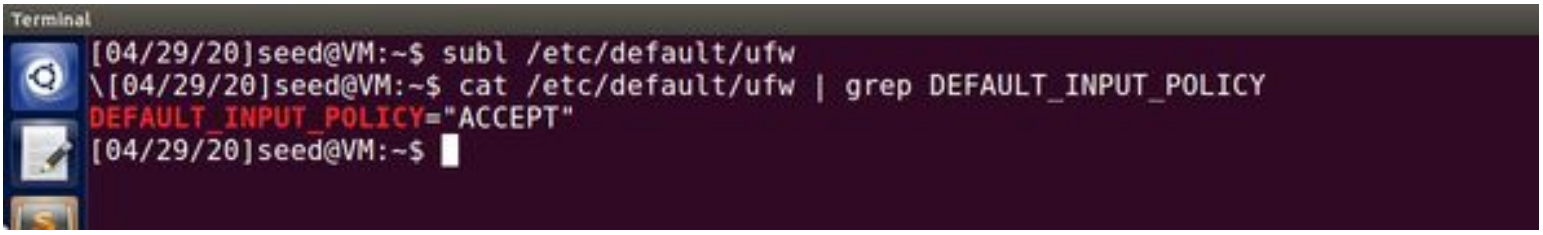
Task 1 : Using Firewall:

In this task we'll make use of linux's infuild tool iptables which also uses a frontend tool called ufw to setup dome firewall policies and observe our system's behaviour when the policies come into effect. Before performing the tasks, we first enable the firewall to accept the incoming traffic. Here is what we do for that, I changed the default value of **DEFAULT_INPUT_POLICY** from drop to accept.



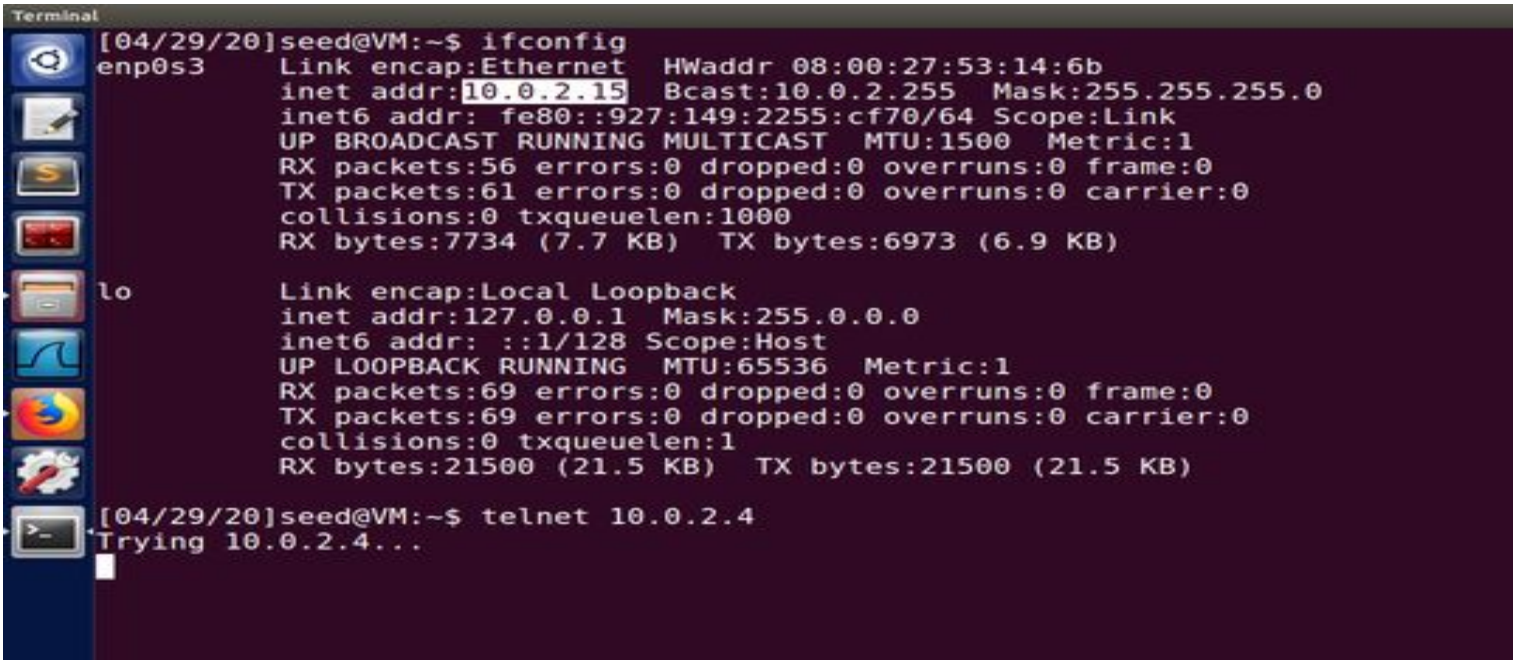
We know by default the telnet runs on port 23. Making use of ufw's deny with the port number 23 as an argument, we can stop other machines accessing the host through telnet as the port 23 is denied access. Below is the screenshot of that being performed.

Prevent A from doing `telnet` to Machine B:

The two machines **B** and **A** I'm using for this attack have IP's **10.0.2.4** and **10.0.2.15** respectively. In the below screenshot we can see I have disabled telnet running on port 23 of Machine A using ufw's deny to port 23.

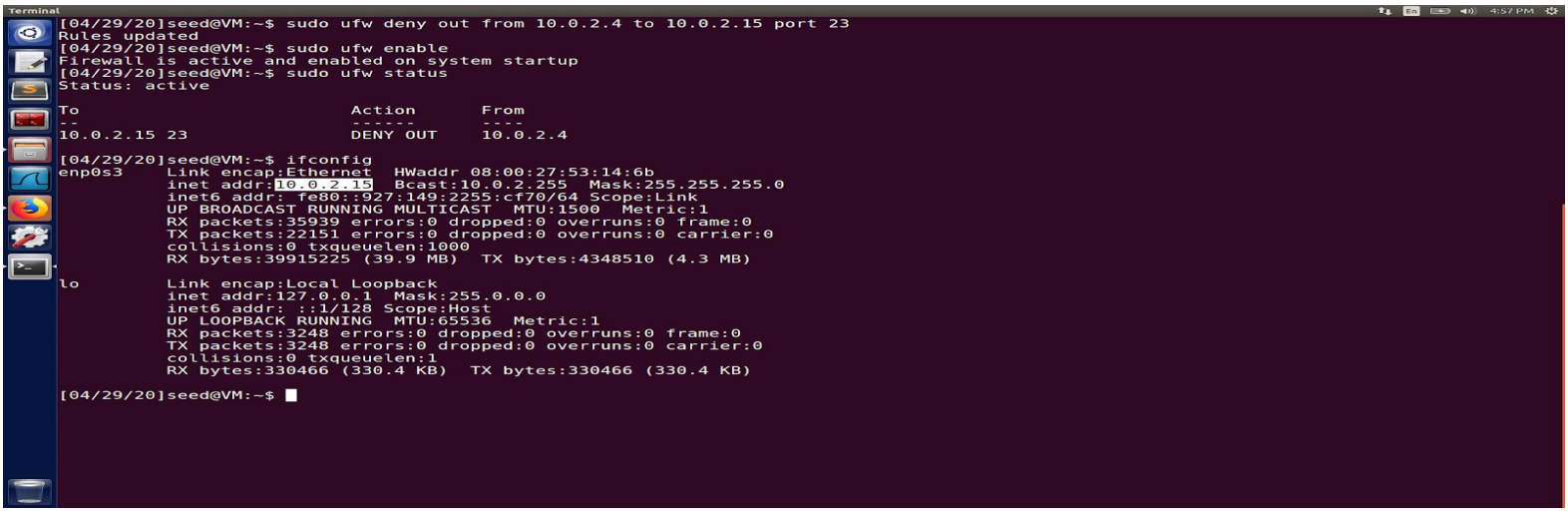
We make use of the command **sudo ufw deny out from 10.0.2.15 to 10.0.2.4 port 23** Doing this denies access to Machine **B**'s Telnet from Machine **A**. And once we execute the command the rules get updated and to enable the updated rules, we run **sudo ufw enable.**

Once the above commands are run, the firewall is now enabled.
In the below screenshot we can see how Machine **A** trying to access machine **B** is failing.

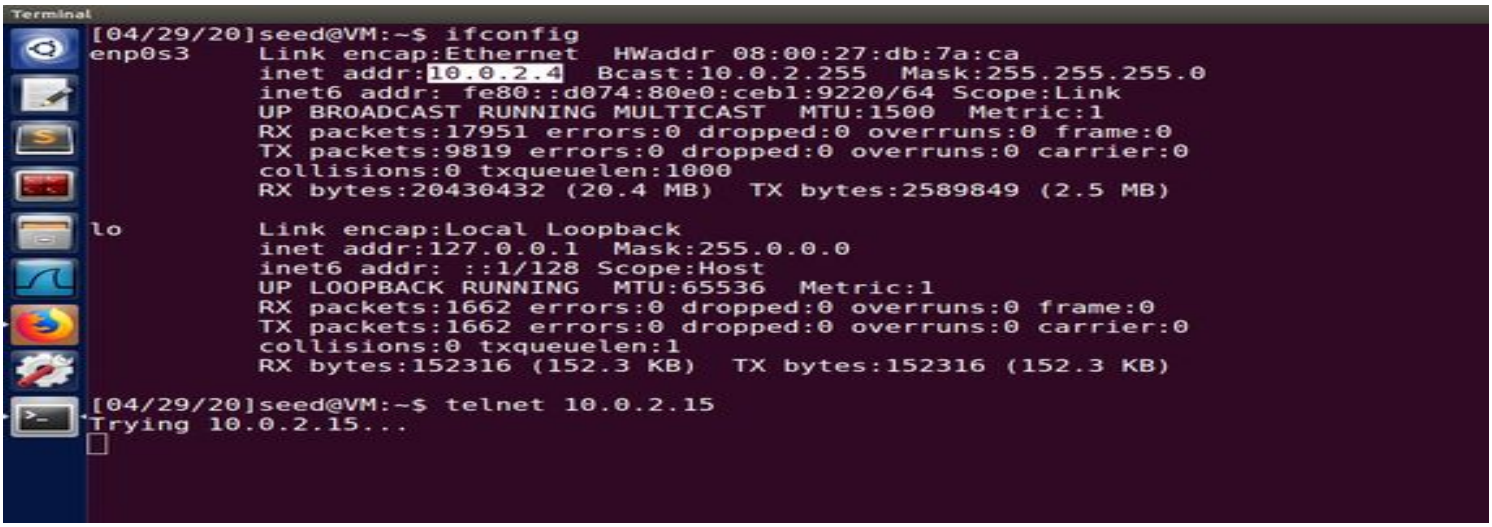


Prevent B from doing `telnet` to Machine A:

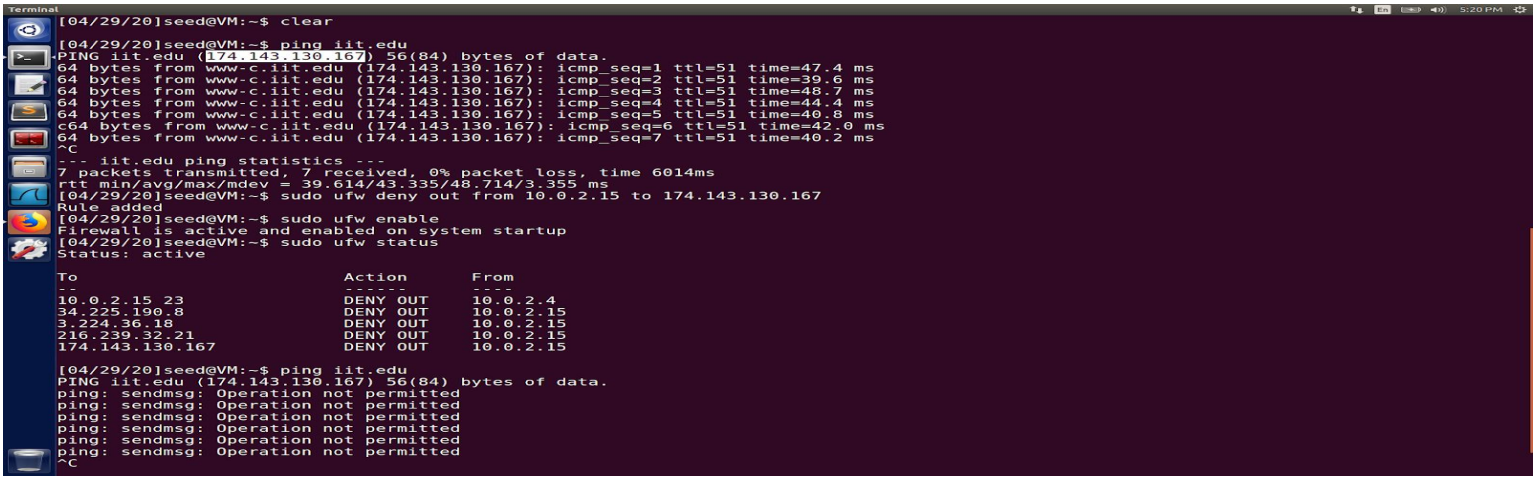
The two machines **B** and **A** I'm using for this attack have IP's **10.0.2.4** and **10.0.2.15** respectively. In the below screenshot we can see I have disabled telnet running on port 23 of Machine A using ufw's deny to port 23.
We make use of the command **sudo ufw deny out from 10.0.2.4 to 10.0.2.15 port 23**
Doing this denies access to Machine A's Telnet from Machine B. And once we execute the command the rules get updated and to enable the updated rules, we run **sudo ufw enable.**



Once the above commands are run, the firewall is now enabled.
In the below screenshot we can see how Machine **B** trying to access machine **A** is failing.



<u>Prevent A from visiting an external web site. You can choose any website that you like to block, but keep in mind, some web servers have multiple IP addresses:</u>
In this task, I'll try to ping iit.edu and we can see it's IP being **174.143.130.167**
Therefore to disable access to instagram.com from Machine **A**, we update the firewall using the below command. **sudo ufw deny out from 10.0.2.15 to 174.143.130.167**
And once the above command is executed, it updates all the rules and to enable it, we run **sudo ufw enable.** And now, once the firewall rules are enabled, when we try to ping **iit.edu**, we can see the ping being failed making out task successful.



The reason for using iit.edu being it's IP did not change with each ping command. I tried using instagram.com… but each time it's IP kept changing making it hard to update the firewall each time.

Task 2: Implementing a Simple Firewall

In this task I extended the code given in the textbook code from here to perform various tasks defined in the handout.

Task 2.1: **Prevent A from doing `telnet` to Machine B:**

For this task, I have chosen Machine A with an IP 10.0.2.4 and Machine B that has an IP 10.0.2.5. The code that I'm using to prevent Machine A from doing telnet to Machine B is shown below.

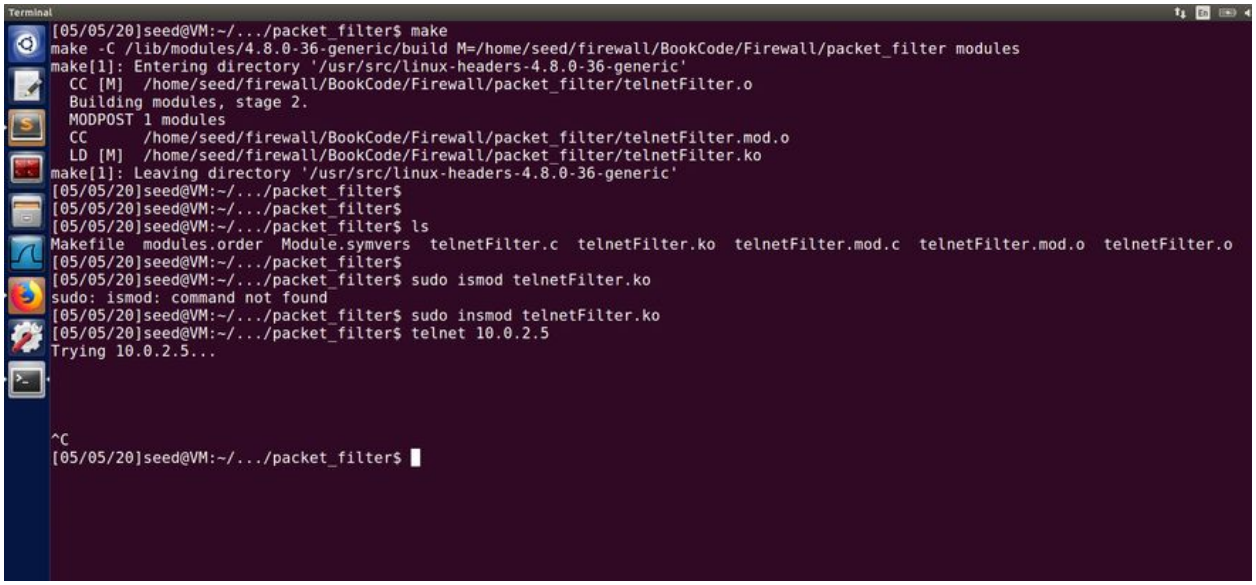
telnetFilter.c Makefile

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/inet.h>

static struct nf_hook_ops telnetFilterHook;


unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                  const struct nf_hook_state *state)
{
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = ip_hdr(skb);
  tcph = (void *)iph+iph->ihl*4;

  if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && iph->saddr == in_aton("10.0.2.4") && iph->daddr == in_aton("10.0.2.5")) {
    printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr)[0],
        ((unsigned char *)&iph->daddr)[1],
        ((unsigned char *)&iph->daddr)[2],
        ((unsigned char *)&iph->daddr)[3]);
    return NF_DROP;
  } else {
    return NF_ACCEPT;
  }
}


int setUpFilter(void) {
        printk(KERN_INFO "Registering a Telnet filter.\n");
        telnetFilterHook.hook = telnetFilter;
        telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
        telnetFilterHook.pf = PF_INET;
        telnetFilterHook.priority = NF_IP_PRI_FIRST;
        nf_register_hook(&telnetFilterHook);
        return 0;
}

void removeFilter(void) {
        printk(KERN_INFO "Telnet filter is being removed.\n");
        nf_unregister_hook(&telnetFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);
```

The content of the Makefile is given below

```
obj-m += telnetFilter.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

In the screenshot below, we can see I compiled the code using make command and then I inserted the kernel module using the insmod command. Once the kernel module is inserted, I try to execute the telnet to Machine B with an IP 10.0.2.5. We can see, it fails to execute the telnet command making the task successful. That is because a tiny firewall is being implemented on executing the code that required us to load our code into the kernel using the hooks provided by the netfilter.



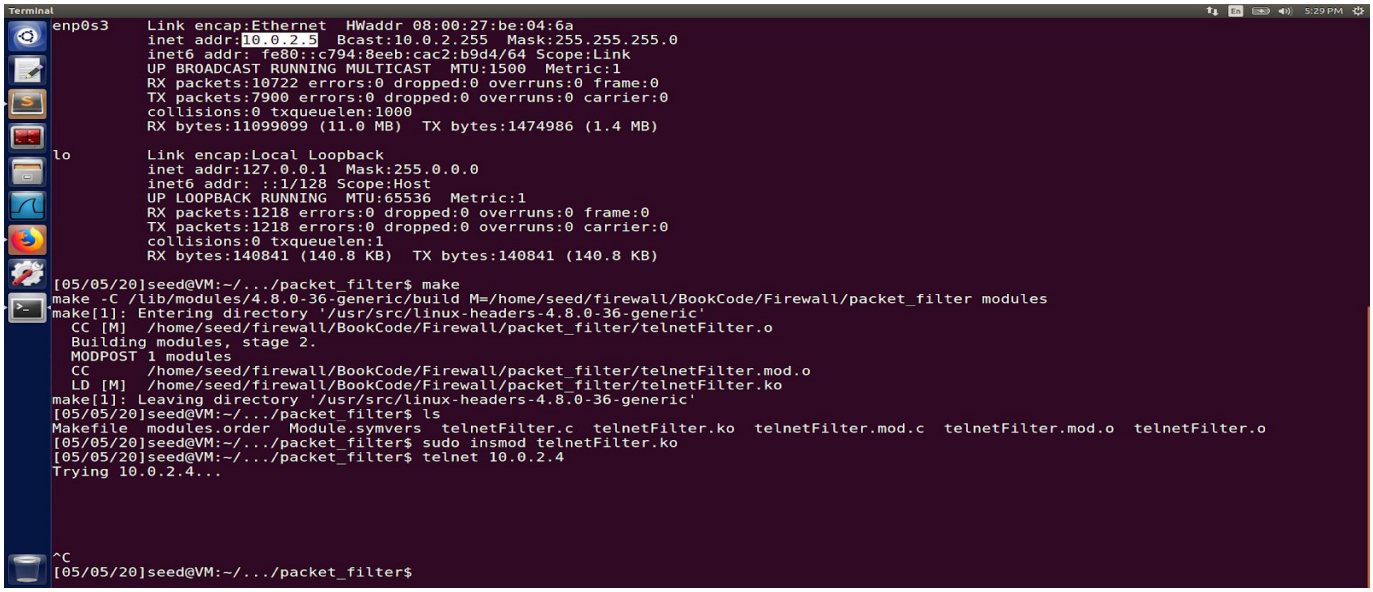
Task 2.2: Prevent B from doing `telnet` to Machine A:

This task is just the opposite of **task 2.1**, I have chosen Machine A with an IP 10.0.2.4 and Machine B that has an IP 10.0.2.5. The code that I'm using to prevent Machine B from doing telnet to Machine A is shown below. Here the code too is very similar to the code of the

previous one except for that the if condition following from line 21 in the code screenshot of the previous task is changed to

```
if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23) && iph->saddr ==
in_aton("10.0.2.5") && iph->daddr == in_aton("10.0.2.4")) {
        printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr)[0],
        .....
        ((unsigned char *)&iph->daddr)[3]);
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
```

This change employs the telnet is being prevented from Machine B with IP 10.0.2.5 to Machine A with IP 10.0.2.4. The Makefile remains the same. In the screenshot below, we can see from the highlighted section that the program is running from Machine B with an IP 10.0.2.15. I then compiled the code using make command and then I inserted the kernel module using the insmod command. Once the kernel module is inserted, I try to execute the telnet to Machine A with an IP 10.0.2.4. We can see, it fails to execute the telnet command making the task successful. That is because a tiny firewall is being implemented on executing the code that required us to load our code into the kernel using the hooks provided by the netfilter.



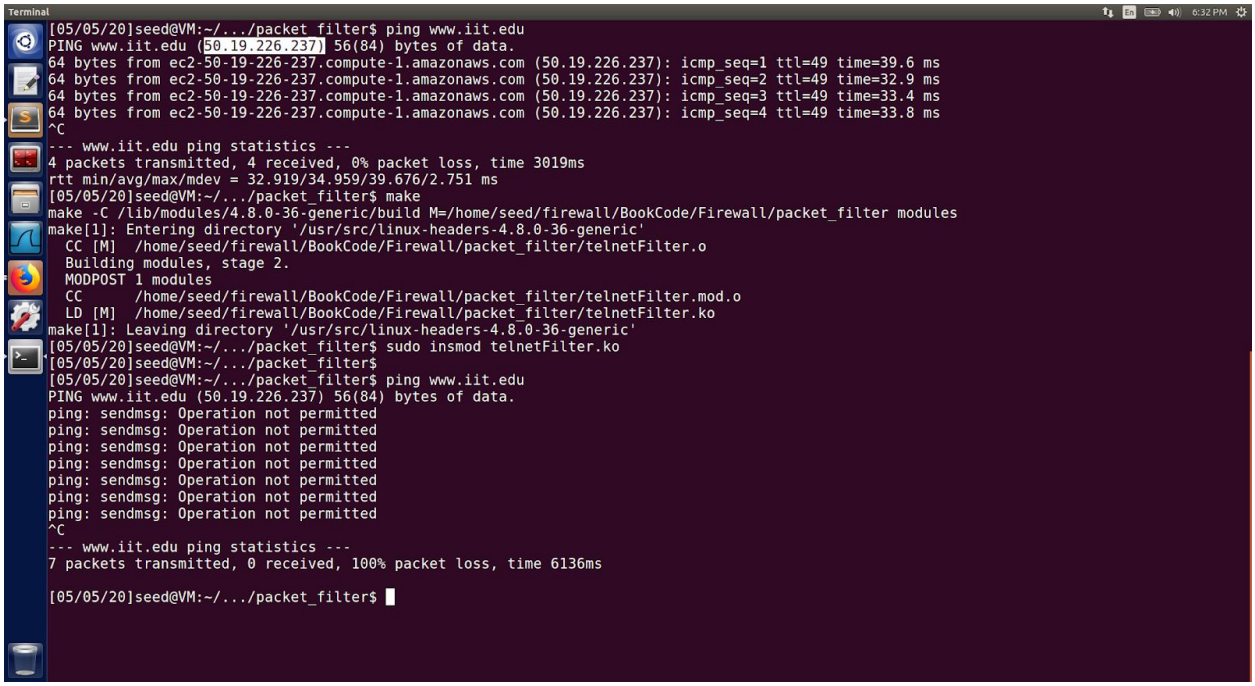
Task 2.3: Prevent A from visiting an external web site. You can choose any website that you like to block, but keep in mind, some web servers have multiple IP addresses:

This task is very similar to **task 2.2**, I have chosen Machine A with an IP 10.0.2.4 and an external site as www.iit.edu with an IP 50.19.226.237. The code that I'm using to prevent Machine A from Accessing www.iit.edu is shown below. Here the code too is very similar to the code of the previous one except for that the if condition following from line 21 in the code screenshot of the previous task is changed to

```
if (iph->saddr == in_aton("10.0.2.4") && iph->daddr == in_aton("50.19.226.237")) {
        printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr)[0],
        .......
        ((unsigned char *)&iph->daddr)[3]);
        return NF_DROP;
} else {
        return NF_ACCEPT;
}
```

This change employs the access being denied to www.iit.edu from Machine A with IP 10.0.2.4. The Makefile remains the same.

In the screenshot below, we can see from the highlighted section that the IP of www.iit.edu is 50.19.226.237. I then compiled the code using make command and then I inserted the kernel module using the insmod command. Once the kernel module is inserted, I try to ping www.iit.edu . We can see, it fails to execute the ping command making the task successful. That is because a tiny firewall is being implemented on executing the code that required us to load our code into the kernel using the hooks provided by the netfilter.



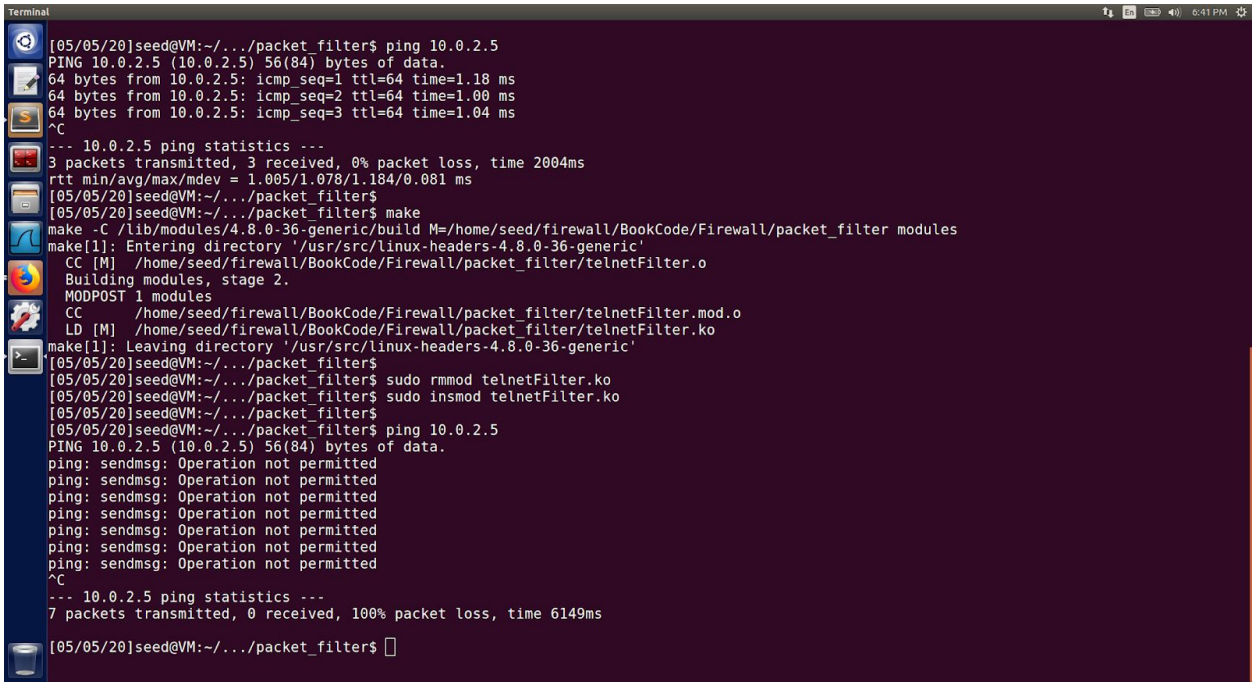
Task 2.4: Prevent A from sending ICMP to any machine

This task is very similar to **task 2.3**, I have chosen Machine A with an IP 10.0.2.4 and Machine B with an IP 10.0.2.5. The code that I'm using to prevent Machine A from sending ICMP packets to Machine B is shown below. Here the code too is very similar to the code of the previous one except for that the if condition following from line 21 in the code screenshot of the previous task is changed to

```
if (iph->protocol == IPPROTO_ICMP && iph->saddr == in_aton("10.0.2.4")) {
        printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr)[0],
        ((unsigned char *)&iph->daddr)[1],
        ((unsigned char *)&iph->daddr)[2],
        ((unsigned char *)&iph->daddr)[3]);
        return NF_DROP;
} else {
        return NF_ACCEPT;
}
```

This change employs that ping to any machine from Machine A with IP 10.0.2.4 is denied. The Makefile remains the same.

In the screenshot below, we can see the ping to machine B with IP 10.0.0.5 is successful in the beginning. I then compiled the code using make command and then I inserted the kernel module using the insmod command. Once the kernel module is inserted, I try to ping the same Machine B . We can see, it fails to execute the ping command making the task successful. That is because a tiny firewall is being implemented on executing the code that required us to load our code into the kernel using the hooks provided by the netfilter.



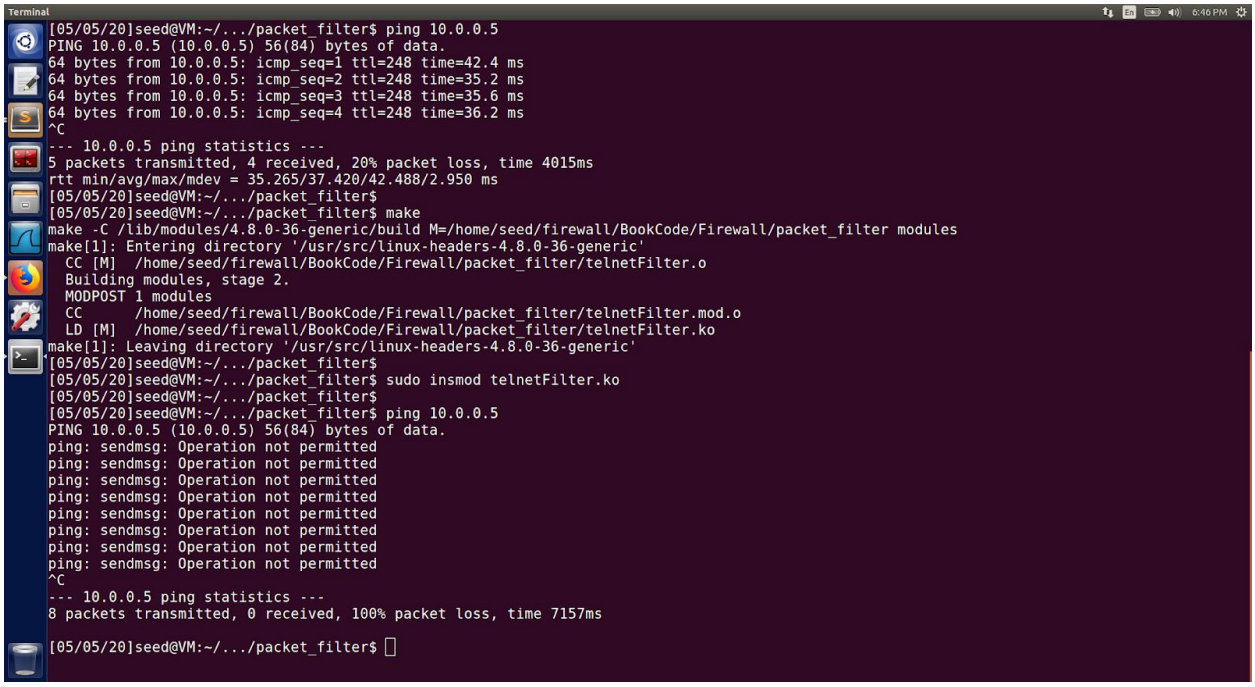
Task 2.5: Prevent B from sending ICMP to any machine

This task is very similar to **task 2.4**, I have chosen Machine A with an IP 10.0.2.4 and Machine B with an IP 10.0.2.5. The code that I'm using to prevent Machine B from sending ICMP packets to Machine A is shown below. Here the code too is very similar to the code of the previous one except for that the if condition following from line 21 in the code screenshot of the previous task is changed to

if (iph->protocol == IPPROTO_ICMP && iph->saddr == in_aton("10.0.2.5")) {

```
        printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
        ((unsigned char *)&iph->daddr)[0],
        ((unsigned char *)&iph->daddr)[1],
        ((unsigned char *)&iph->daddr)[2],
        ((unsigned char *)&iph->daddr)[3]);
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
```

This change employs that ping to any machine from Machine **A** with IP 10.0.2.5 is denied. The Makefile remains the same. In the screenshot below, we can see the ping to machine A with IP 10.0.0.5 is successful in the beginning. I then compiled the code using make command and then I inserted the kernel module using the insmod command. Once the kernel module is inserted, I try to ping the same Machine A . We can see, it fails to execute the ping command making the task successful. That is because a tiny firewall is being implemented on executing the code that required us to load our code into the kernel using the hooks provided by the netfilter.



With this I have specified 5 different rules including the ones we've performed in task 1.

Task 3 : Evading egress Filtering

In this task we make use of tunnels to bypass egress filtering implemented. Firstly even before we proceed further with the tasks below, we setup the firewalls to make sure we're ready to perform the tasks listed below. Here I'm using www.iit.edu instead of www.facebook.com as facebook has multiple static IP's which makes it difficult to perform the following attacks.

I'll be performing the tasks from Machine A which has an IP of 10.0.2.4. And here are the commands to set the firewall rules.

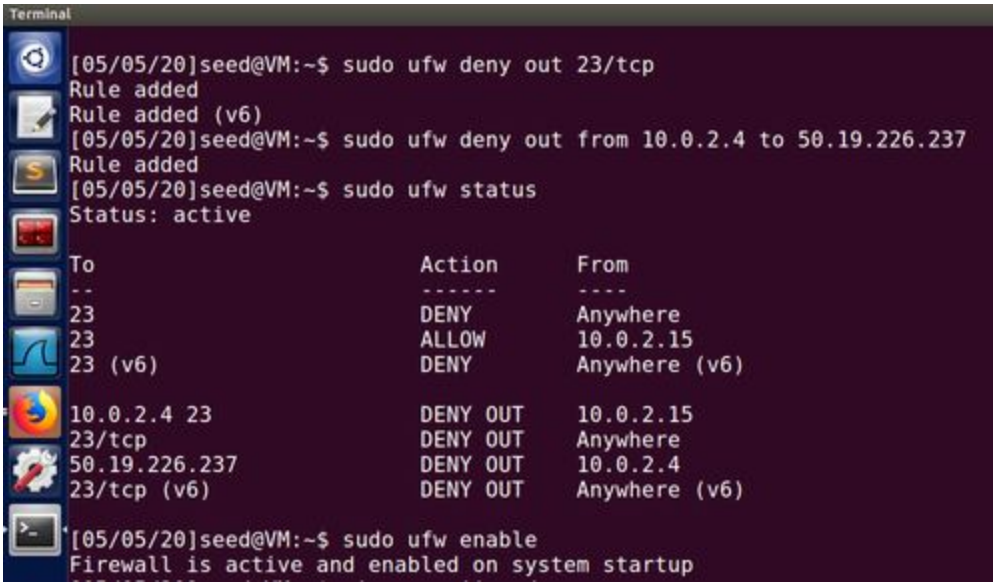
To block any telnet connections, we execute

sudo ufw deny out 23/tcp

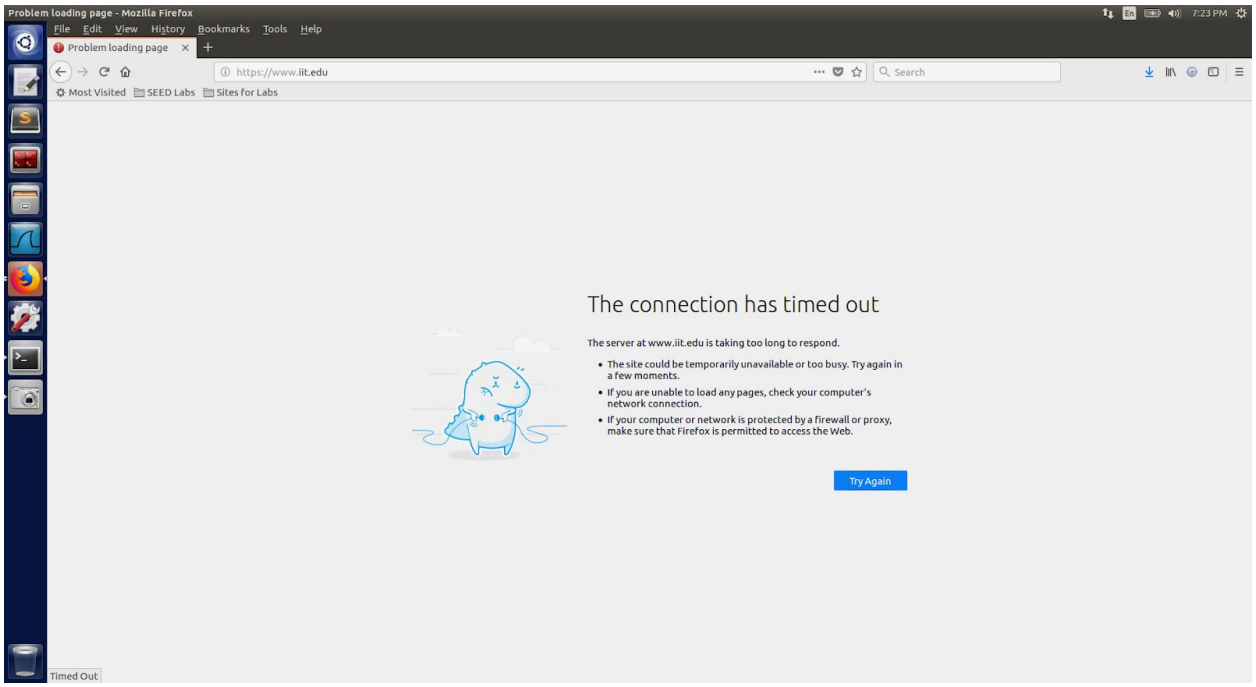
To block access to www.iit.edu running on IP 50.19.226.237, we execute

sudo ufw deny out from 10.0.2.4 to 50.19.226.237

Here is the complete setup



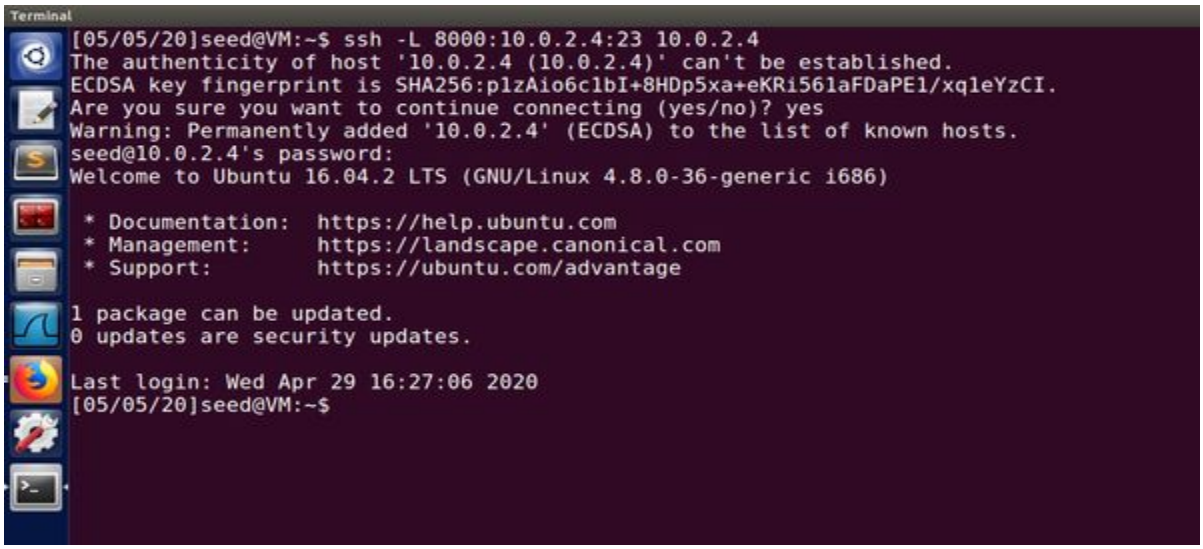
Once the access to www.iit.edu is revoked by tweaking the firewalls, when tried to access www.iit.edu from the browser, we get the following message



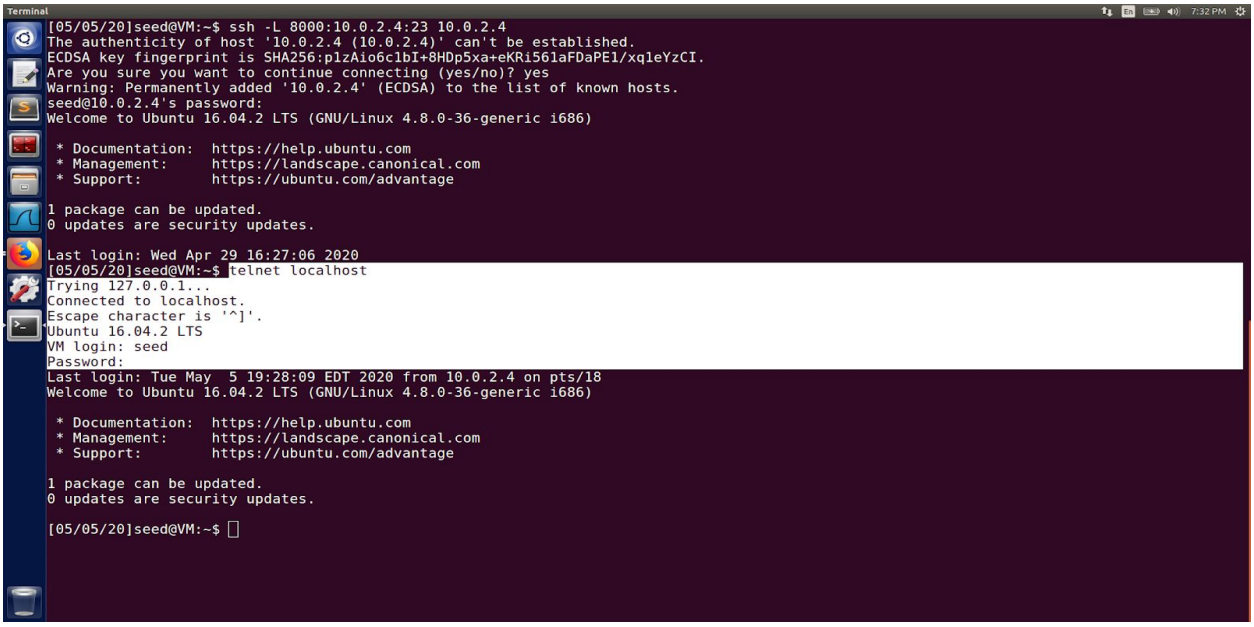
Task 3.a: Telnet to Machine B through the firewall

I now setup the ssh connection to create tunnel from 10.0.2.4 to 10.0.2.5 using the below command **ssh -L 8000:10.0.2.4:23 10.0.2.4**

On executing the above command, here is how it looks



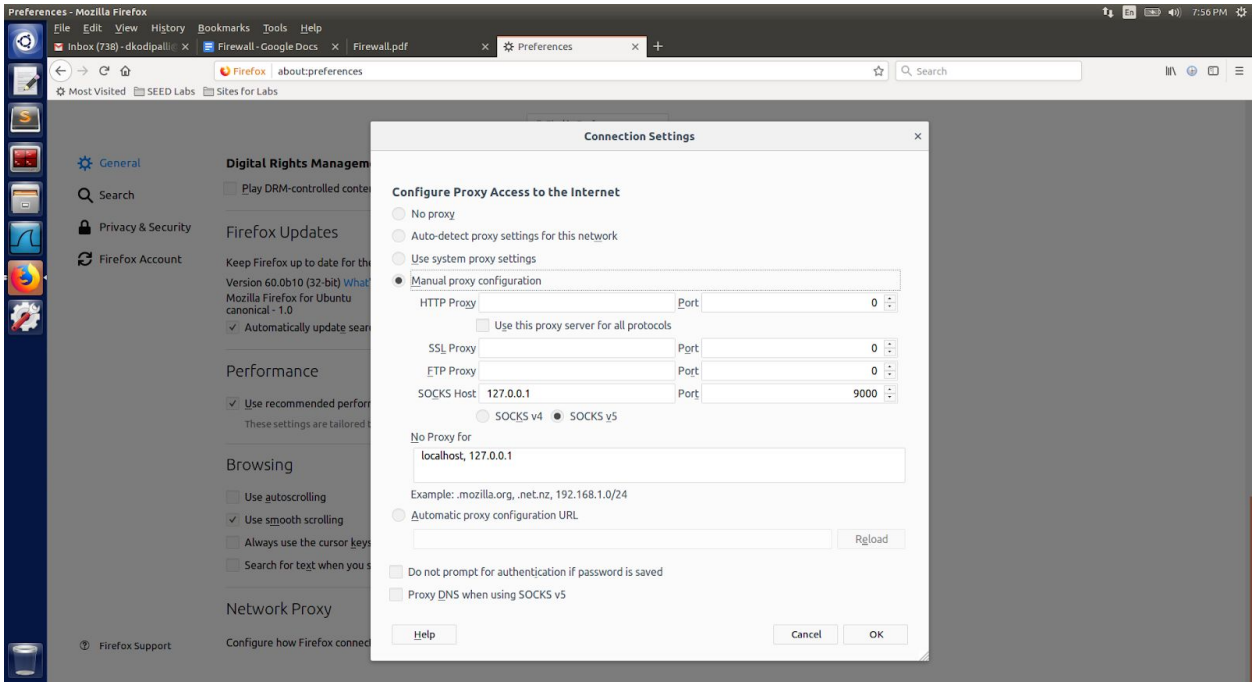
Now, we have successfully established a connection to Machine B with an IP 10.0.2.4. Previously we have disabled telnet to any machine from Machine A by tweaking the Firewall rules using ufw. Now once we have ssh'ed into Machine B, we execute telnet to localhost. Which means we're executing the telnet to Machine B from Machine A through a tunnel created in the previous step.



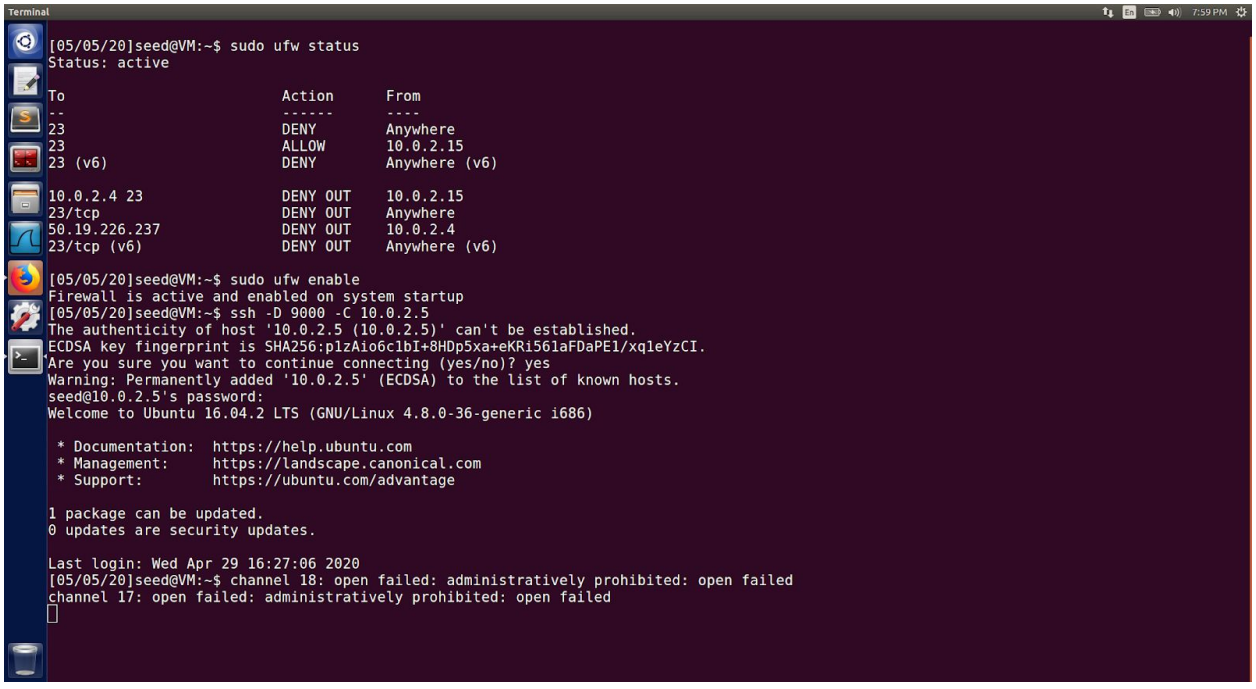
In the screenshot, the highlighted section shows how we telnet'ed to Machine B from Machine A after ssh'ing to Machine B from A. Once the ssh is done, we performed telnet to localhost which means telnetting to Machine B from A. This executed the task successfully. UFW basically blocks all telnet connection going out from Machine A, so to evade it we created an SSH connection between Machine A and Machine B. We could still perform telnet which means the Packet Filters usually don't look at the content inside the packets, they only look at packet level data like IP address and port information.

Task 3.b: Connect to Facebook/(www.iit.edu) using SSH Tunnel.

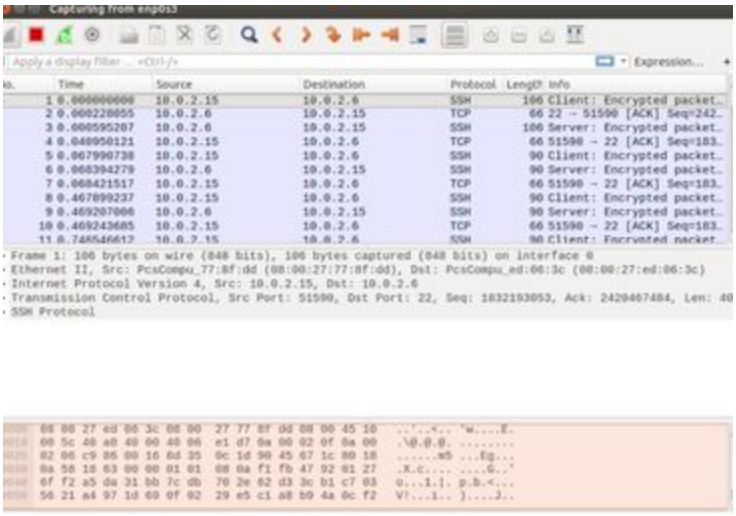
We now configure the firefox in the way shown below:



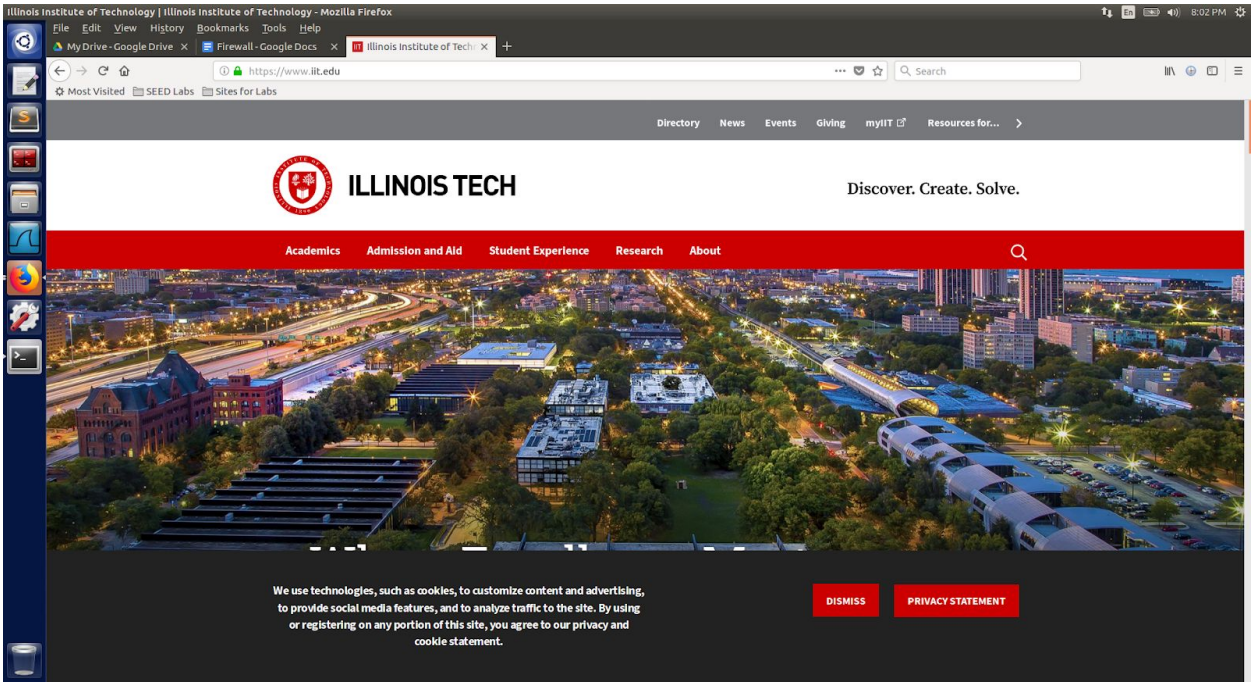
Once the proxy changes are made we now establish a tunnel. Below is how it is done



Once the ssh tunnel is established, I now try to access the www.iit.edu from the browser. This is how it looks when the packets are captured from the wireshark.



Since the Packet Filters usually don't look at the content inside the packets, they only look at packet level data like IP address and port information, we can now see the www.iit.edu page getting loaded on the browser as shows below. This makes this task execute successfully.



Task 4: Evading Ingress Filtering

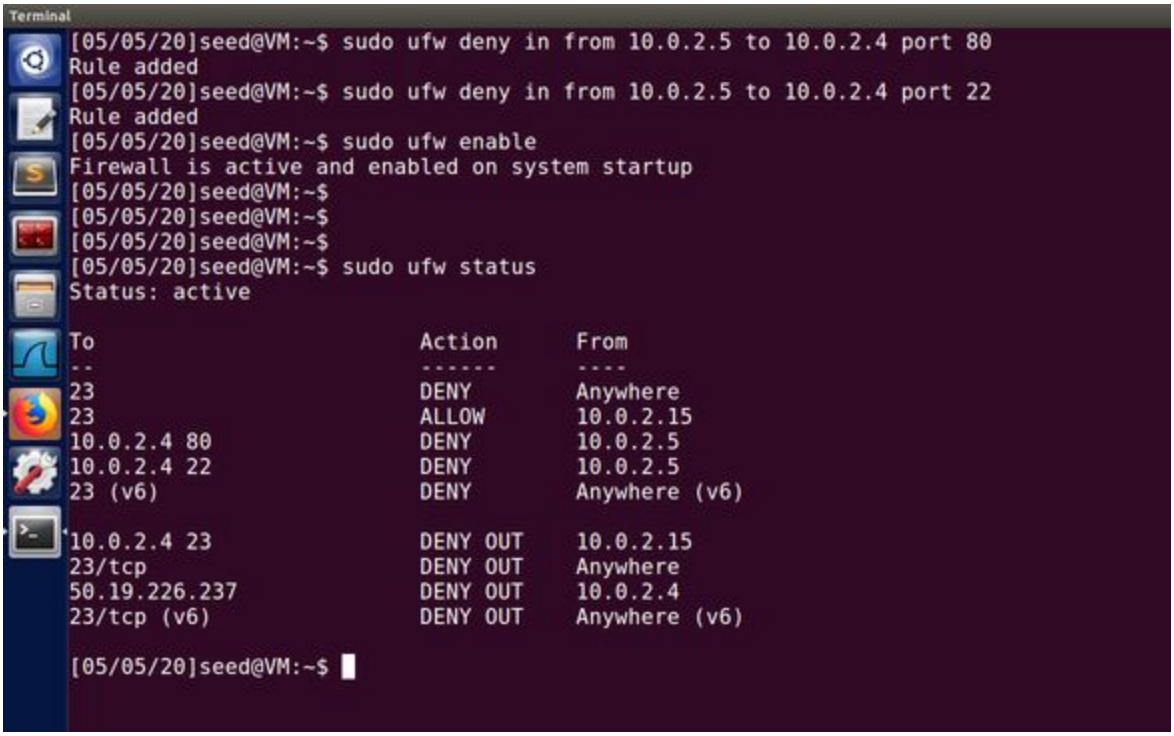
The objective of this task is to be able to access the web server on Machine A with IP 10.0.2.4 from outside. In order to make this successful, I tweaked the firewall rules using the ufw commands that are shown below.

sudo ufw deny in from 10.0.2.5 to 10.0.2.4 port 80
sudo ufw deny in from 10.0.2.5 to 10.0.2.4 port 22

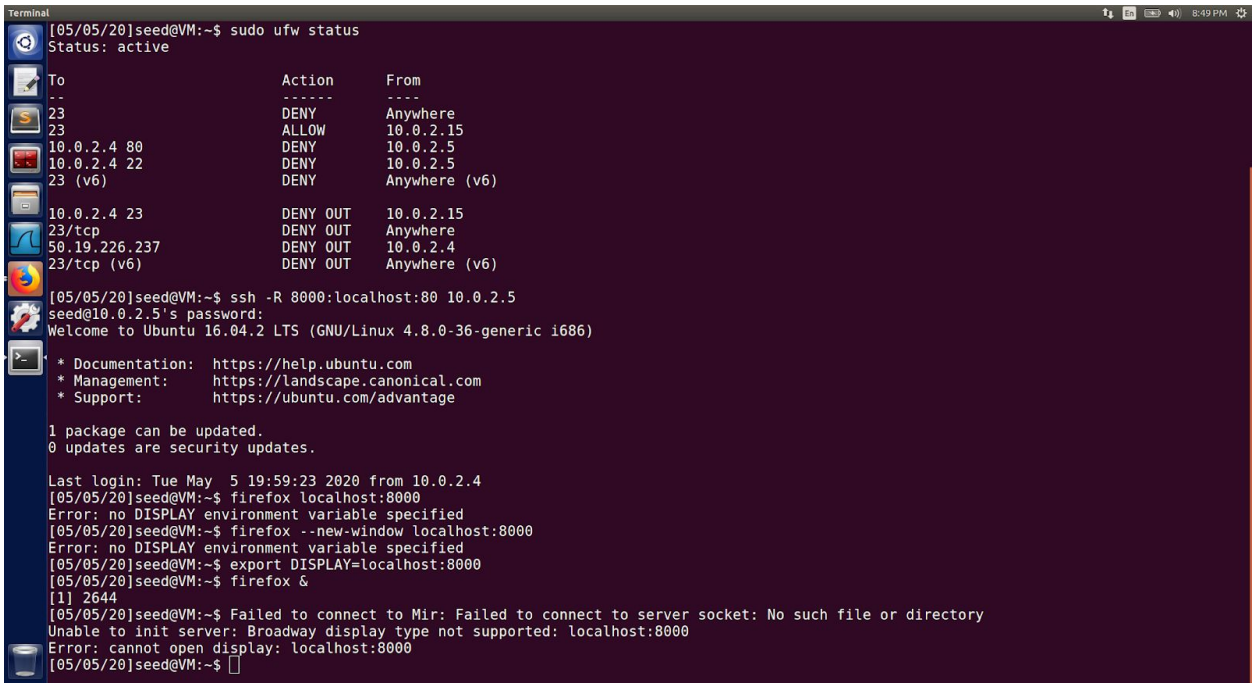
And then enable these firewall rules using

sudo ufw enable

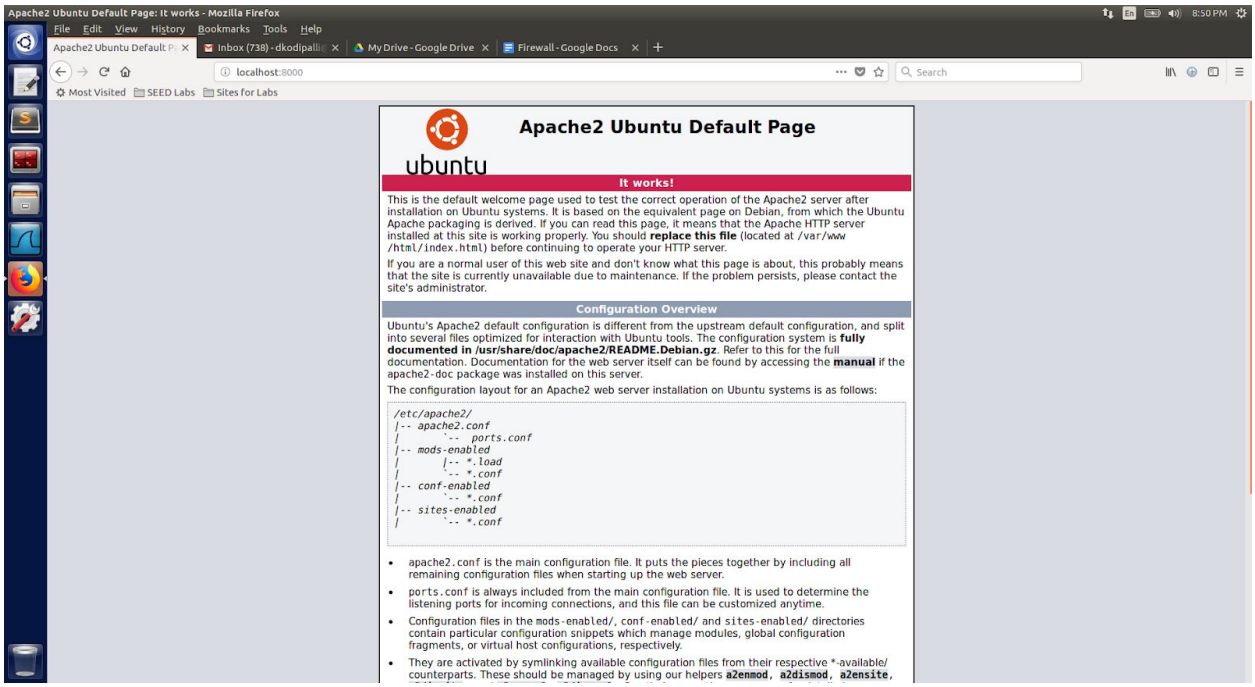
Here is how it looks once we've updated the firewall rules using ufw.



Now that we have all the rules updated, I now ssh into Machine B with an IP 10.0.2.5. And on Machine A I block Machine B from accessing its port 80 and 22 as done in the above step.



In the above screenshot we can see we have now established an ssh connection to Machine B and when we now run localhost:8000 on the browser we get the following output.



From the output above we can see we've successfully set up a reverse SSH tunnel on Machine A as we were still able to access the protected web server on A from home.