**Saishnu Ramesh Kumar (300758706)**

**CSCI 191T (Assignment 3) – Race Condition & CSRF**

**Section 1:**

1. No, this program does not have a race condition vulnerability. This is because the /etc/passwd is a protected file which means that the user will not have access to write into it therefore the block will not execute.
2. If we disable the root privileges before the strcpy() and enable it afterwards, this will not work because the attack happens when we return from the bof(). But if we disable the root privileges before the bof() function call and enable it after that, the attack can be prevented.
3. Yes, there is the race condition vulnerability in this program. The access() system will call to check if the REAL USER ID has the ability to write into the file and if the ID does have the confirmation, the file can be written into. The attack window is found between the status variable and write\_to\_file function and we can use this time window to check the status and write into it by pointing it to an already existing file like the /etc/passwd file.

4a) If the case of request originates from one site only, it will be sent if the same site cookie is also the session cookie. In the case of CSRF attacks, even though the attacker redirects you to the malicious site and makes a request to the original site to exploit your sensitive information, it will fail. This is because the browser will not send the session cookie since the malicious site has a different origin than the original site and this would result in the malicious code from executing since the user is logged out of the original website.

4b) Websites nowadays use anti-CSRF tokens as a secret token protection from the CSRF token. These are always saved in a session variable. The token is hidden in a form field that is used to send through the request. The web application will only allow the request to go through if the value of the session variable and the form field that is hidden are matching.

5) We can redirect to <http://www.example.com/delete.php?pageid=5> by the following:

**Code:**

<html>

<body>

<h1> Delete a page owned by the user. </h1>

<img src = “<http://www.example.com/delete.php?pageid=5>” />

</body>

</html>

6) No, we cannot. Cross-site requests are utilized in numerous scenarios for example, by using your credentials of your Google account to login to YouTube or even from Facebook to Instagram. If the browser is not attaching the cookies, the authentication fails.

**Section 2:**

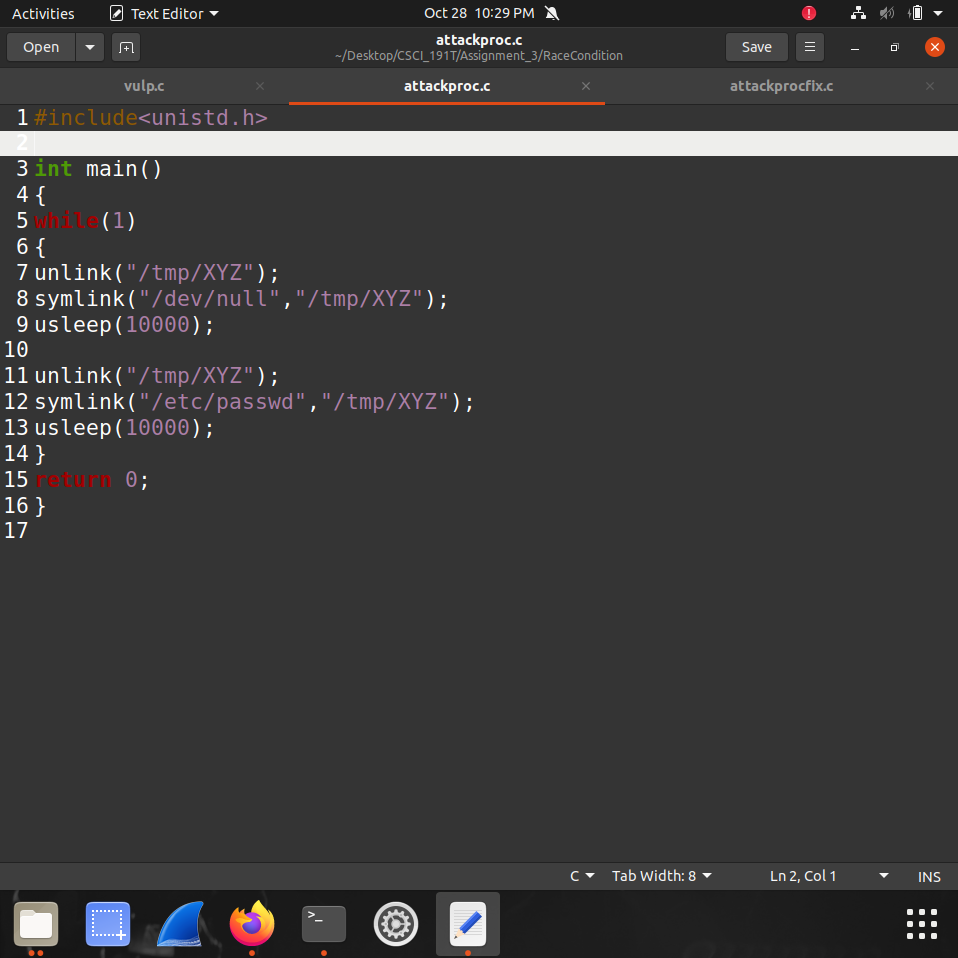
**PART 1:**

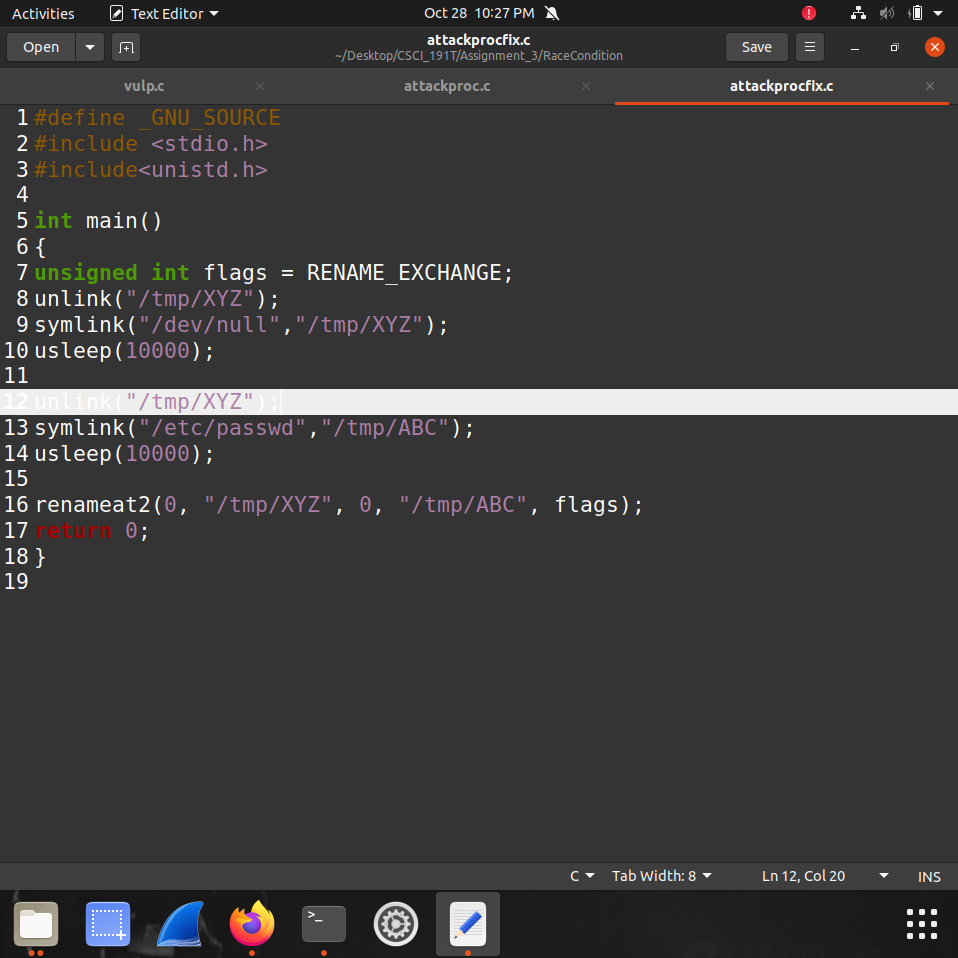
**Step 1:** Turning off all countermeasures to allow attack to take place.

Graphical user interface, text, application

Description automatically generated

**Step 2:** The program that allows the attack to take place. Therefore, we created a fixed program to prevent the attack (seen below in screenshots).





**Step 3:** Compiling the original program to execute the attack.

Text

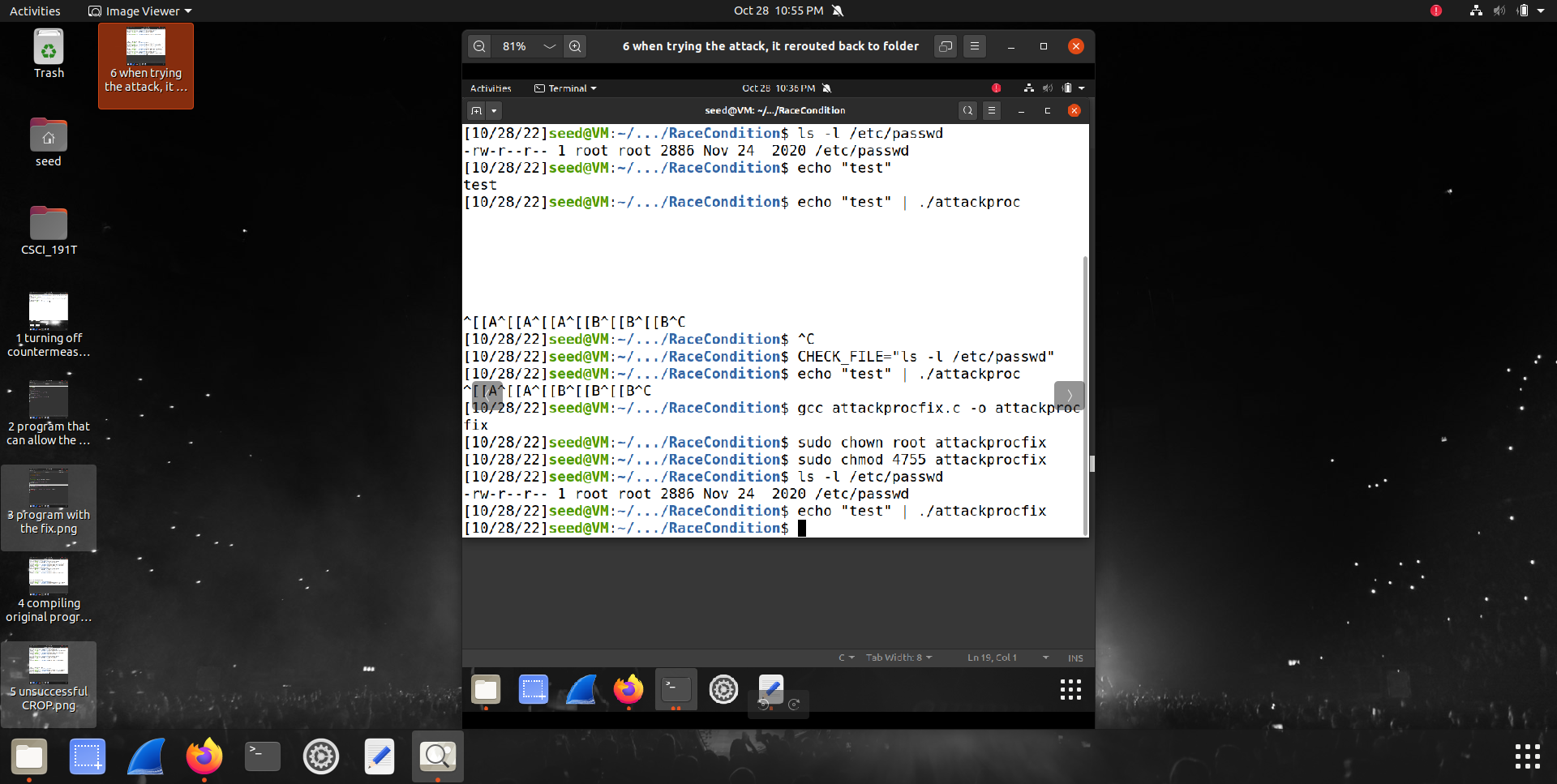
Description automatically generated

**Step 4:** The attack was unsuccessful, after waiting for a while there was no response within the terminal.

A picture containing text

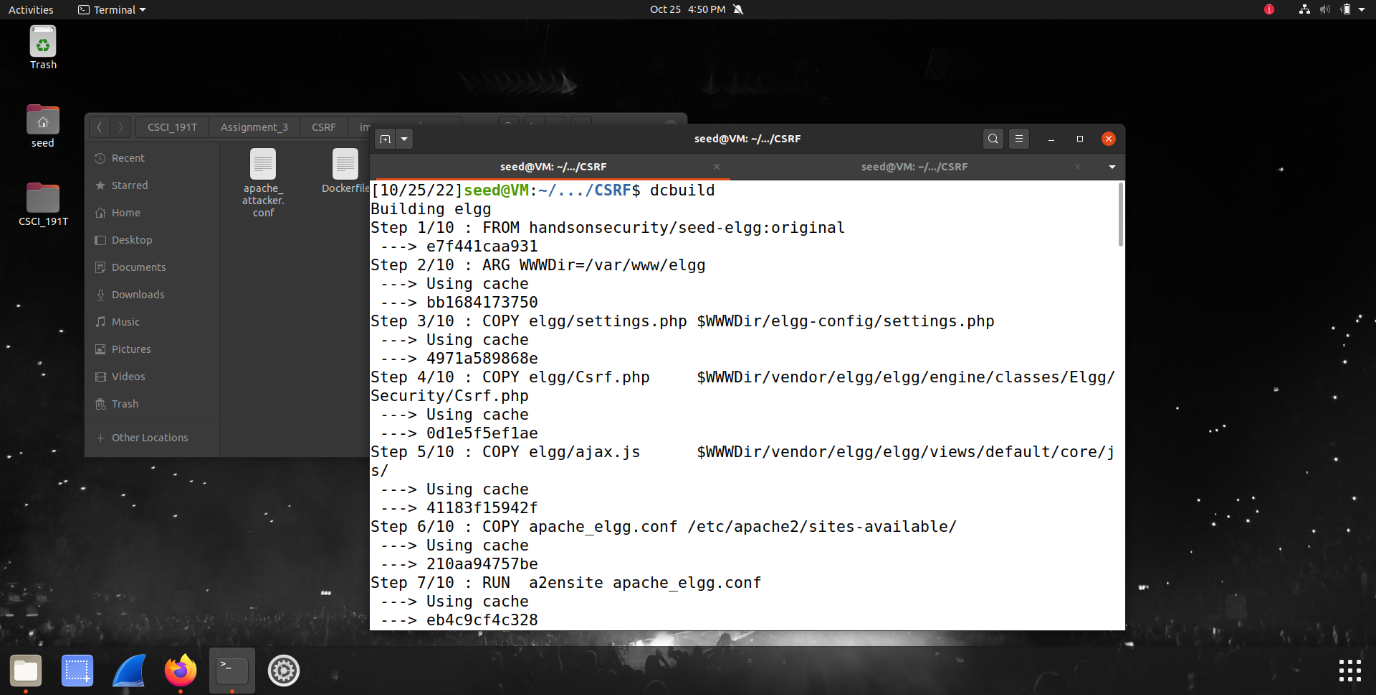
Description automatically generated

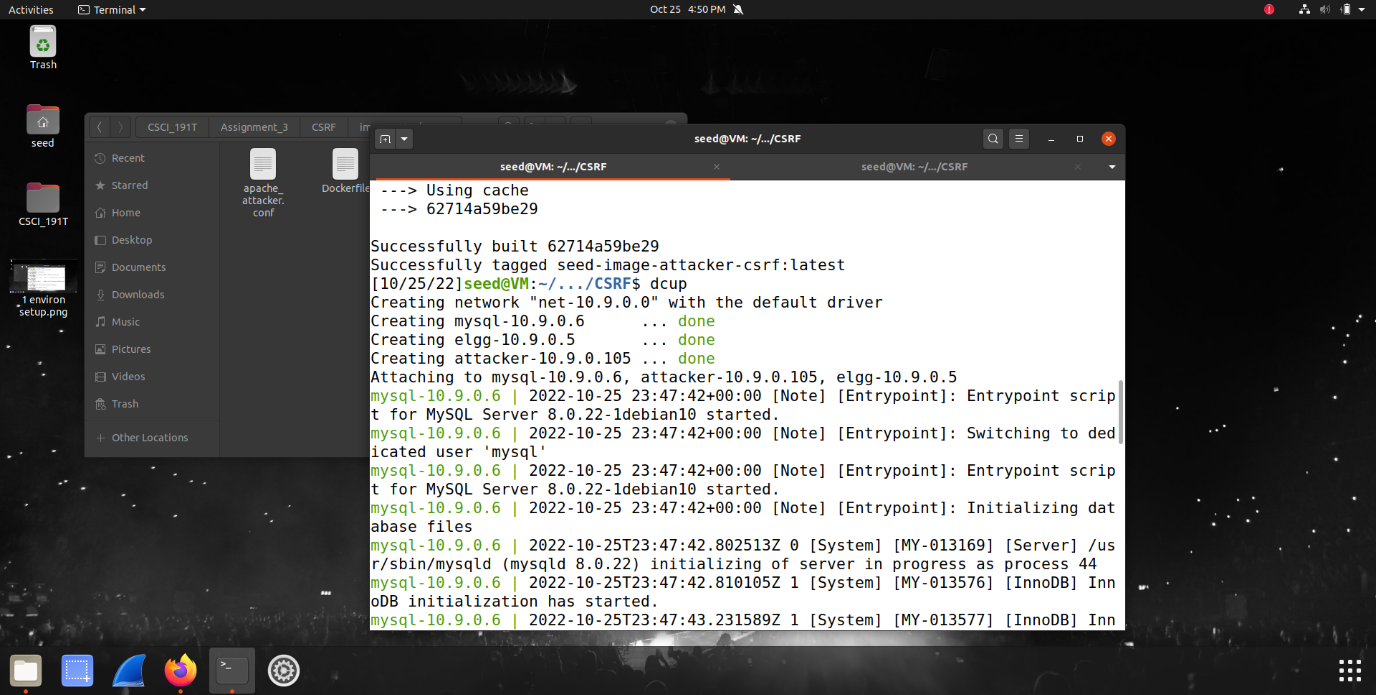
**Step 5:** But when attempting the attack with the fixed program, it reroutes back to the folder the program was on.



**PART 2:**

**Step 1:** Setting up the environment to run the Samy is my Hero attack. Using dcbuild and dcup to start connection to server (as seen in both screenshots below).

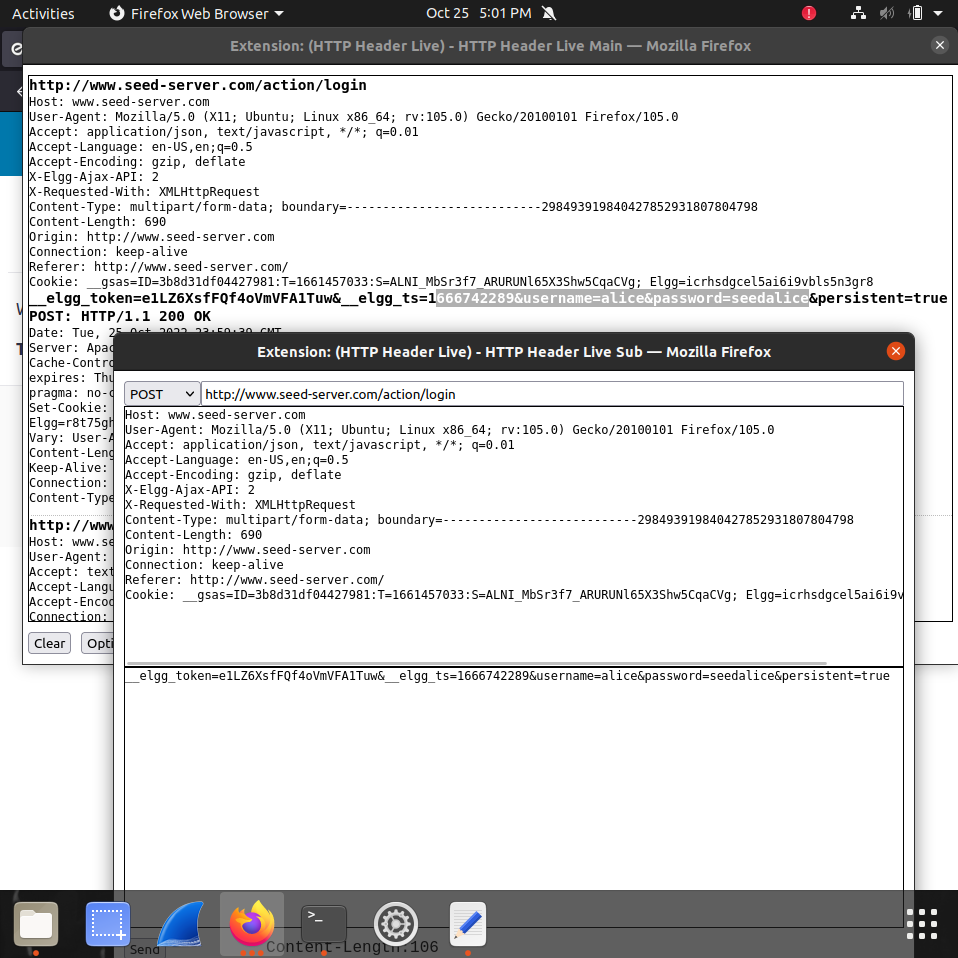




**Step 2:** Adding the respective IP addresses to the /etc/hosts.



**Step 3:** Getting a post request while being logged into active session (Alice’s account).



**Step 4:** While logged in as Samy, we add Alice through Samy’s account as a friend.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Step 5:** We then view the page source of Samy’s active session to get Samy’s GUID number, which is 59.

A screenshot of a computer

Description automatically generated with medium confidence

**Step 6:** Accessing the docker shell while using the information obtained from dockps regarding the attacker IP address.

Graphical user interface, text, application

Description automatically generated

**Step 7:** Inside the addfriend.html file, we then add Samy’s GUID url to the file.

A screenshot of a computer

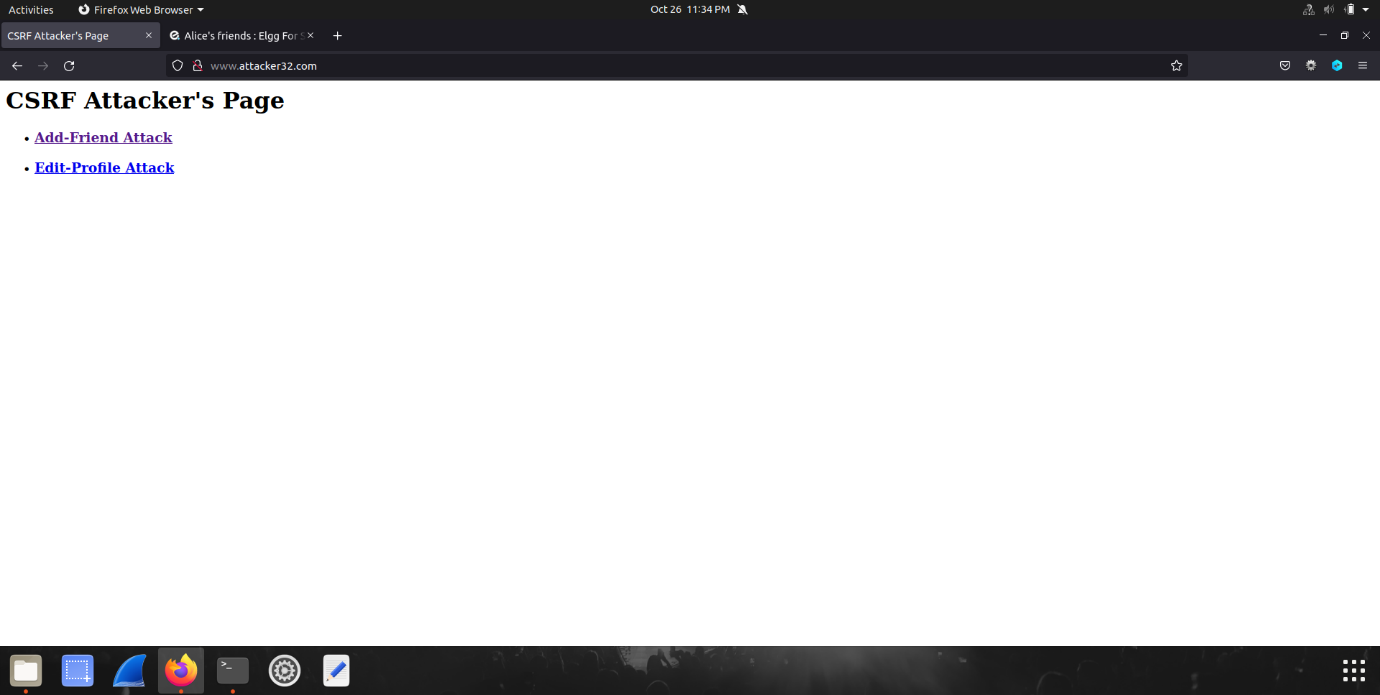
Description automatically generated

**Step 8:** We get the docker to copy the addfriend file and upload it to the attacker server.

Graphical user interface, text, application

Description automatically generated

**Step 9:** Once logged into Alice as an active session, we go to the attacker32.com website to initiate the attacker on Alice as Samy by adding Samy as Alice’s friend on her profile.



**Step 10:** We then click on the add friend attack to allow the attack to start.

Graphical user interface, text

Description automatically generated

**Step 11:** After several times of refreshing the page, we then get a successful response from the attack, Samy is now Alice’s friend.

A screenshot of a computer

Description automatically generated

**Step 12:** We log into Samy’s account and edit his about me and brief description stating “Samy is my Hero.”

A screenshot of a computer

Description automatically generated

**Step 13:** We then open the editprofile.html file to continue the attack process. We add in Alice’s GUID, name and the description to be shown on her profile for a successful attack.

A screenshot of a computer

Description automatically generated

**Step 14:** After saving the file and clicking on the edit profile attack from the attacker32.com website, we then refresh the page and finally get the attack completed. As seen below, Alice has now been attacked, with the description on her page stating “Samy is my Hero.”

A screenshot of a computer

Description automatically generated