

Ethan Booker

Cross-Site Scripting (XSS) Attack Lab

Spring 2020

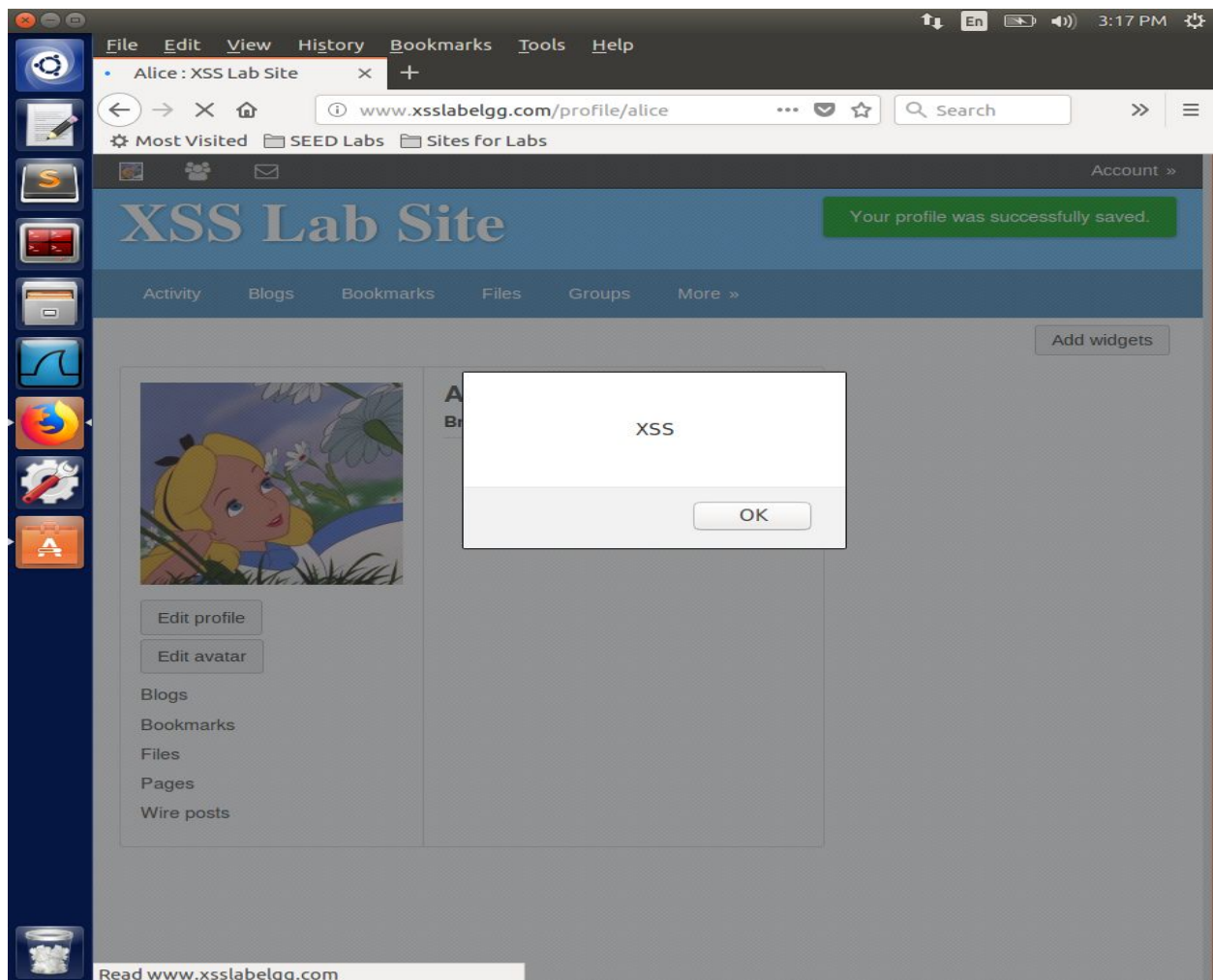
Description of overall goals of lab:

Apply how XSS vulnerabilities can be exploited and learn the implications that can happen. Also understand that there are some countermeasures to minimize the XSS threat.

3.2, Task 1

Description: Posting a malicious message to display an alert window.

Evidence:

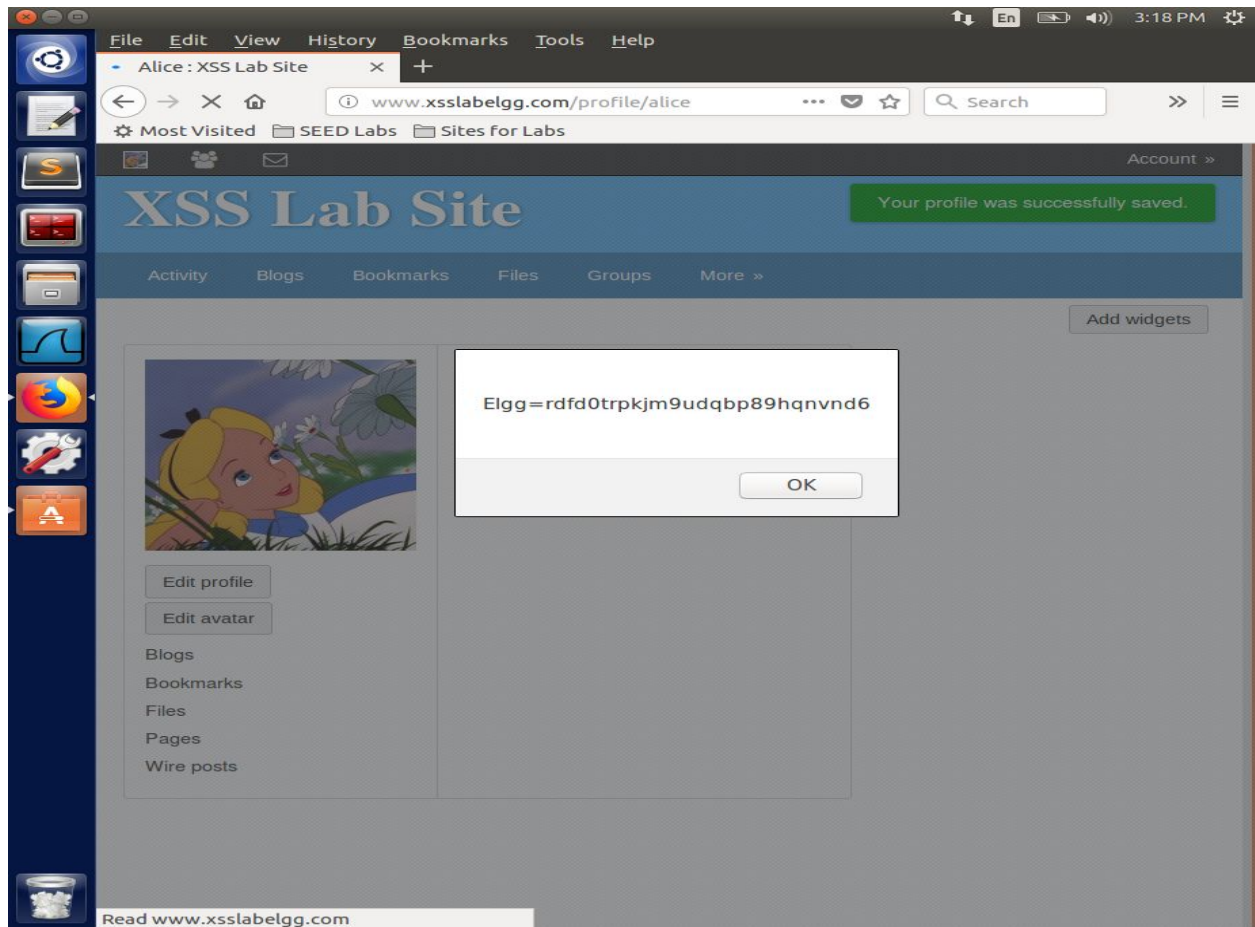


Analysis: This is a simple `alert()` javascript function which displays an alert window with the message inside and that was possible due to the website enabling an HTML editor inside the edit profile page.

3.3, Task 2

Description: Posting a malicious message to display cookies

Evidence:

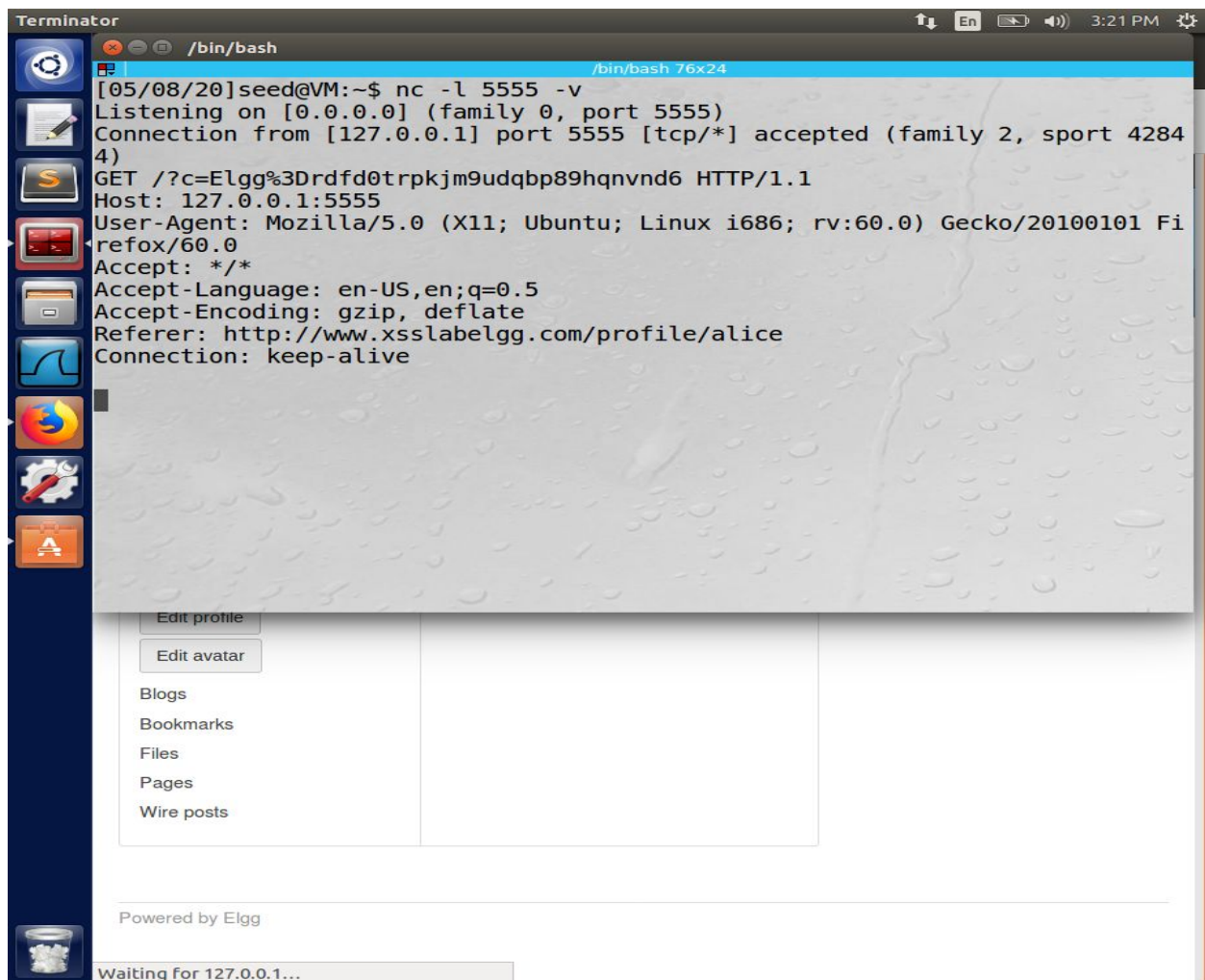


Analysis: Similar to task 1, but instead we printed out the document's cookie information.

3.4, Task 3

Description: Stealing Cookies from the victim's machine so the attacker can see the info

Evidence:



The screenshot shows a Linux desktop environment. In the foreground, a Terminator terminal window is open, displaying the following text:

```
/bin/bash  
[05/08/20]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 42844)  
GET /?c=Elgg%3Drdfd0trpkjm9udqbp89hqnvnd6 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/alice  
Connection: keep-alive
```

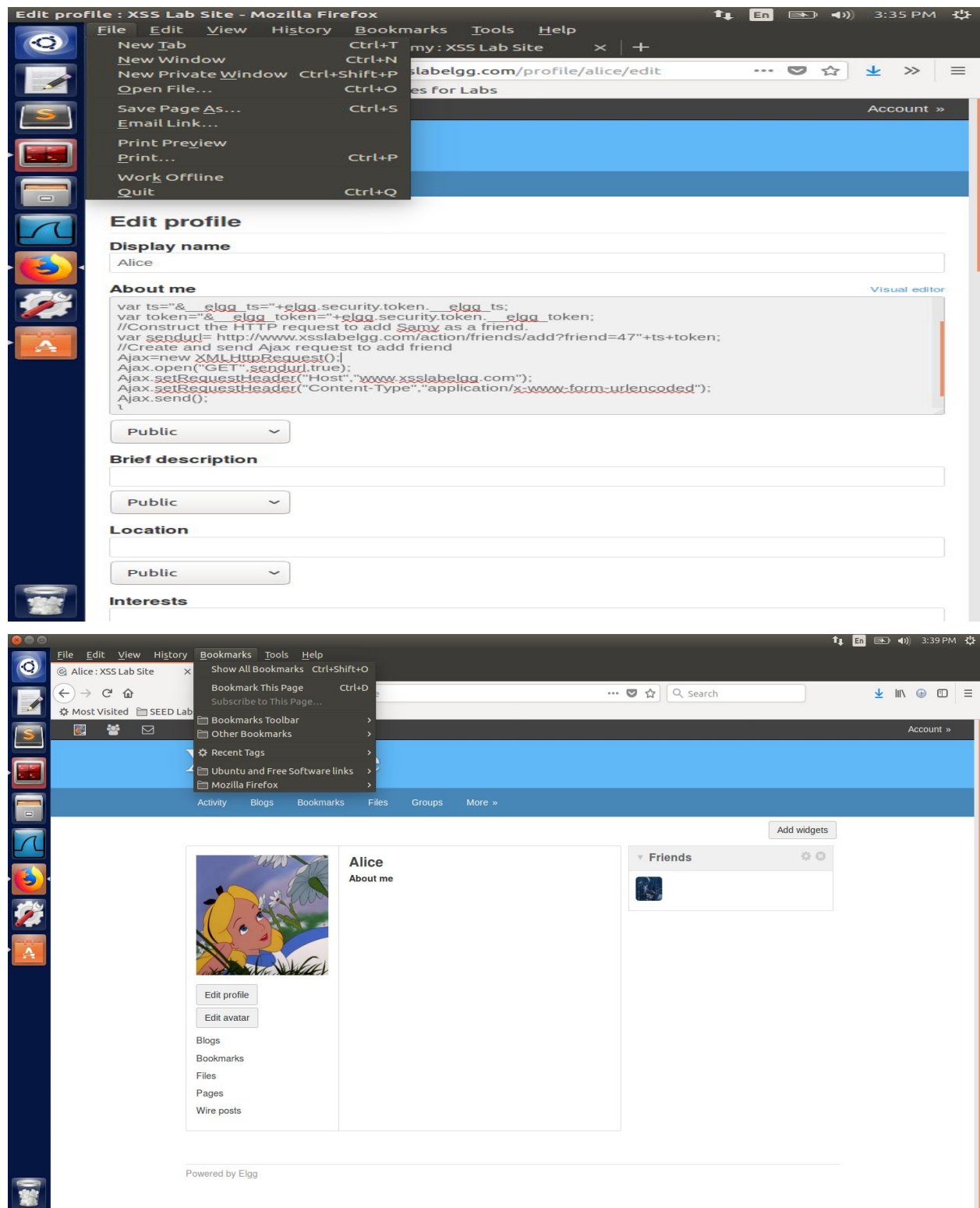
In the background, a web browser window is visible, showing a profile page for 'alice' on the Elgg platform. The page includes buttons for 'Edit profile' and 'Edit avatar', and links for 'Blogs', 'Bookmarks', 'Files', 'Pages', and 'Wire posts'. The footer of the browser window says 'Powered by Elgg' and 'Waiting for 127.0.0.1...'.

Analysis: My machine was able to steal the cookies by creating malicious code that requested a resource (image) but used our IP address and redirected the request to a port 5555 which is the active TCP port listening. The picture above shows the cookie information of that user.

3.5, Task 4

Description: Becoming the victim's friend by writing an XSS worm

Evidence:

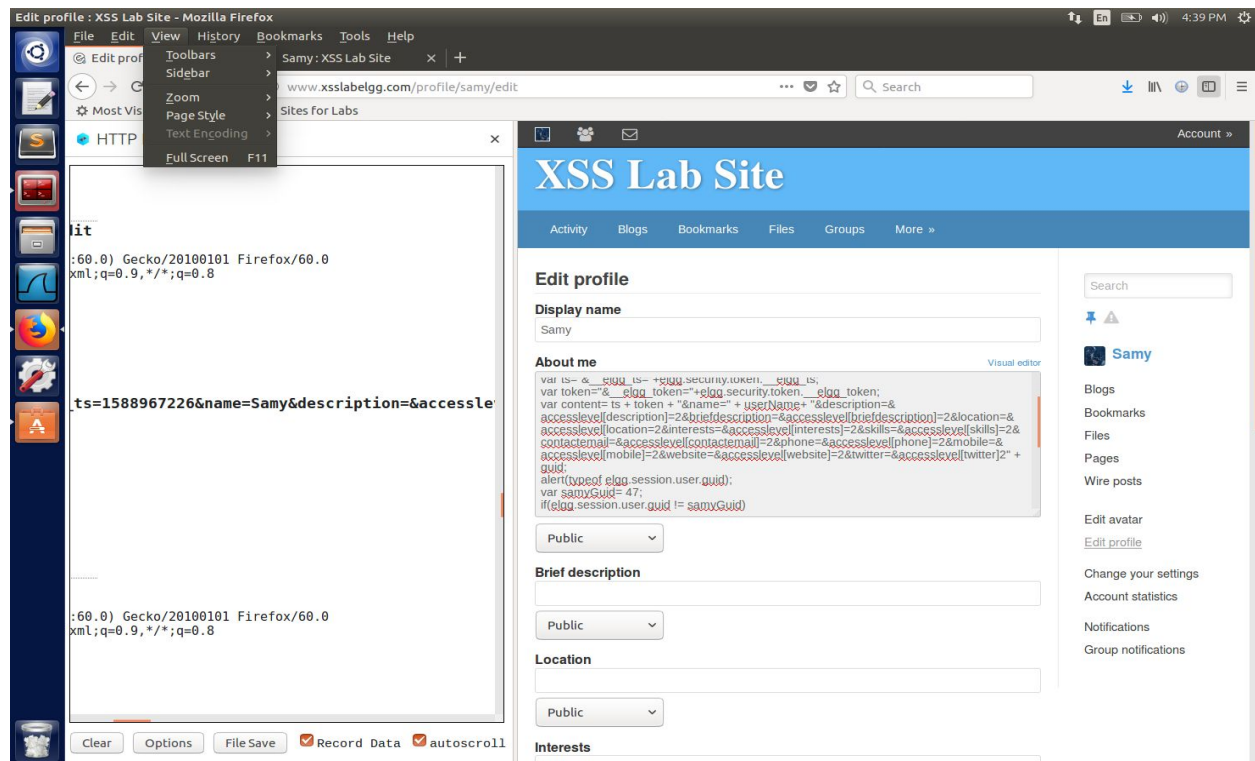


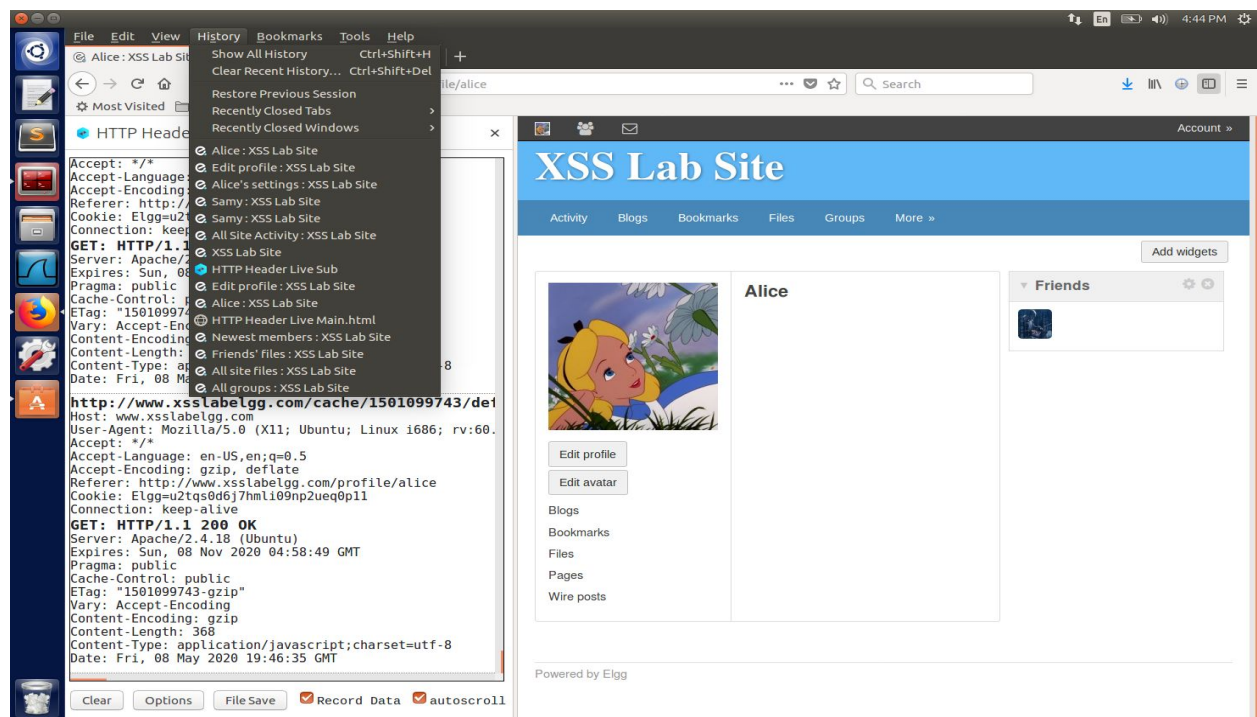
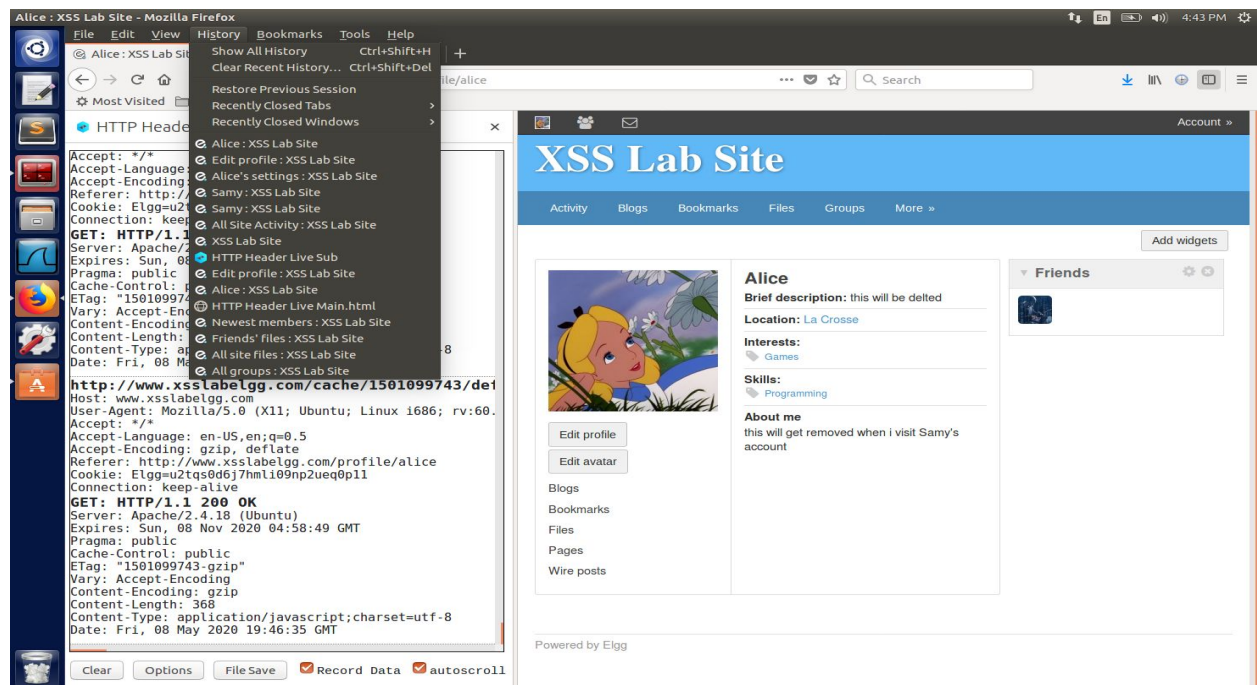
Analysis: After modifying the given code to include the send url of Samy, when Alice visited Samy's profile, Alice added Samy as a friend. Lines 1 and 2 are needed because they are part of the send url for our action to occur. If the editor only had editor mode then the code would not work because there is extra HTML code wrapping our javascript code preventing the script from ever being able to run but only rendered as text.

3.6, Task 5

Description: Modifying the victim's profile by writing an XSS worm.

Evidence:



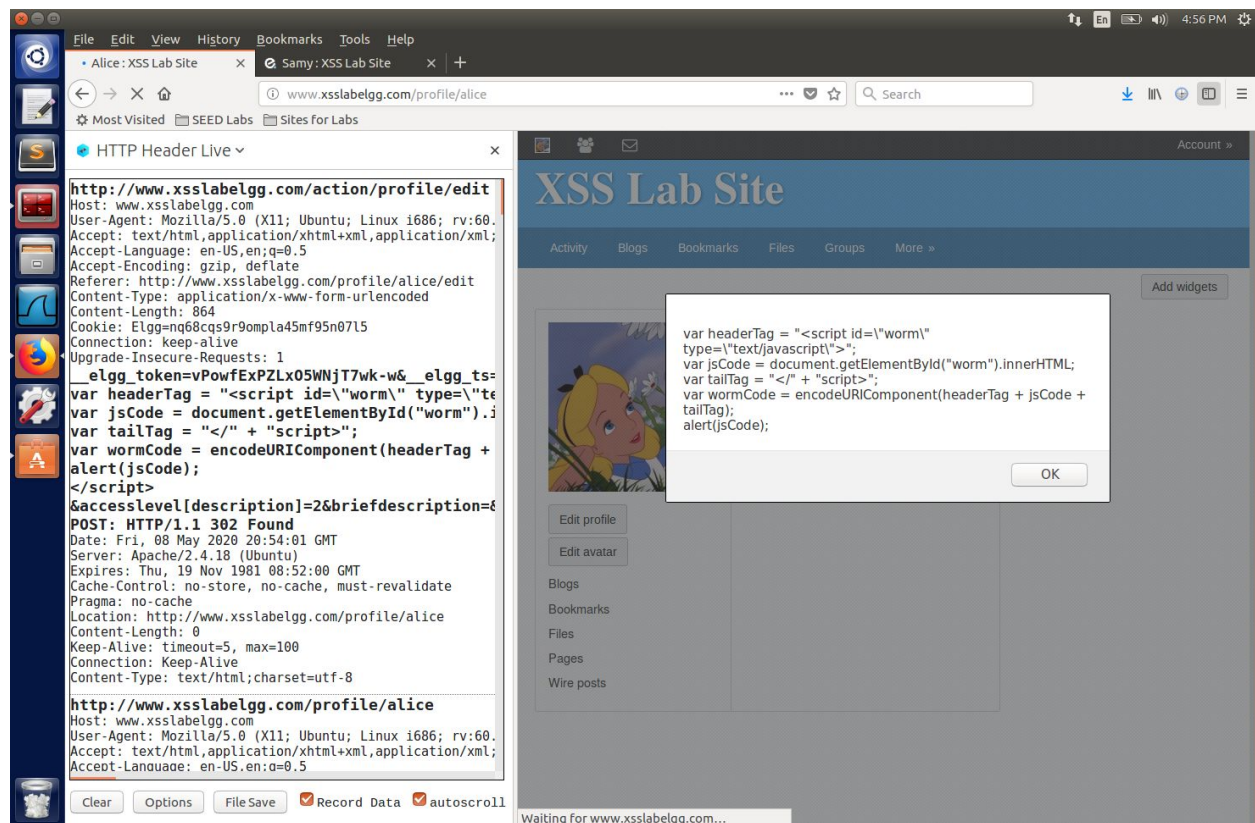


Analysis: First to obtain the content url, by looking at the contents inside HTTP Header Live you could find a POST request to update. There was a very long post request that contained the structural request for making an update. I simply copied that format but used the given variables to create a generic content of Samy's current profile content. Alice's profile information was filled out but after she visited Samy's profile then Alice's profile became blank due to the content updating the profile account. If we were to remove the if statement in our code then if Samy were to log in then her profile info would be erased thus erasing our code.

3.7, Task 6

Description: Writing a self-propagating XSS worm

Evidence:

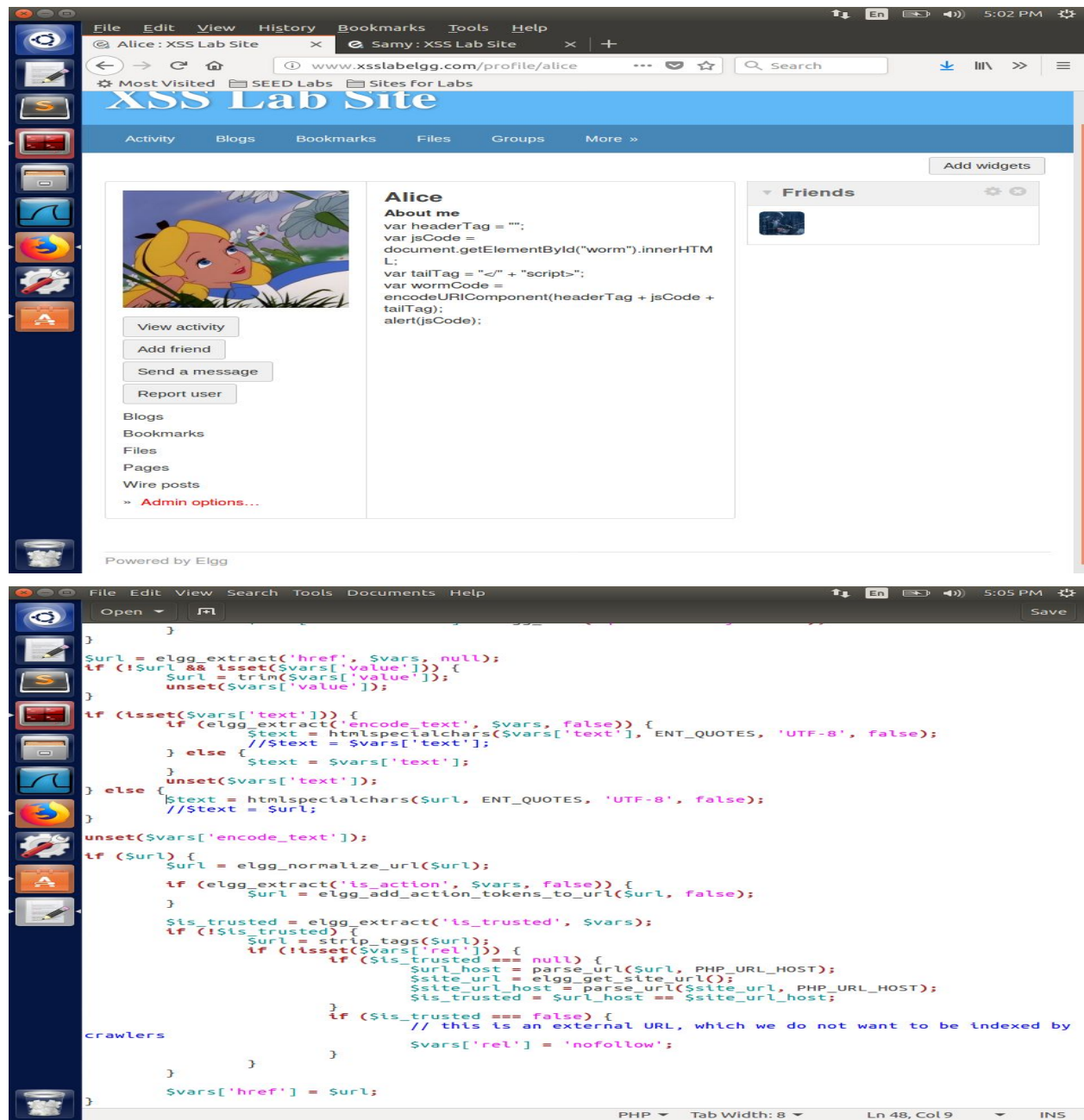


Analysis: Basically what could've happened to create a self propagating worm is to inject malicious javascript code inside the jsCode variable which would cause a request to happen. But in these examples you can see the potential of creating a self propagating code.

3.7, Task 6

Description: Writing a self-propagating XSS worm

Evidence:



Analysis: When I activated the HTMLawed countermeasure, the script code that used to run does not run anymore and is just treated as text. I'm assuming that any dangerous html code was stripped out before displaying. As for the htmlspecialchars feature, when I turned that feature on I did not notice much but realized that non-alphanumeric characters were replaced with hexadecimal values in the URI. So a bit of processing on the values must happen in order to prevent script code from being written and executed.

Conclusion: In the lab I went through basic ideas of how to create XSS attacks and how a website could be vulnerable if not handled properly. XSS attacks seem quite dangerous where lots of personal information could be leaked or infect others to cause a massive attack. I think with a bit of sanitation on the given input strings that contain script code can be modified to prevent malicious code from executing.