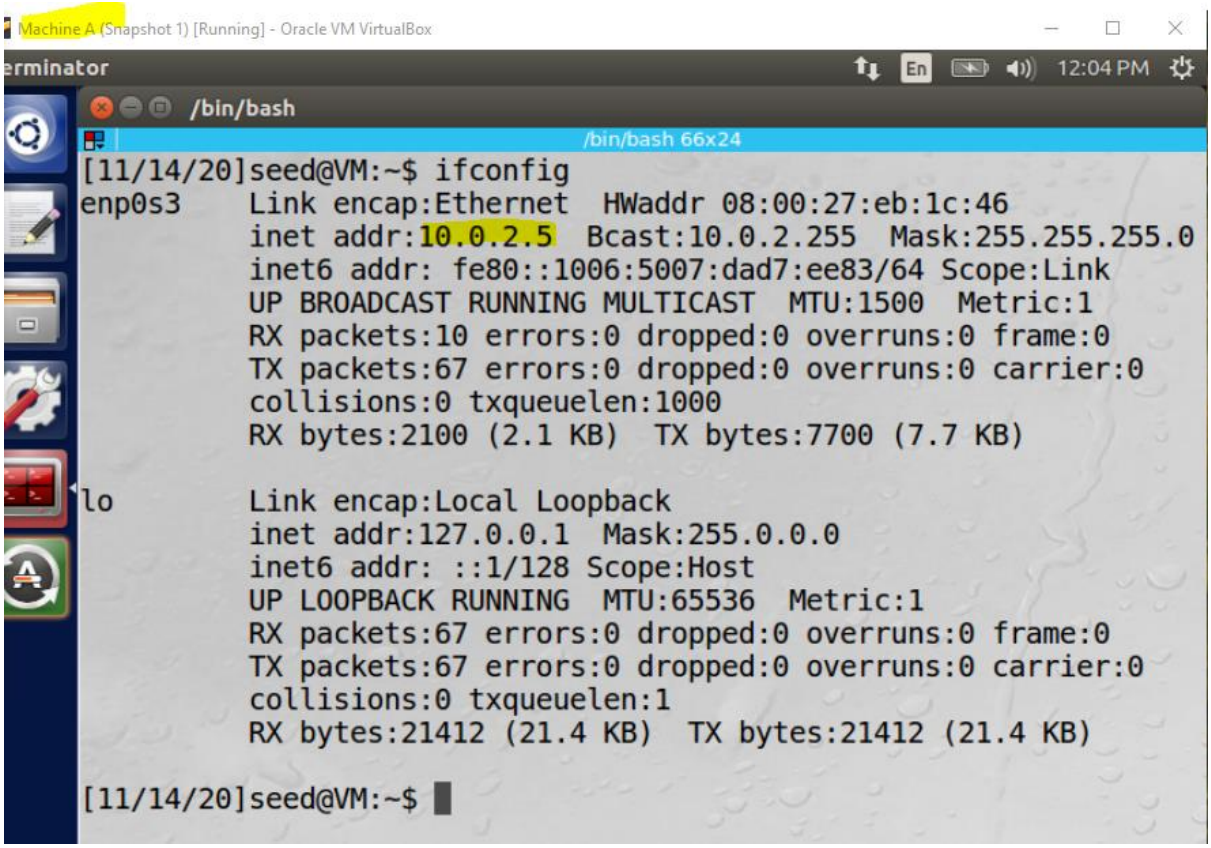
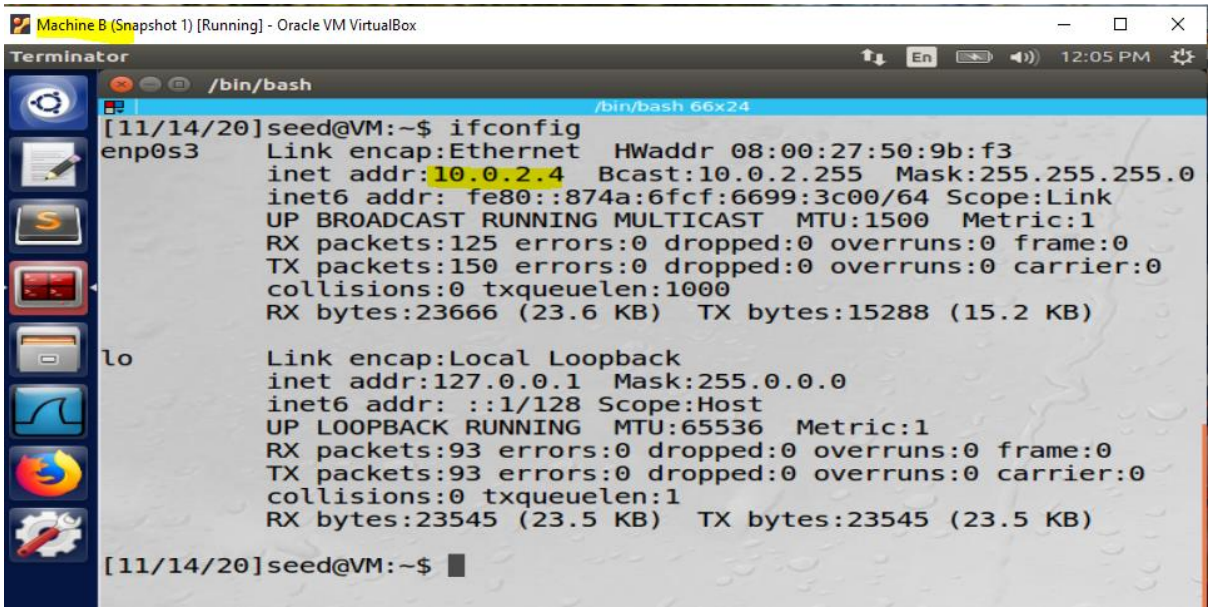
Lab: Linux Firewall Exploration

Task 1: Using Firewall

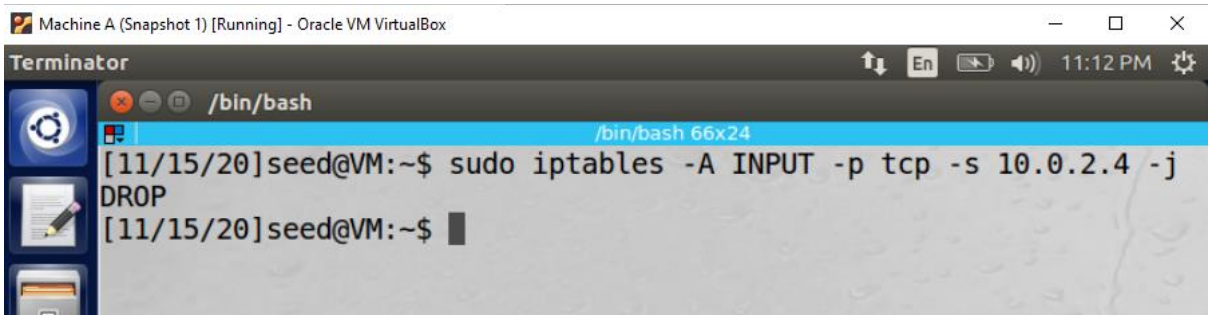
- With the help of ifconfig command, checked the IP's of both Machine A and Machine B



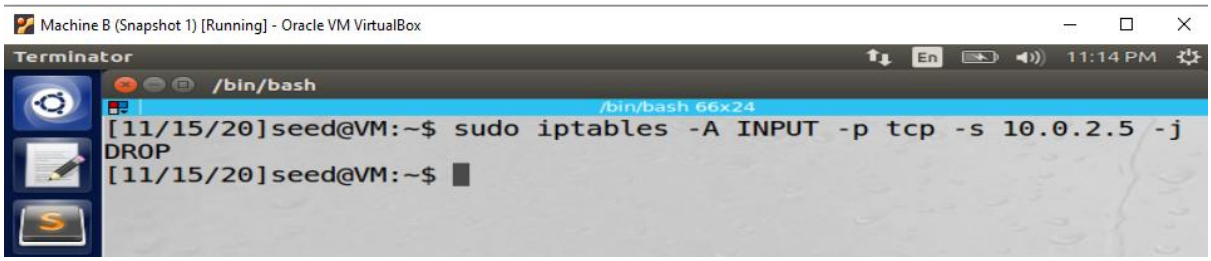


- These two machines can communicate amongst each other. Verified using telnet command. telnet 10.0.2.4

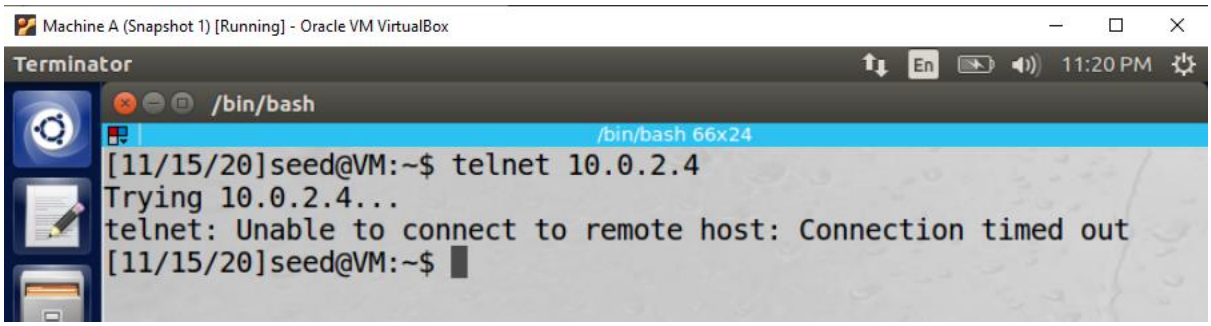
- Now to disable the communication between both machines, executed below command on Machine A
  sudo iptables -A INPUT -p tcp -s 10.0.2.4 -j DROP



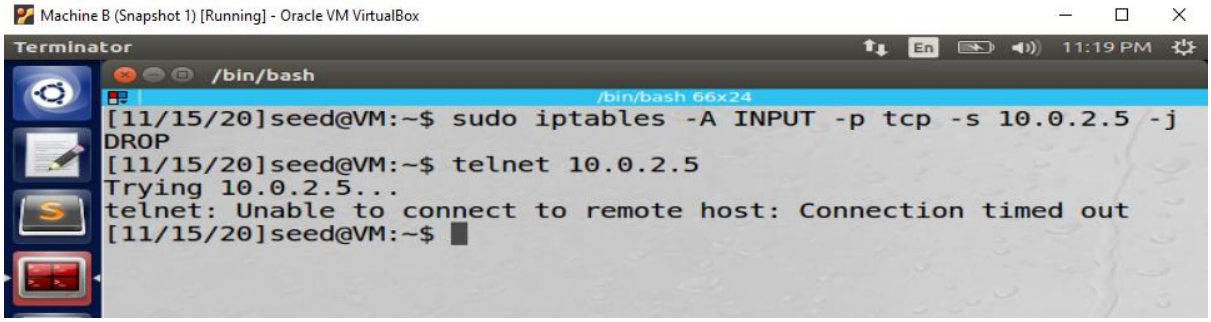
- Then execute iptables command on Machine B (10.0.2.4)
  sudo iptables -A INPUT -p tcp -s 10.0.2.5 -j DROP



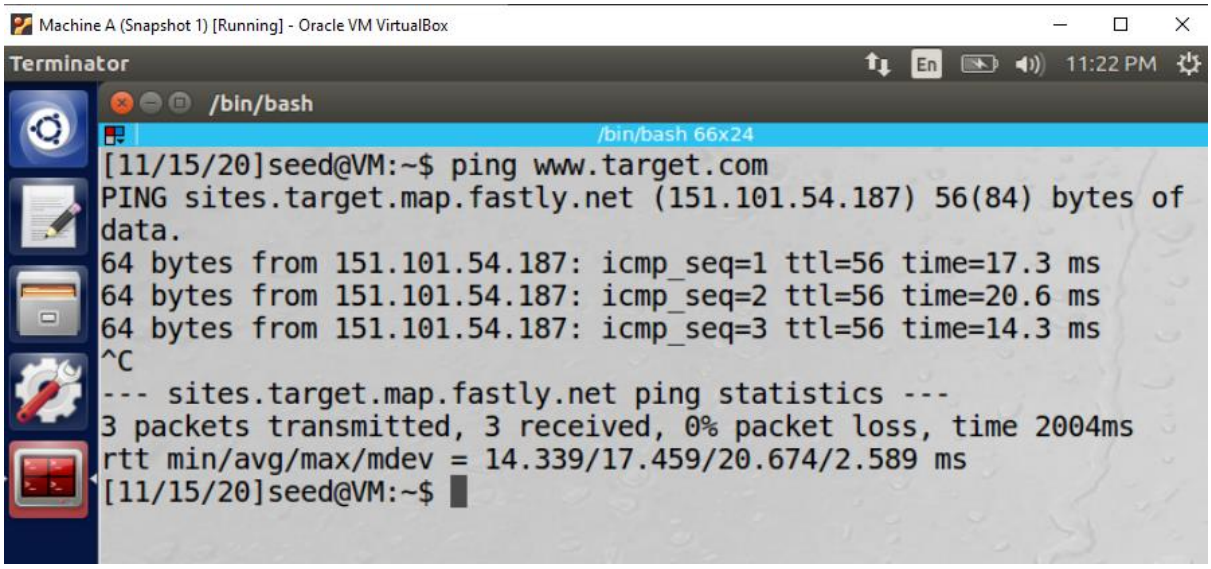
- Validated Machine A to Machine B connectivity using telnet. Hence, telnet cannot be done from Machine A to Machine B.



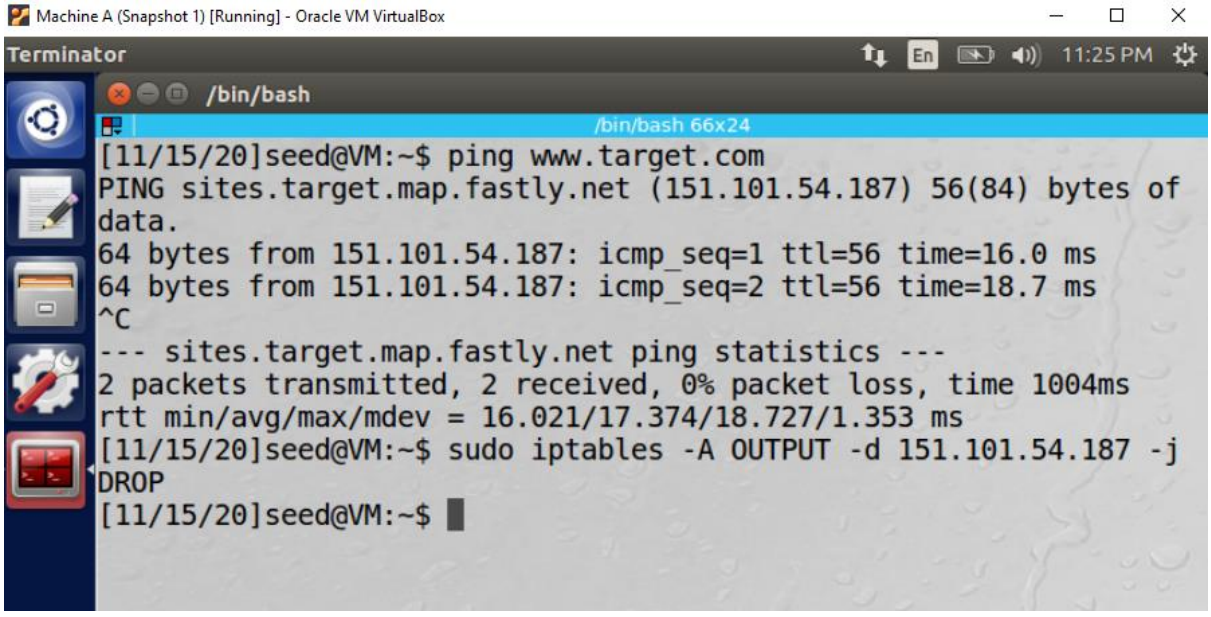
- Similarly, validating connectivity from Machine B to Machine A using telnet. Hence, getting connection timed out.



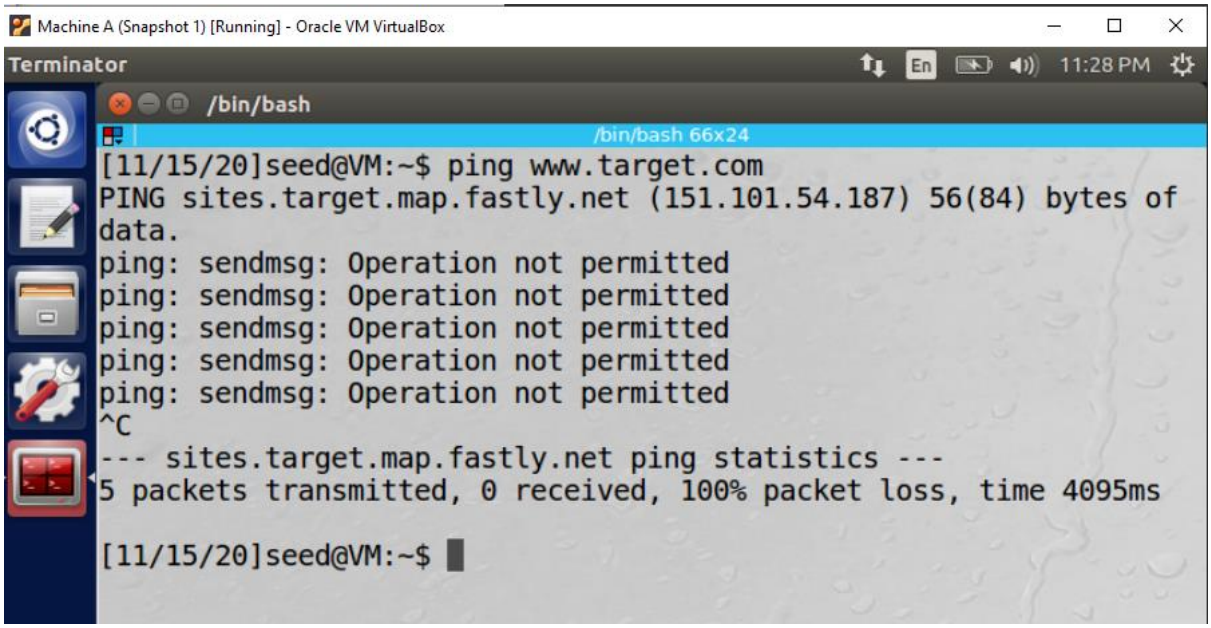
- Took www.target.com website to block in Machine A. Validated connectivity of this website using ping command on Machine A.
  Ping www.target.com



- Dropping www.target.com incoming packets from Machine A using iptables command *sudo iptables -A OUTPUT -d 151.101.54.187 -j DROP*



- Again ping www.target.com from Machine A. Now, the website is blocked on Machine A and gives 100% packet loss, as shown in below snapshot:



Task 2: Implementing a Simple Firewall

- Using LKM and Netfilter to implement the packet filtering module.

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

/* This is the structure we shall use to register our function */
static struct nf_hook_ops nfho;

/* This is the hook function itself */
unsigned int hook_func(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->saddr == in_aton("10.0.2.5") && iph->daddr == in_aton("10.0.2.6")) {
        printk("Dropping packet from %d.%d.%d.%d to %d.%d.%d.%d", ((unsigned char *)&iph->saddr)[0], ((unsigned char *)&
        iph->saddr)[1], ((unsigned char *)&iph->saddr)[2], ((unsigned char *)&iph->saddr)[3], ((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;
    }
}

/* Initialization routine */
int init_module() {
    /* Fill in our hook structure */
    nfho.hook = hook_func; /* Handler function */
    nfho.hooknum = NF_INET_PRE_ROUTING; /* First hook for IPv4 */
    nfho.pf = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho);
    return 0;
}

/* Cleanup routine */
void cleanup_module() {
    nf_unregister_hook(&nfho);
}
```

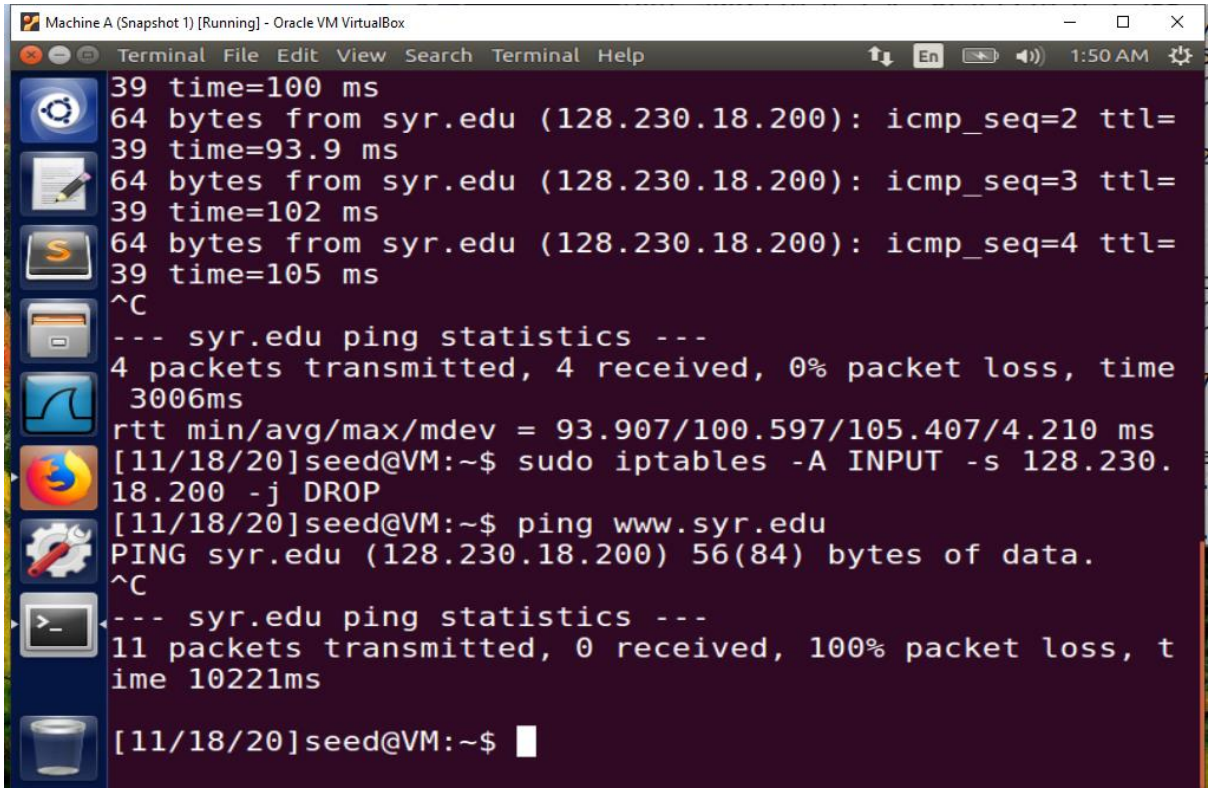
- Make file for LKM and Netfilter

```
obj-m += filter.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
```

Task 3: Evading Egress Filtering

- Validated connectivity of www.syr.edu from Machine-A and then executed command to block this website.
  sudo iptables -A INPUT -s 128.230.18.200 -j DROP
- Observed that the website www.syr.edu blocked from Machine A as shown in snapshot (100% packet loss).

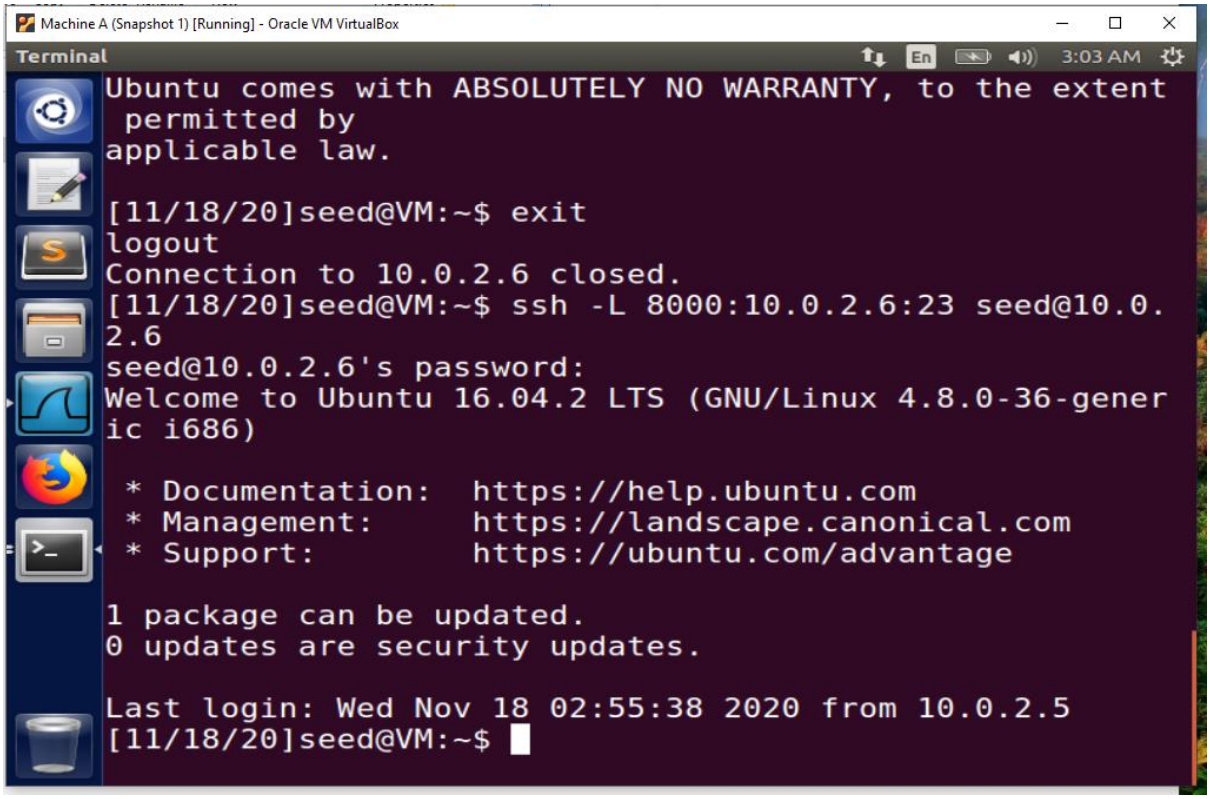
Note: Since www.facebook.com has many IP's. Therefore, I have used www.syr.edu.



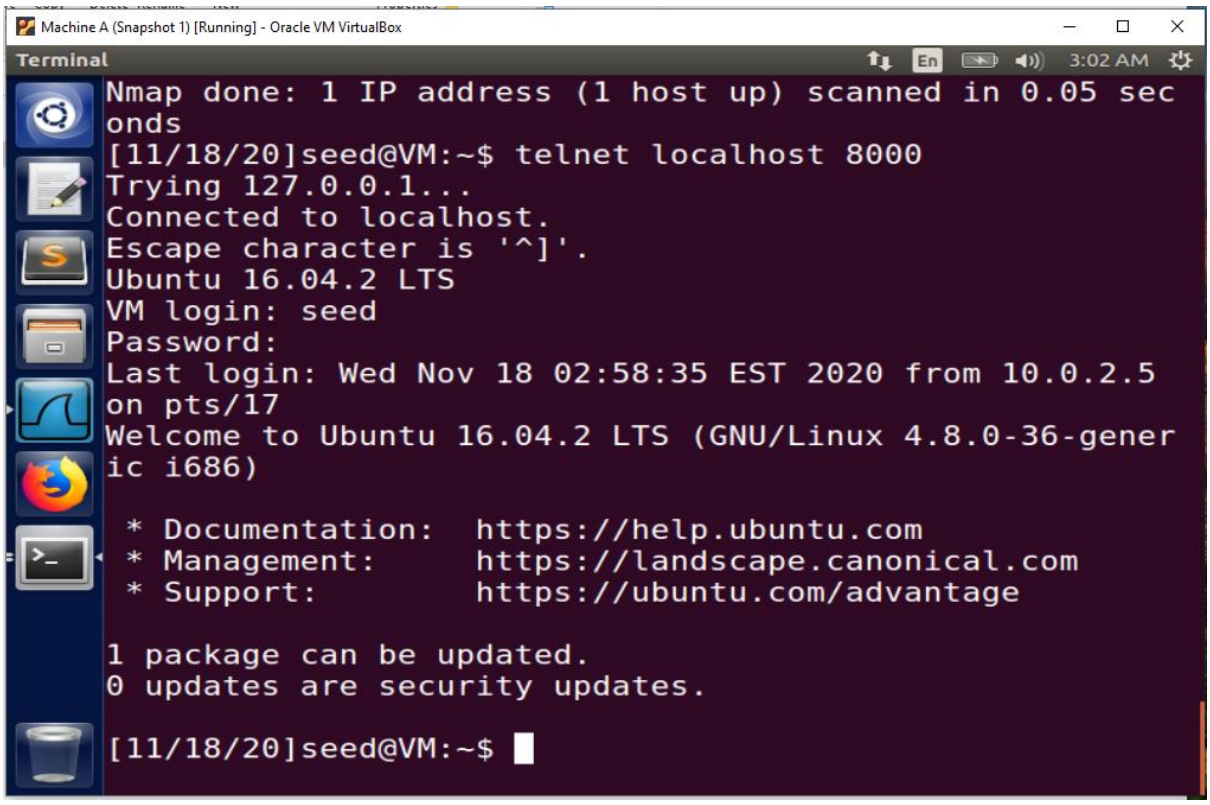
Task 3.a: Telnet to Machine B through the firewall

- Establishing SSH tunnel between the localhost (port 8000) and machine C (port 22) when packets come out of C's end, it will be forwarded to Machine C's port 23 (telnet port)
  ssh -L 8000:10.0.2.6:23 seed@10.0.2.6

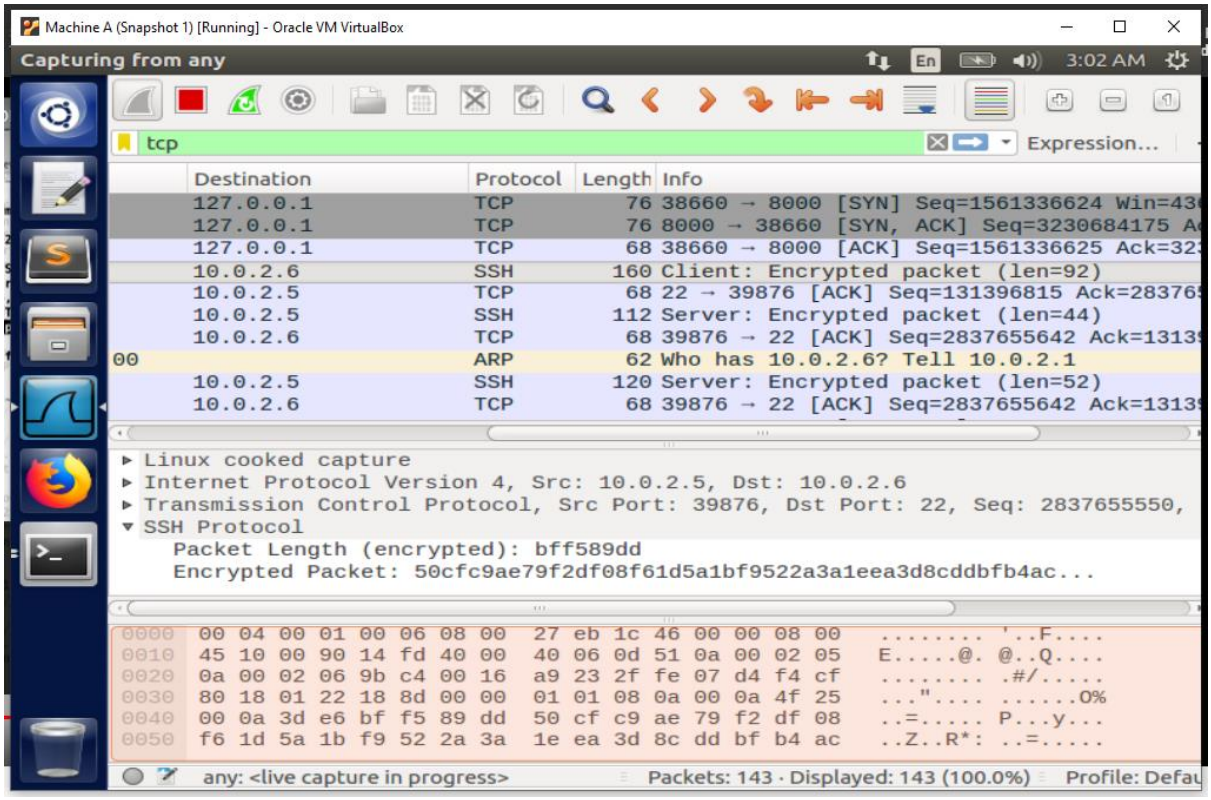
  Note: Since Machine B vm got crashed. So, I have taken Machine A and Machine C



- Now from Machine A, connecting to Machine C using tunnel
  telnet localhost 8000

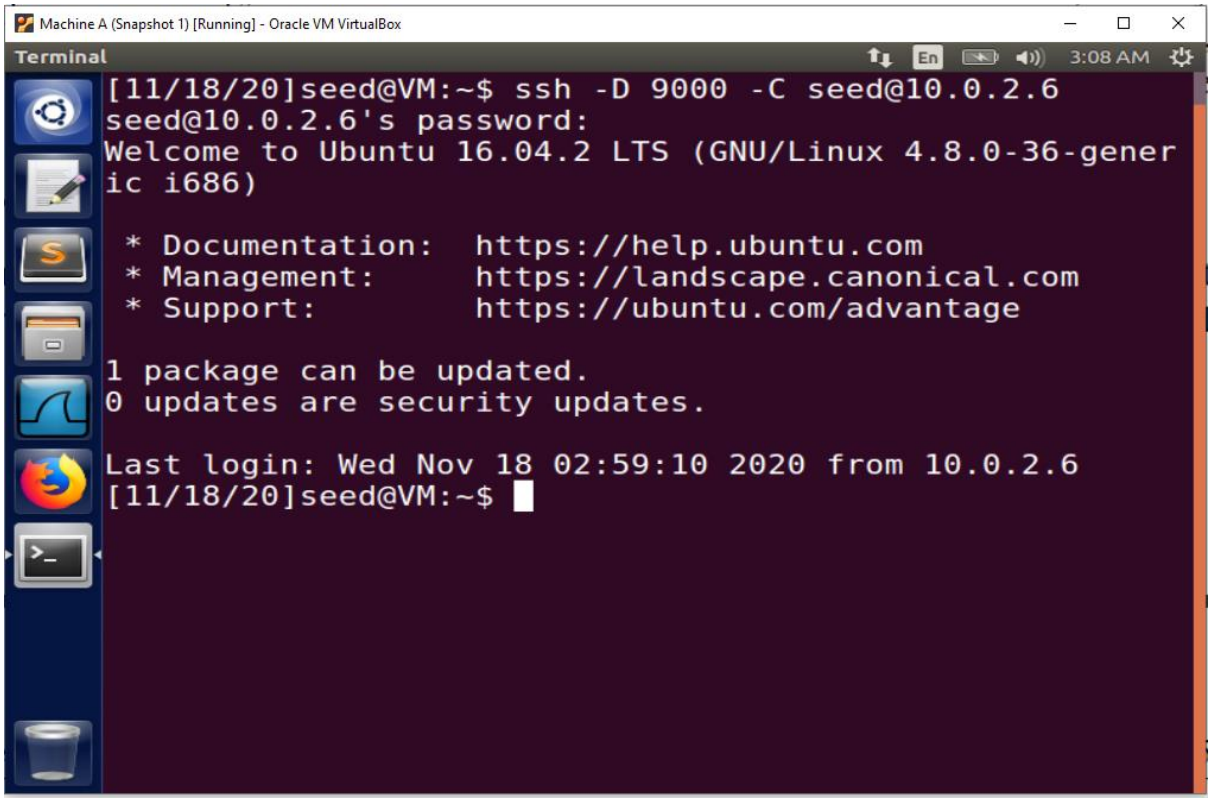


- In the wireshark of Machine A, observe SSH Encrypted Packet. Refer below snapshot:

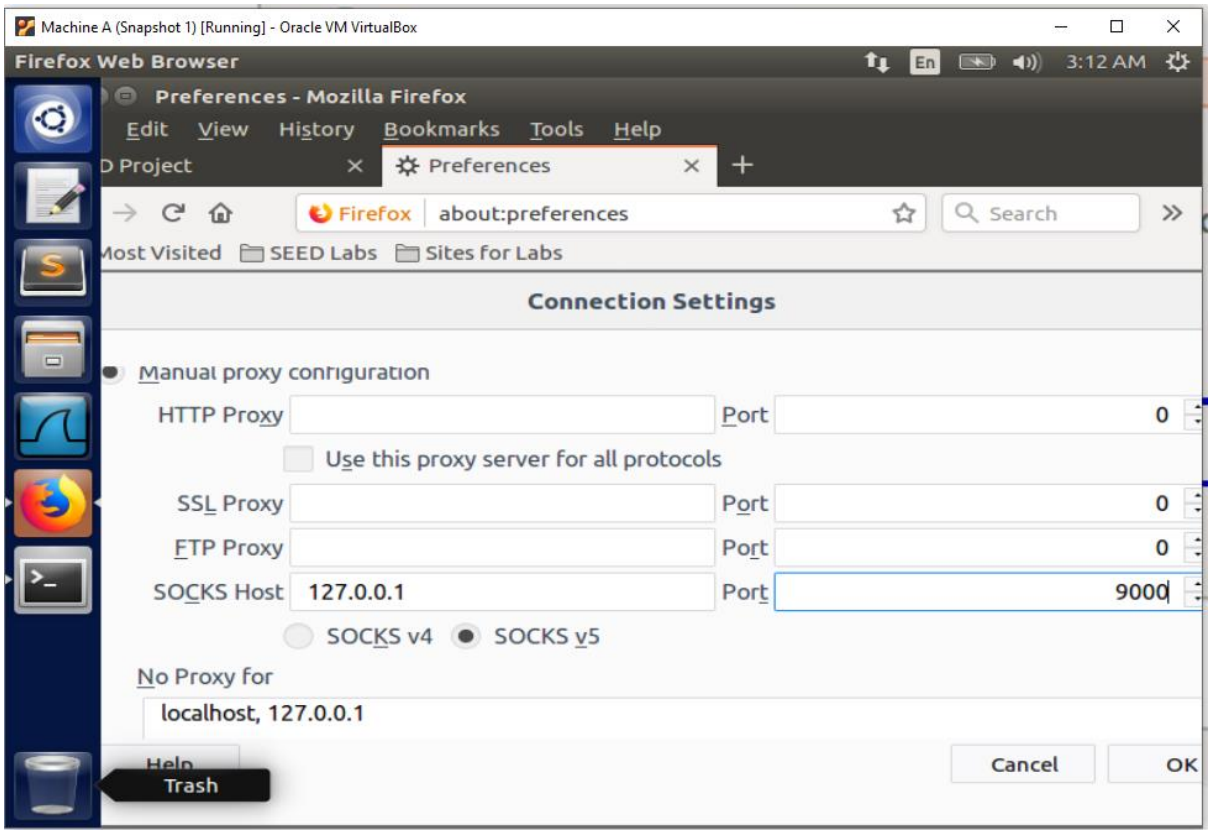


Task 3.b: Connect to Facebook using SSH Tunnel

- Using -D in the command to dynamically forward the packet based on the destination information of the packet.
  ssh -D 9000 -C seed@10.0.2.6



- To apply dynamic forwarding to port in Firefox, using SOCKS proxy which can be selected from connection setting

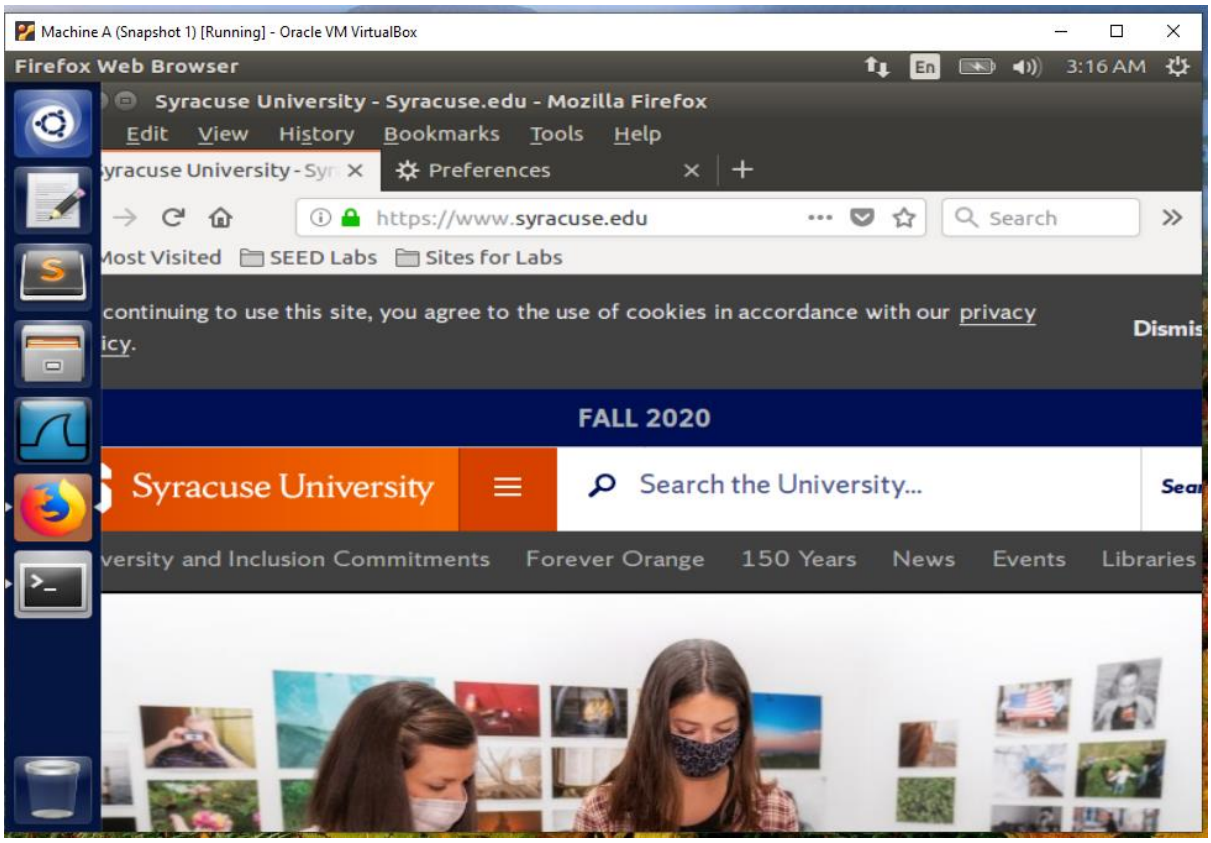


Note: Since Facebook may be using multiple Ips, I have done this task on www.syr.edu. This is the same as recommended for Task 3a

i. **Run Firefox and go visit the Facebook page. Can you see the Facebook page? Please describe your observation.**

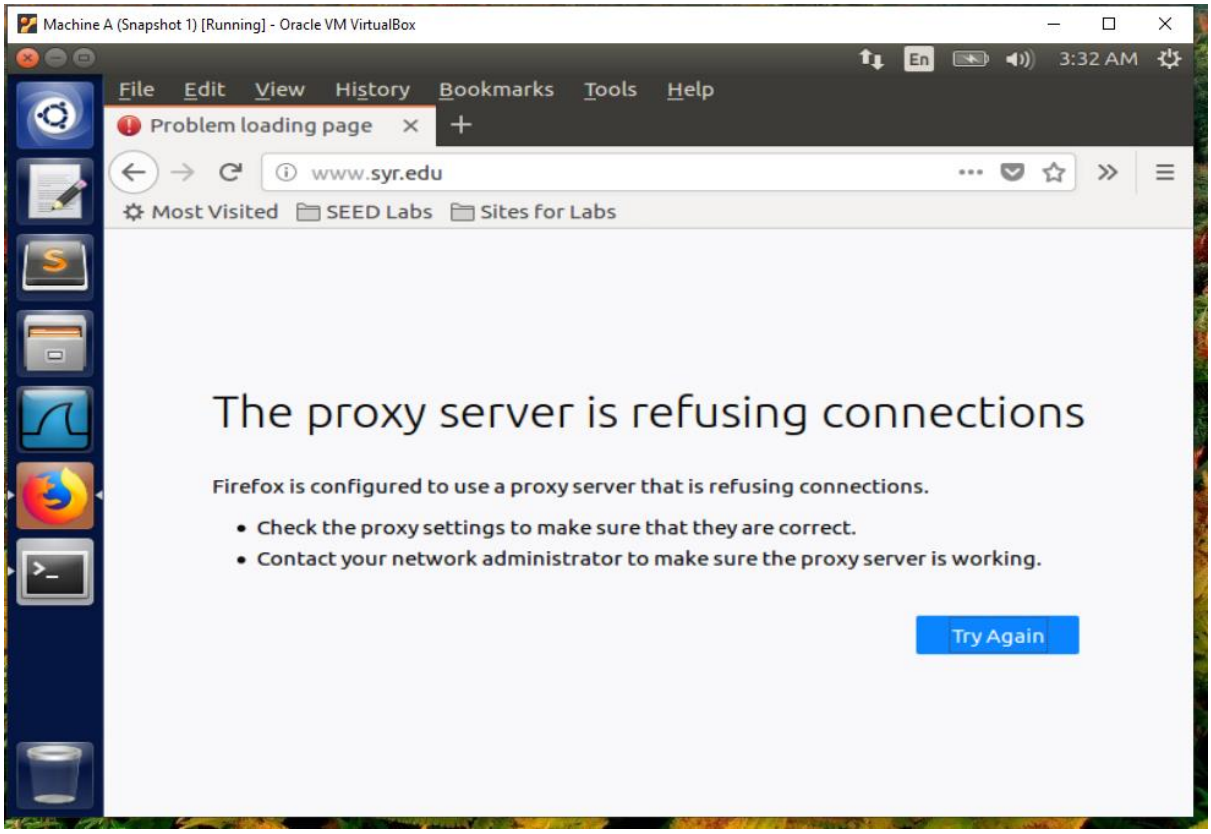
Since I am using www.syr.edu website. So, I am doing this task on this website.

Yes, I am able to see the www.syr.edu page loaded.



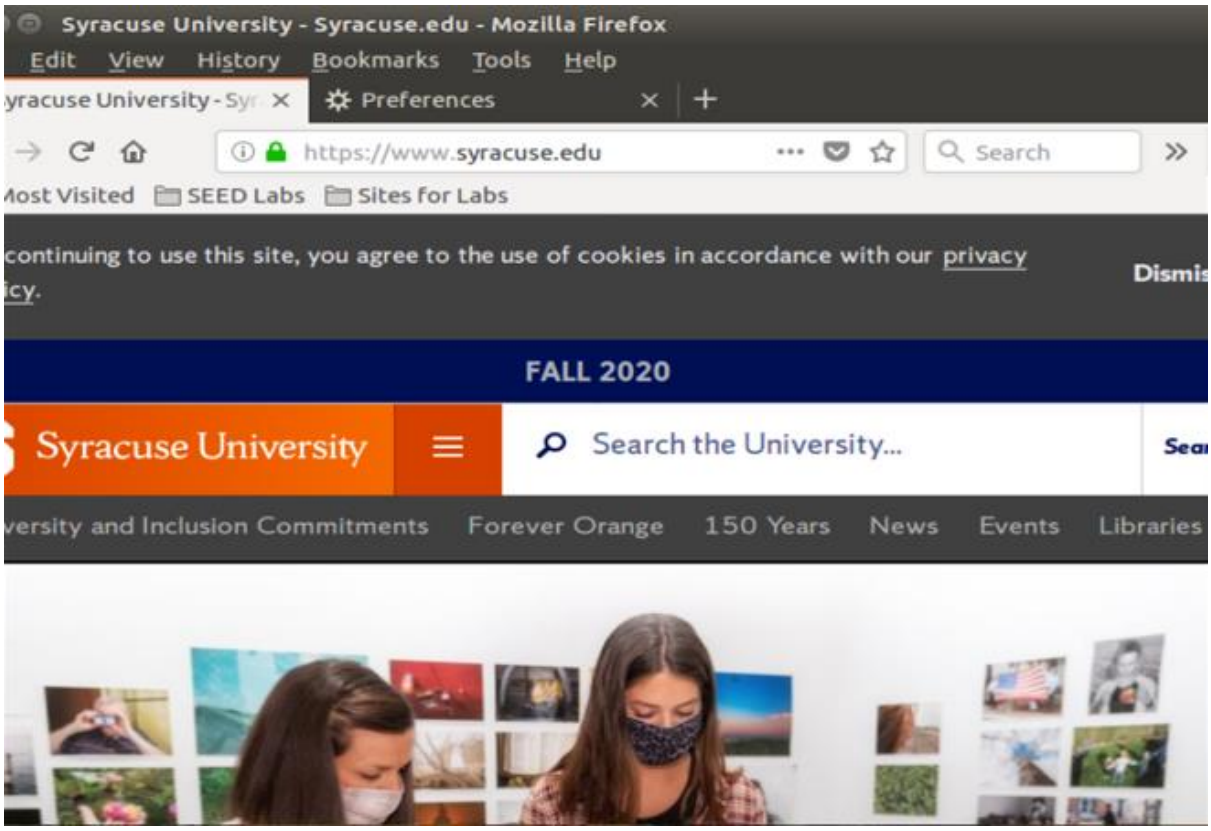
ii After you get the facebook page, break the SSH tunnel, clear the Firefox cache, and try the connection again. Please describe your observation.

After breaking the SH tunnel ad clearing the firefox browser cache and history. I observed that www.syr.edu was not able to load. It gave connection refused. Refer below snaphot.



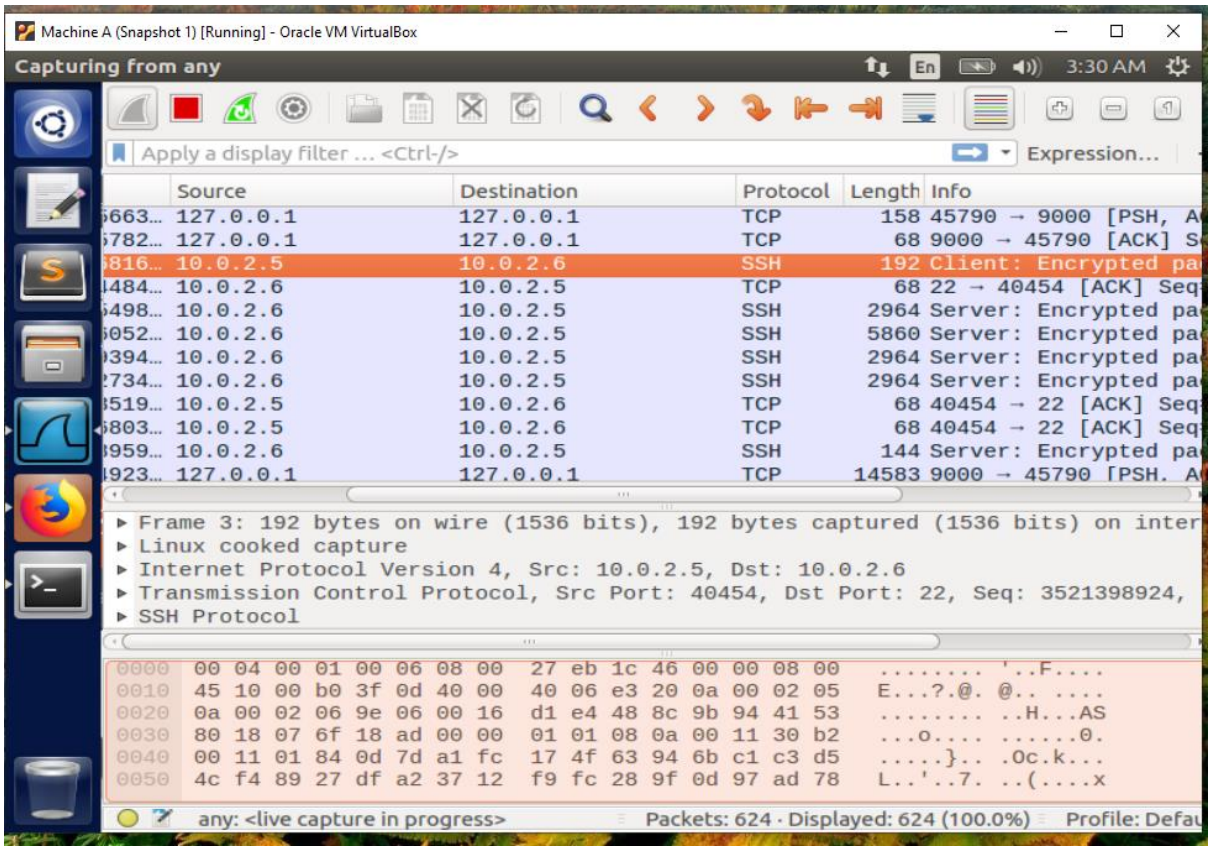
iii **Establish the SSH tunnel again and connect to Facebook. Describe your observation.**

Again, establishing the SSH tunnel connection to www.syr.edu, I was able to load the page.



iv Please explain what you have observed, especially on why the SSH tunnel can help bypass the egress filtering. You should use Wireshark to see what exactly is happening on the wire. Please describe your observations and explain them using the packets that you have captured.

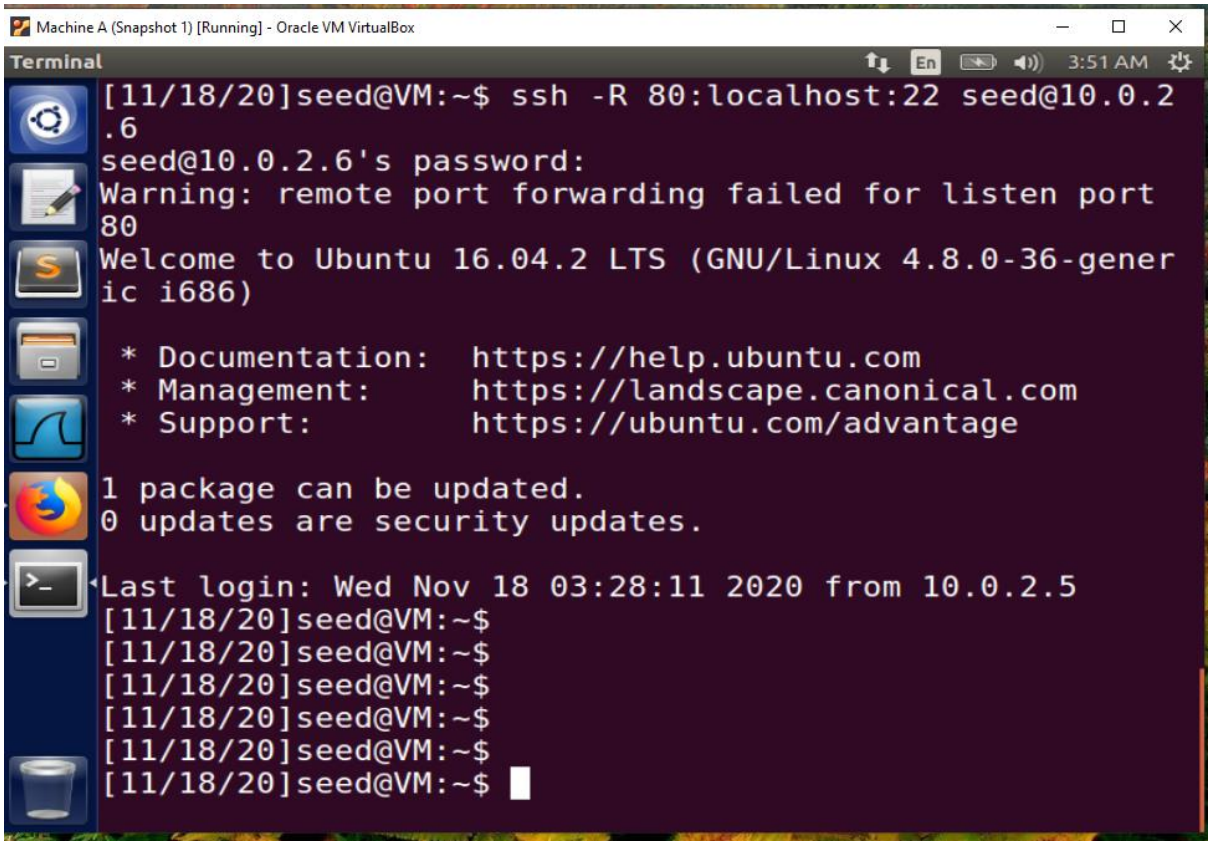
I observed whenever the SSH tunnel connection establish, then I was able to connect to www.syr.edu and when the SSH tunnel connection is lost, then I was not able to connect to this website. The packets sent from Machine A (source) to Machine B (destination) were encrypted as they were sent using SSH protocol. Refer below wireshark screenshot.



Task 4: Evading Ingress Filtering

Applying reverse SSH on Machine A where blocking Machine B from accessing its port 80(web server) and 22 (SSH server). Executed below reverse SSH command on Machine A.

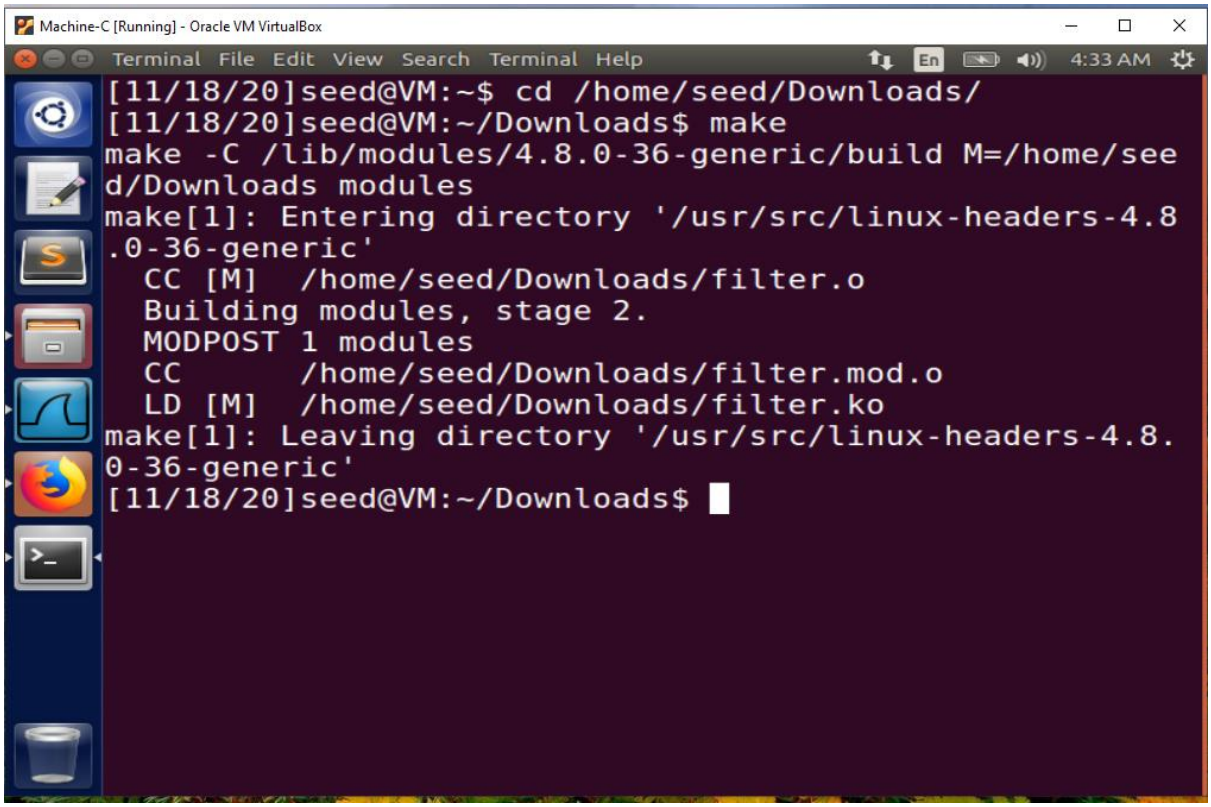
ssh -R 80:localhost:22 seed@10.0.2.6



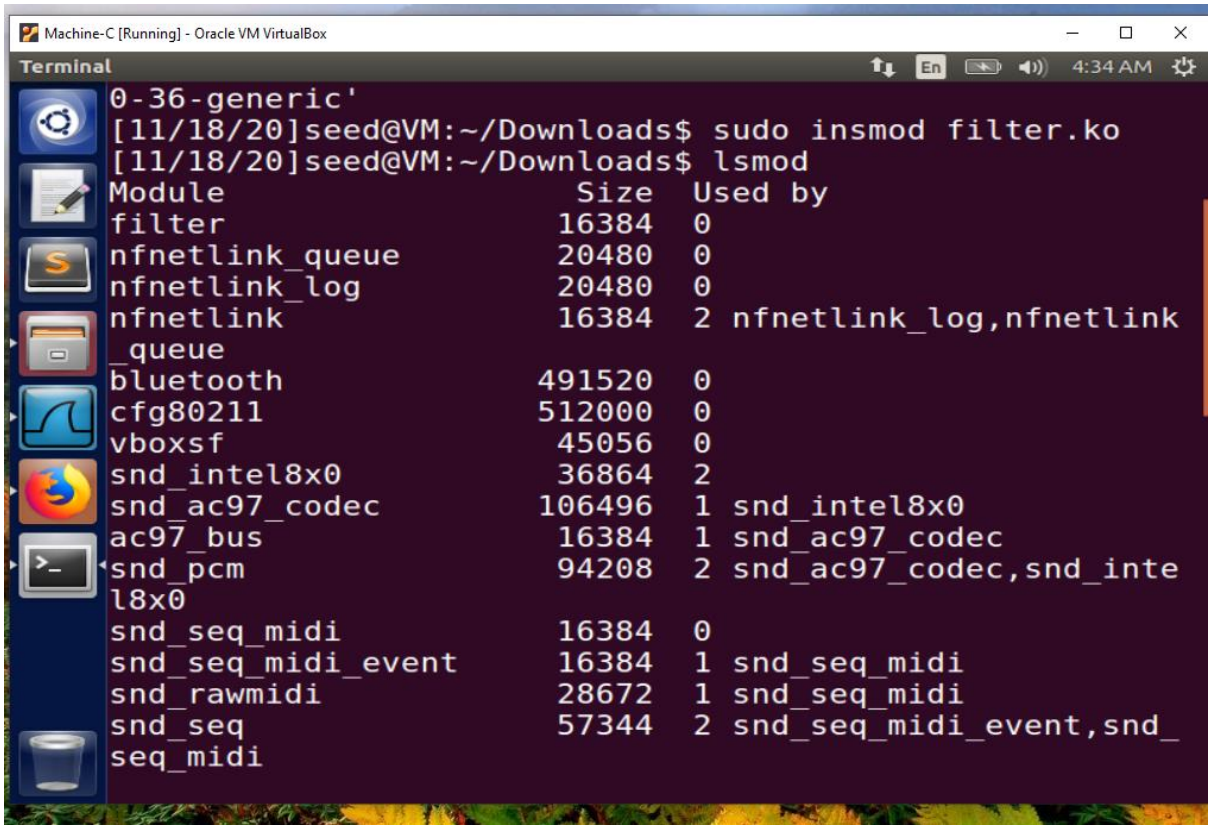
3.1 Loadable Kernel Module

- Loading make file as used in Task-2

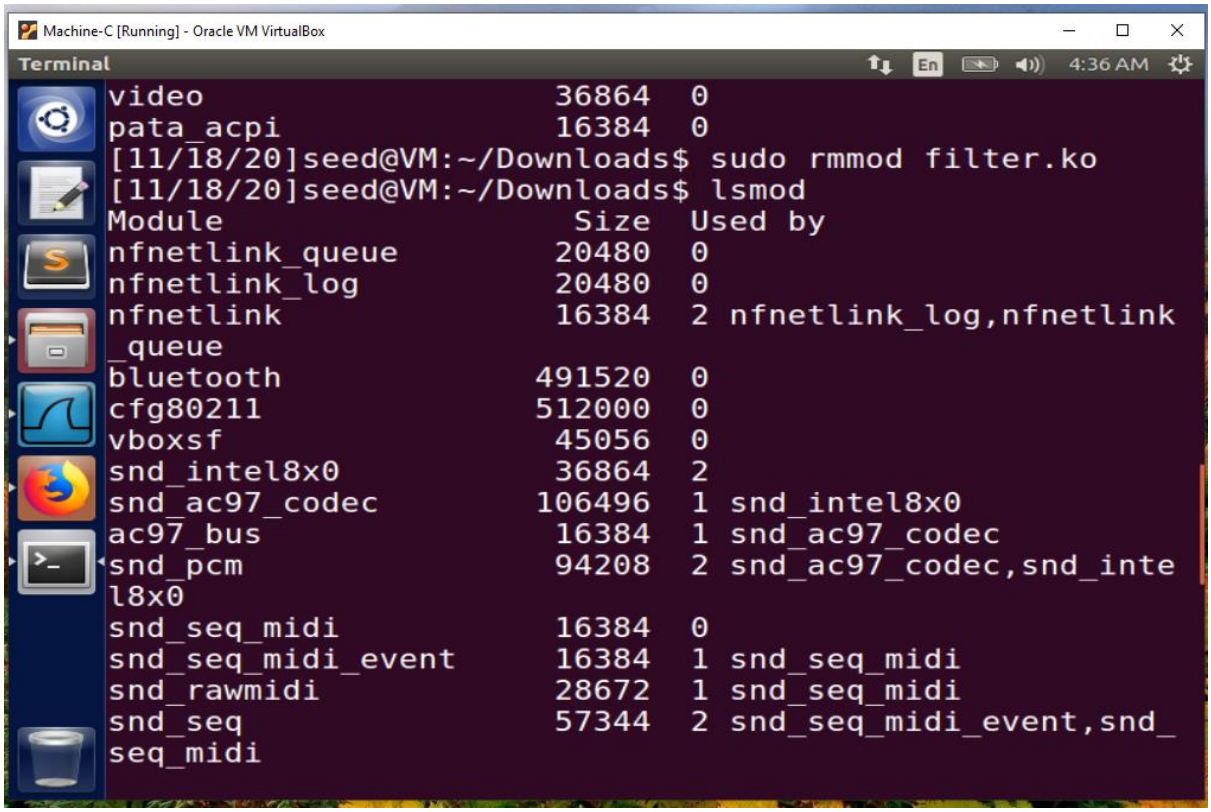
```
obj-m += filter.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```



- After the module has been built using make file, load the module with the help of below command.
  sudo insmod filter.ko
- After listing the module, I was able to see filter using **lsmod** command.



- Remove filter.ko module using command *sudo rmmod filter.ko*

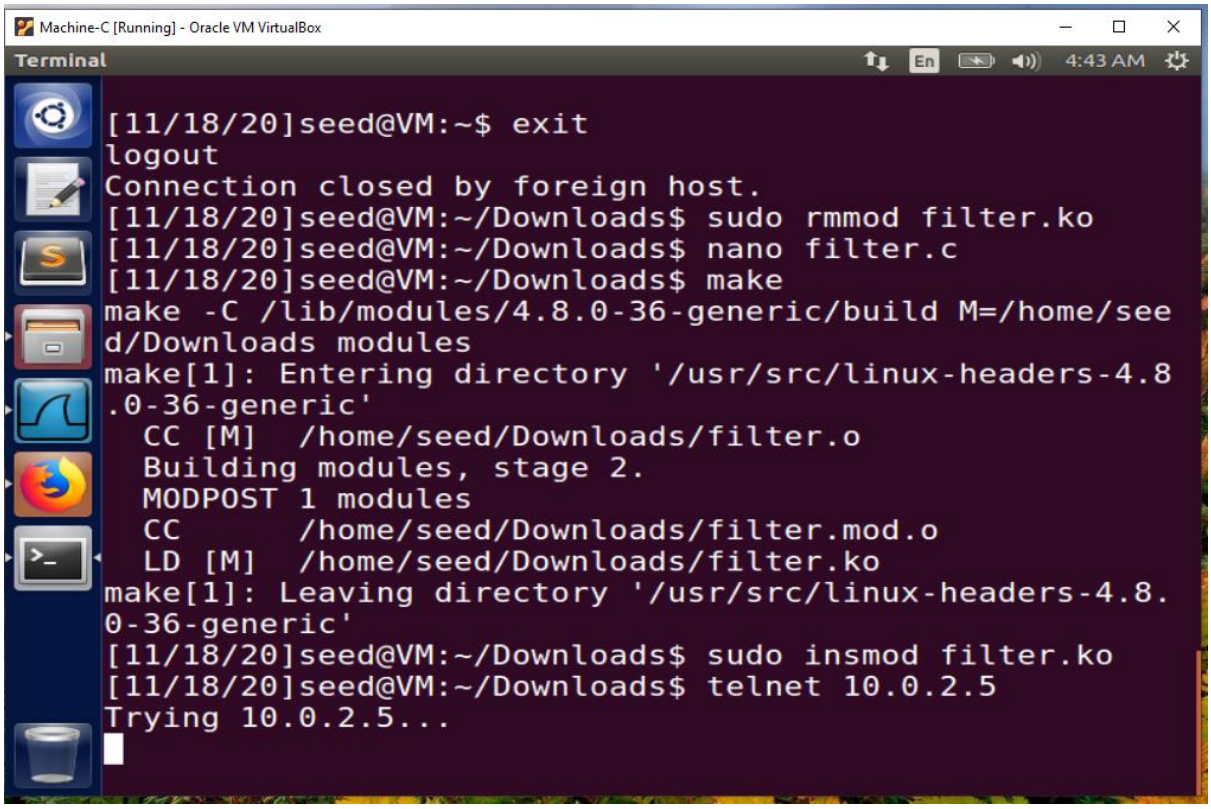


3.2 A Simple Program that Uses Netfilter

- With the help of program used in Task.2

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

/* This is the structure we shall use to register our function */
static struct nf_hook_ops nfho;

/* This is the hook function itself */
unsigned int hook_func(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->saddr == in_aton("10.0.2.5") && iph->daddr == in_aton("10.0.2.6")) {
        printk("Dropping packet from %d.%d.%d.%d to %d.%d.%d.%d", ((unsigned char *)&iph->saddr)[0], ((unsigned char *)&
        iph->saddr)[1], ((unsigned char *)&iph->saddr)[2], ((unsigned char *)&iph->saddr)[3], ((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;
    }
}

/* Initialization routine */
int init_module() {
    /* Fill in our hook structure */
    nfho.hook = hook_func; /* Handler function */
    nfho.hooknum = NF_INET_PRE_ROUTING; /* First hook for IPv4 */
    nfho.pf = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST; /* Make our function first */
    nf_register_hook(&nfho);
    return 0;
}

/* Cleanup routine */
void cleanup_module() {
    nf_unregister_hook(&nfho);
}
```

- Observed that packets were dropped when sent from source (Machine C) to destination (Machine A)



- Refer below logs showing dropping packet from 10.0.2.5 to 10.0.2.6