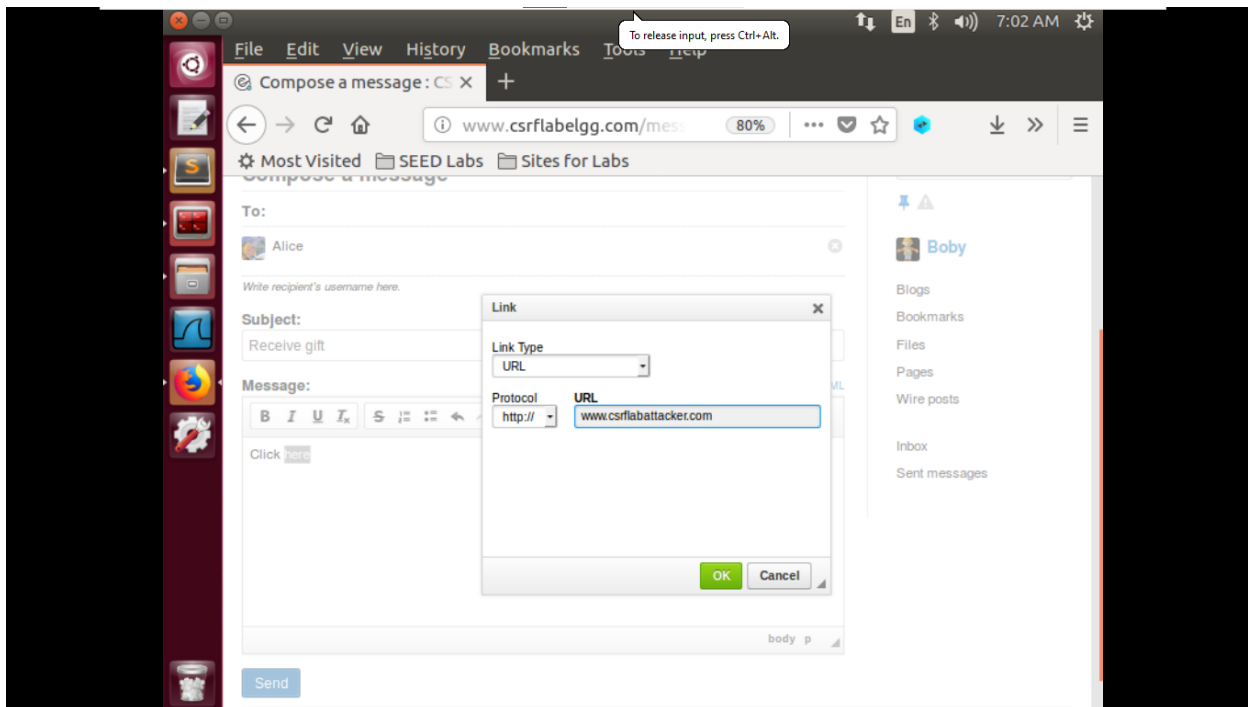

Đăng nhập với User Boby để thực hiện task 3: Xem profile của Alice

Nhấn vào Send a massage để gửi massage cho Alice

Viết massage như bên dưới và gắn link url: http://www.csrflabelgg.com vào message để người dùng Alice nhấn vào. Sau đó Send
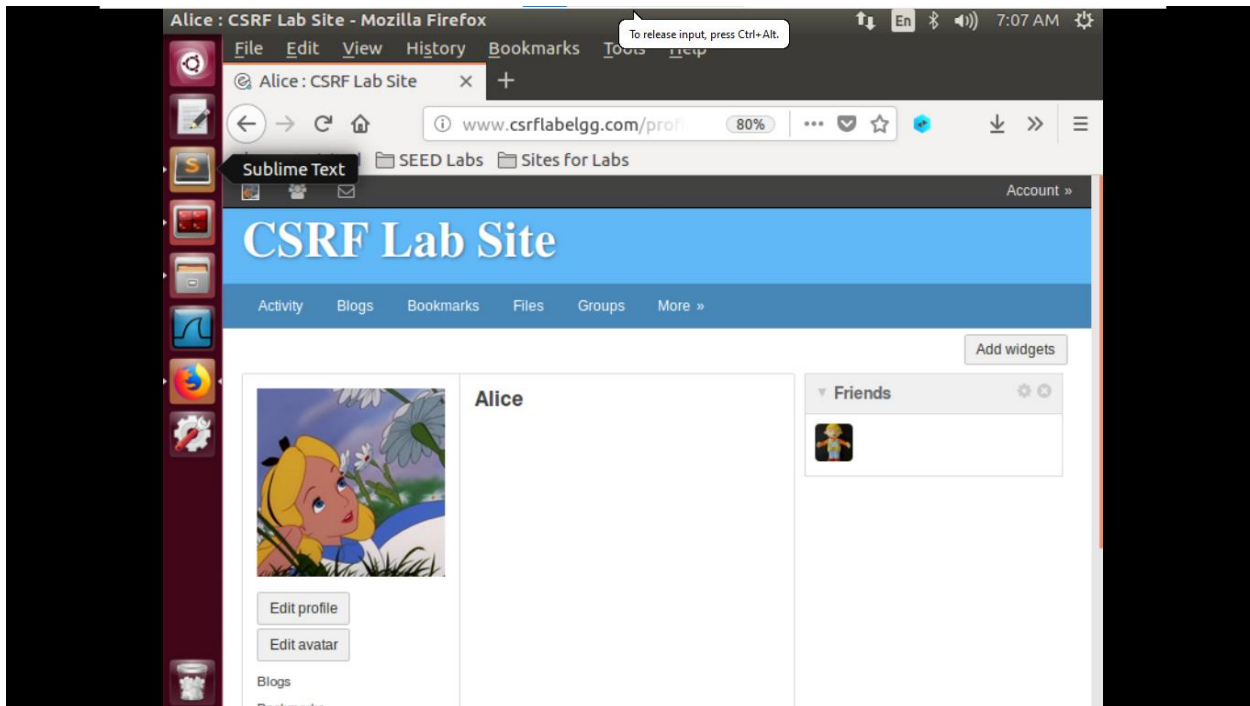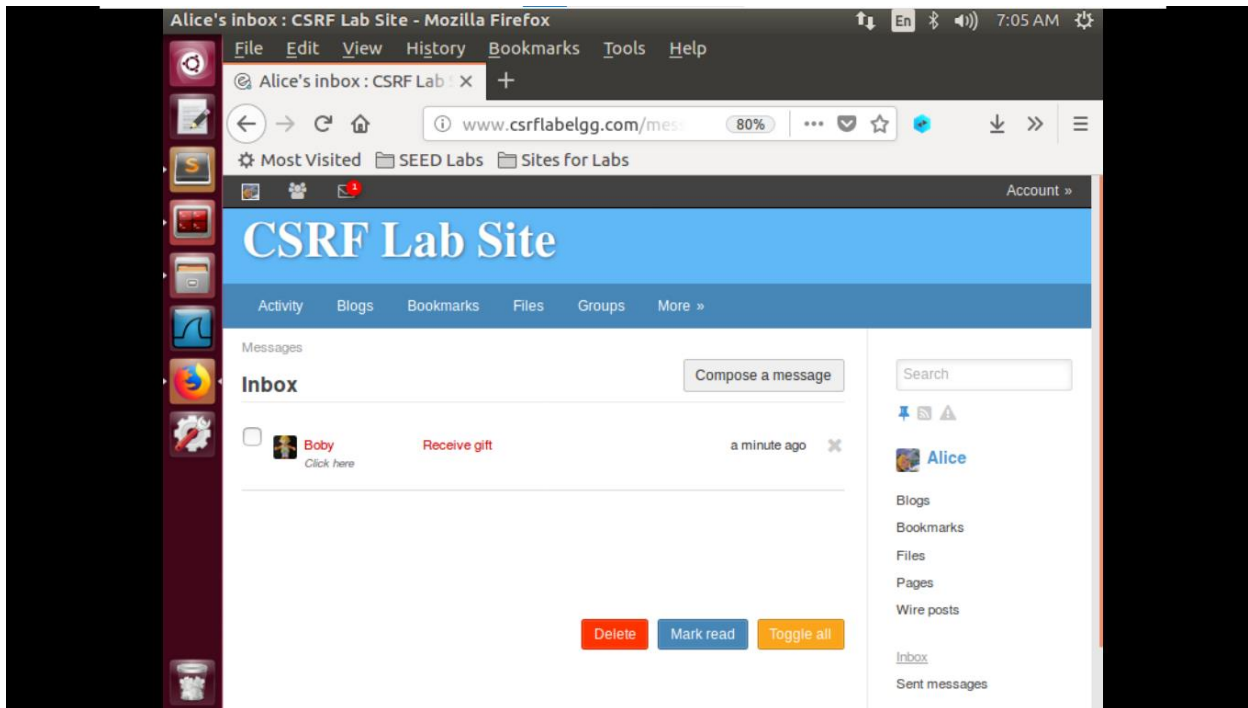
Kiểm tra kết quả:

Đăng nhập với User Alice
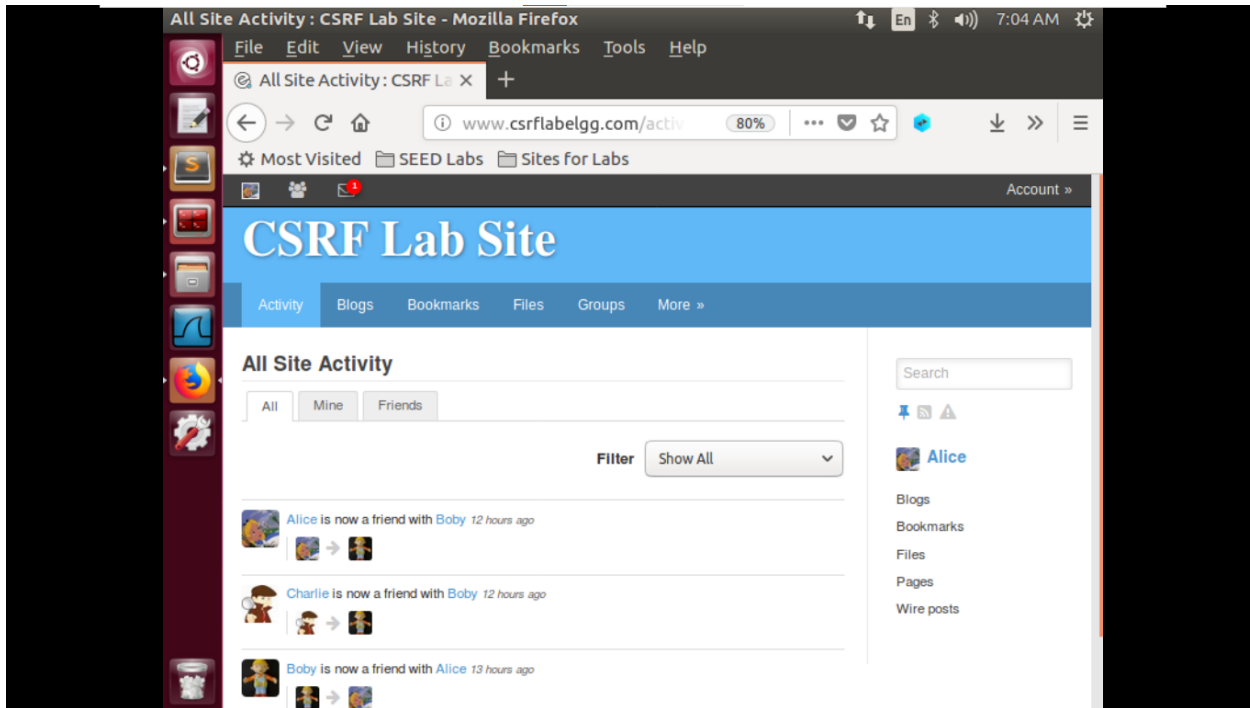
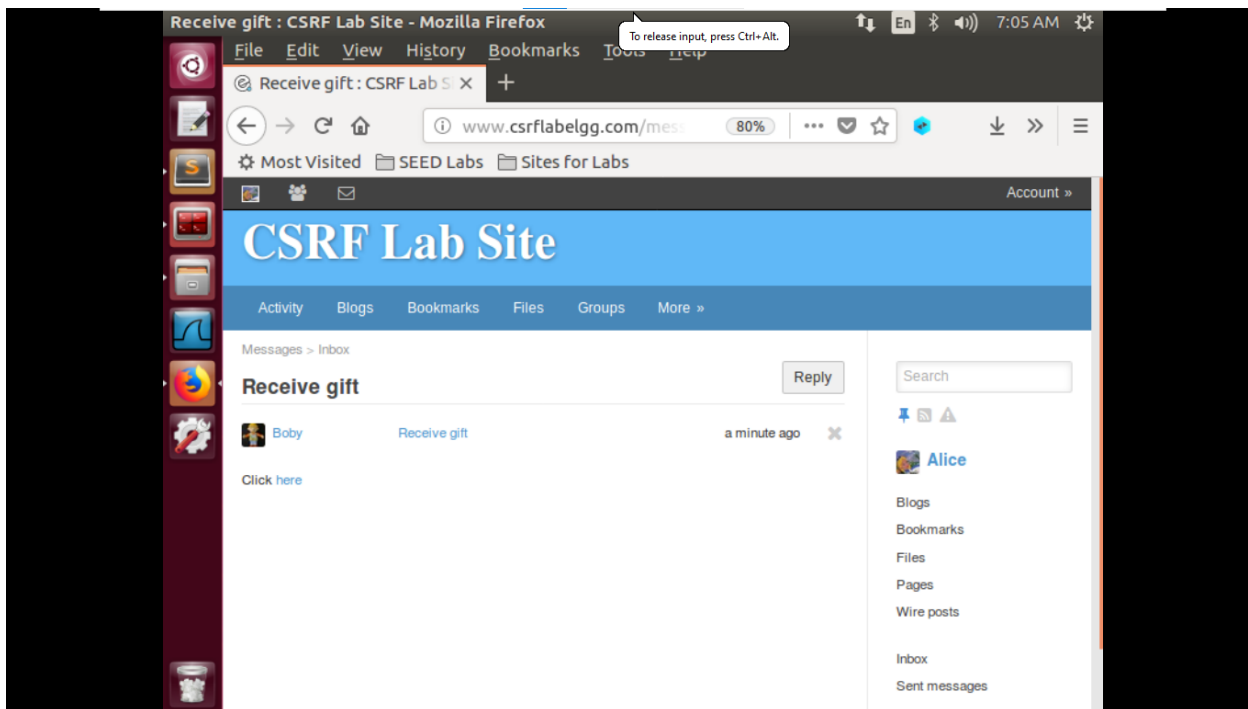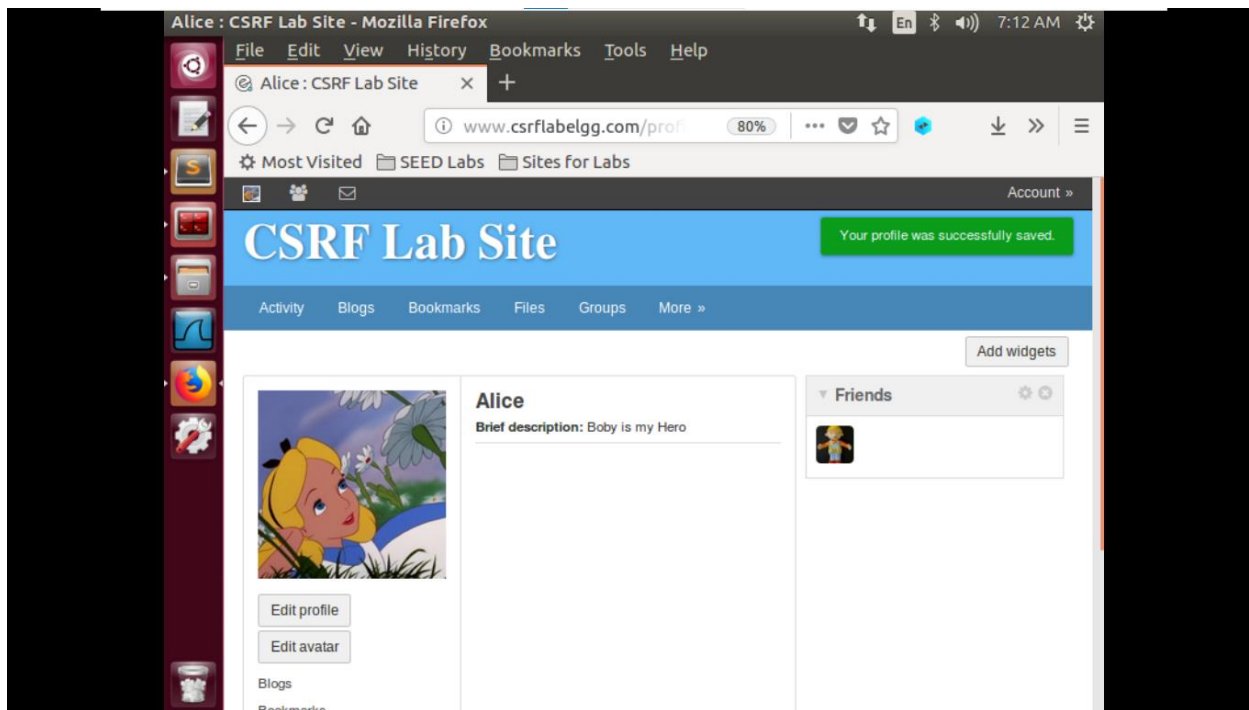Profile của Alice chưa có thông tin gì

Người dùng Alice xem tin nhắn

Khi người dùng Alice click vào here thì Alice sẽ được dẫn đến url mà Boby đã cài sẵn

Profile của Alice đã được chỉnh sửa thông tin

2.4.Task 4: Implementing a countermeasure for Elgg

Elgg does have a built-in countermeasures to defend against the CSRF attack. We have commented out the countermeasures to make the attack work. CSRF is not difficult to defend against, and there are several common approaches:

> • *Secret-token approach*: Web applications can embed a secret token in their pages, and all requests coming from these pages will carry this token. Because cross-site requests cannot obtain this token, their forged requests will be easily identified by the server.

> • *Referrer header approach*: Web applications can also verify the origin page of the request using the *referrer* header. However, due to privacy concerns, this header information may have already been filtered out at the client side.

The web application Elgg uses secret-token approach. It embeds two parameters __elgg_ts and __elgg_token in the request as a countermeasure to CSRF attack. The two parameters are added to the HTTP message body for the POST requests and to the URL string for the HTTP GET requests.

Elgg **secret-token and timestamp in the body of the request**. Elgg adds security token and timestamp to all the user actions to be performed. The following HTML code is present in all the forms where user action is required. This code adds two new hidden parameters __elgg_ts and __elgg_token to the POST request:

```
<input type = "hidden" name = "__elgg_ts" value = "" />
<input type = "hidden" name = "__elgg_token" value = "" />
```
The __elgg_ts and __elgg_token are generated by the views/default/input/securitytoken.php module and added to the web page. The code snippet below shows how it is dynamically added to the web page.

```
$ts = time();
$token = generate_action_token($ts);

echo elgg_view('input/hidden', array('name' => '__elgg_token', 'value' =>
   $token));
echo elgg_view('input/hidden', array('name' => '__elgg_ts', 'value' => $ts));
```
Elgg
also adds the security tokens and timestamp to the JavaScript which can be accessed by

```
elgg.security.token.__elgg_ts;
elgg.security.token.__elgg_token;
```

Elgg security token is a hash value (md5 message digest) of the site secret value (retrieved from database), timestamp, user sessionID and random generated session string. There by defending against the CSRF attack. The code below shows the secret token generation in Elgg.

```
function generate_action_token($timestamp)
{
  $site_secret = get_site_secret();
  $session_id = session_id();
  // Session token
  $st = $_SESSION['__elgg_session'];

  if (($site_secret) && ($session_id))
  {
    return md5($site_secret . $timestamp . $session_id . $st);
  }

  return FALSE;
}
```

The PHP function session_id() is used to get or set the session id for the current session. The below code snippet shows random generated string for a given session __elgg_session apart from public user Session ID.

```
......
// Generate a simple token (private from potentially public session id)
if (!isset($_SESSION['__elgg_session'])) {
$_SESSION['__elgg_session'] =
  ElggCrypto::getRandomString(32,ElggCrypto::CHARS_HEX);
........
```

Elgg **secret-token validation**. The elgg web application validates the generated token and timestamp to defend against the CSRF attack. Every user action calls validate_action_token function and this function validates the tokens. If tokens are not present or invalid, the action will be denied and the user will be redirected.

The below code snippet shows the function validate_action_token.

```
function validate_action_token($visibleerrors = TRUE, $token = NULL, $ts =
    NULL)
{
  if (!$token) {  $token = get_input('__elgg_token'); }
  if (!$ts) {$ts = get_input('__elgg_ts');    }
  $session_id = session_id();
  if (($token) && ($ts) && ($session_id)) {
    // generate token, check with input and forward if invalid
    $required_token = generate_action_token($ts);

    // Validate token
    if ($token == $required_token) {

      if (_elgg_validate_token_timestamp($ts)) {
        // We have already got this far, so unless anything
        // else says something to the contrary we assume we're ok
        $returnval = true;
        ......
      }
      else {
        ......
        register_error(elgg_echo('actiongatekeeper:tokeninvalid'));
        ......
      }
      ......
}
```

Turn on countermeasure. To turn on the countermeasure, please go to the directory
/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg and find the function gatekeeper in the
ActionsService.php file. In function gatekeeper() please comment out the "return true;" statement as specified in
the code comments.

```
public function gatekeeper($action) {
   //SEED:Modified to enable CSRF.
   //Comment the below return true statement to enable countermeasure
   return true;
   ......
}
```

Task: After turning on the countermeasure above, try the CSRF attack again, and describe your observation.
Please point out the secret tokens in the HTTP request captured using Firefox's HTTP inspection tool. Please
explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding
out the secret tokens from the web page?