

Author: Wong Tin Kit  
Student ID: 1003331

## Task 1: Using Firewall

Machine	Name	IP
A	SEEDUbuntu	10.0.2.6
B	SEEDUbuntu Clone	10.0.2.10

I will be using the above configurations and implement the firewall configurations on Machine A for this task.

First I ran ***cleanup.sh*** (Fig. 1.0) which first sets up the default policy (ACCEPT) on all tables (filter, forward, output). I then proceeded to flush all existing configurations. We can run this script by typing the command ***sudo bash cleanup.sh***.

Next, I ran ***task1.sh*** which contains the commands to reject ingress and egress telnet packets as well as drop all egress packets destined to [www.facebook.com](http://www.facebook.com). We can run this script by typing the command ***sudo bash task1.sh***. We can see the updated firewall rules with ***sudo iptables -L*** as illustrated in Fig. 1.2.

The first 2 commands are for bidirectional telnet packets:

- ***iptables -A INPUT -p tcp --dport telnet -j REJECT***
- ***iptables -A OUTPUT -p tcp --dport telnet -j REJECT***

To drop telnet packets bidirectionally on Machine A, we have to append the rule to both the ***INPUT*** and ***OUTPUT*** chains. The rest of the commands are the same for both chains. I could have specified the action on telnet traffic to be ***DROP*** to achieve the same results. The difference is that the user will have to wait for the telnet connection timeout message which is much longer as compared to receiving the Connection refused message when specifying the action to be ***REJECT***. In Fig. 1.3, we see on Machine A (left VM) and Machine B (right VM) being unable to establish a new telnet connection with each other.

To illustrate that this concept will work with websites that have multiple ip addresses. I will modify the search string to be ***google.com***. The command now is:

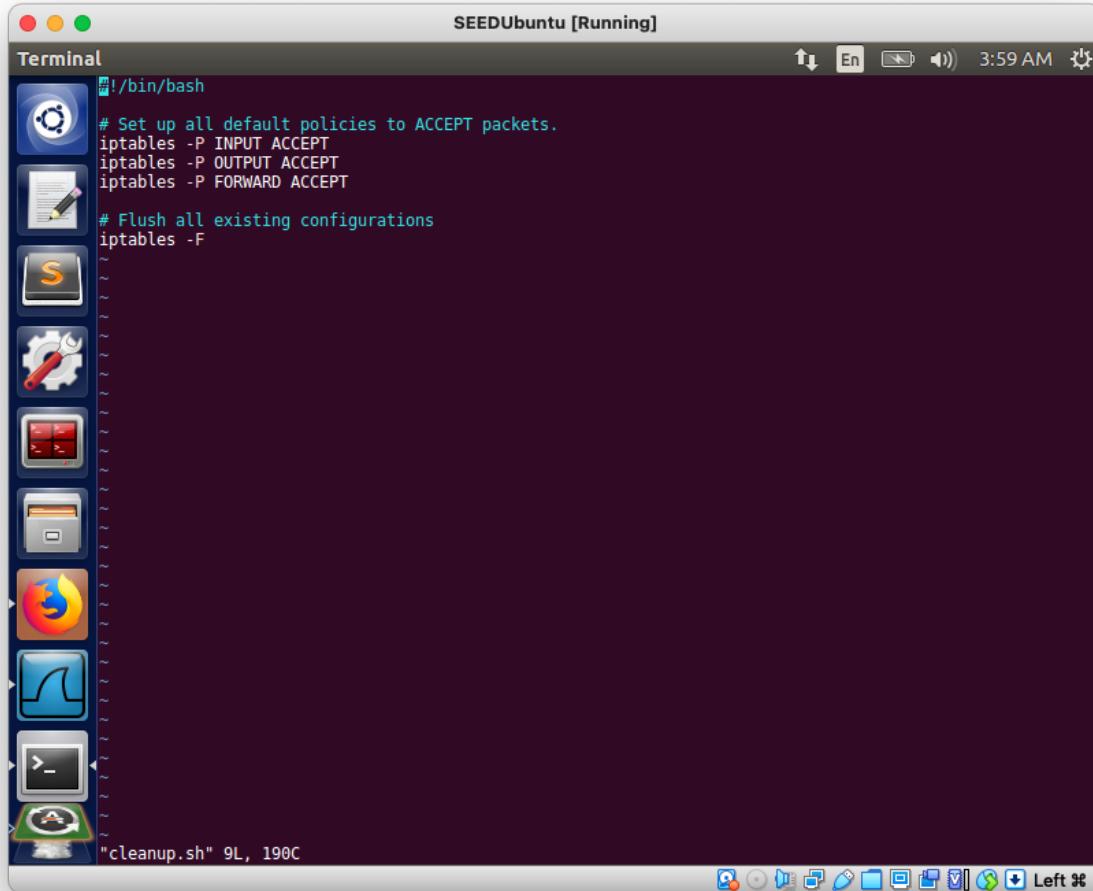
The 3rd command is to prevent Machine A from visiting an external web site:

- ***iptables -A OUTPUT -p tcp -m string --string "facebook.com" --algo kmp -j DROP***
- ***iptables -A OUTPUT -p tcp -m string --string "google.com" --algo kmp -j DROP***

I chose [www.facebook.com](http://www.facebook.com) as the restricted website. We tell iptables to look for the string ***facebook.com*** with ***-m string --string "facebook.com" --algo kmp*** using Knuth-Morris-Pratt matching algorithm.

Similar to visiting [www.facebook.com](http://www.facebook.com), we are unable to visit [www.google.com](http://www.google.com). We see this behavior in Fig. 1.5.

Another way would be to dynamically lookup the website's multiple ip addresses and add a new rule for each ip address. This can be done within ***task1.sh***.



The screenshot shows a terminal window titled "SEEDUbuntu [Running]" with the status bar indicating "3:59 AM". The terminal content is as follows:

```
#!/bin/bash
# Set up all default policies to ACCEPT packets.
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
# Flush all existing configurations
iptables -F
```

The terminal window has a dark background and a vertical dock on the left side containing icons for various applications like a terminal, file manager, and browser.

Fig. 1.0 ***cleanup.sh***

Fig. 1.1 *task1.sh*

The screenshot shows a terminal window titled "SEEDUbuntu [Running]". The terminal displays the output of the command `sudo iptables -L`. The output shows three chains: INPUT, FORWARD, and OUTPUT. The INPUT chain has a REJECT rule for TCP port 23 (telnet) from anywhere. The FORWARD chain has a REJECT rule for TCP port 23 from anywhere to anywhere. The OUTPUT chain has two DROP rules for TCP port 23 from anywhere to "facebook.com" and "google.com". The timestamp in the terminal is [03/09/21].

```
[03/09/21]seed@VM:~/.../lab5$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
REJECT    tcp  --  anywhere             anywhere            tcp dpt:telnet reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
REJECT    tcp  --  anywhere             anywhere            tcp dpt:telnet reject-with icmp-port-unreachable

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DROP      tcp  --  anywhere             anywhere            STRING match "facebook.com" ALGO name kmp TO 65535
DROP      tcp  --  anywhere             anywhere            STRING match "google.com" ALGO name kmp TO 65535

[03/09/21]seed@VM:~/.../lab5$
```

Fig. 1.2 *iptables* configurations

The screenshot shows two terminal windows side-by-side. Both are titled "SEEDUbuntu [Running]". The left terminal shows a user attempting to connect to port 23 on host 10.0.2.10, but receives a "Connection refused" message. The right terminal shows a similar attempt to host 10.0.2.6, also failing with a "Connection refused" message. Both terminals have the same timestamp: [03/09/21].

```
[03/09/21]seed@VM:~/.../lab5$ telnet 10.0.2.10...
Trying 10.0.2.10...
telnet: Unable to connect to remote host: Connection refused
[03/09/21]seed@VM:~/.../lab5$
```

```
[03/09/21]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
telnet: Unable to connect to remote host: Connection refused
[03/09/21]seed@VM:~$
```

Fig. 1.3 Results of telnet attempts

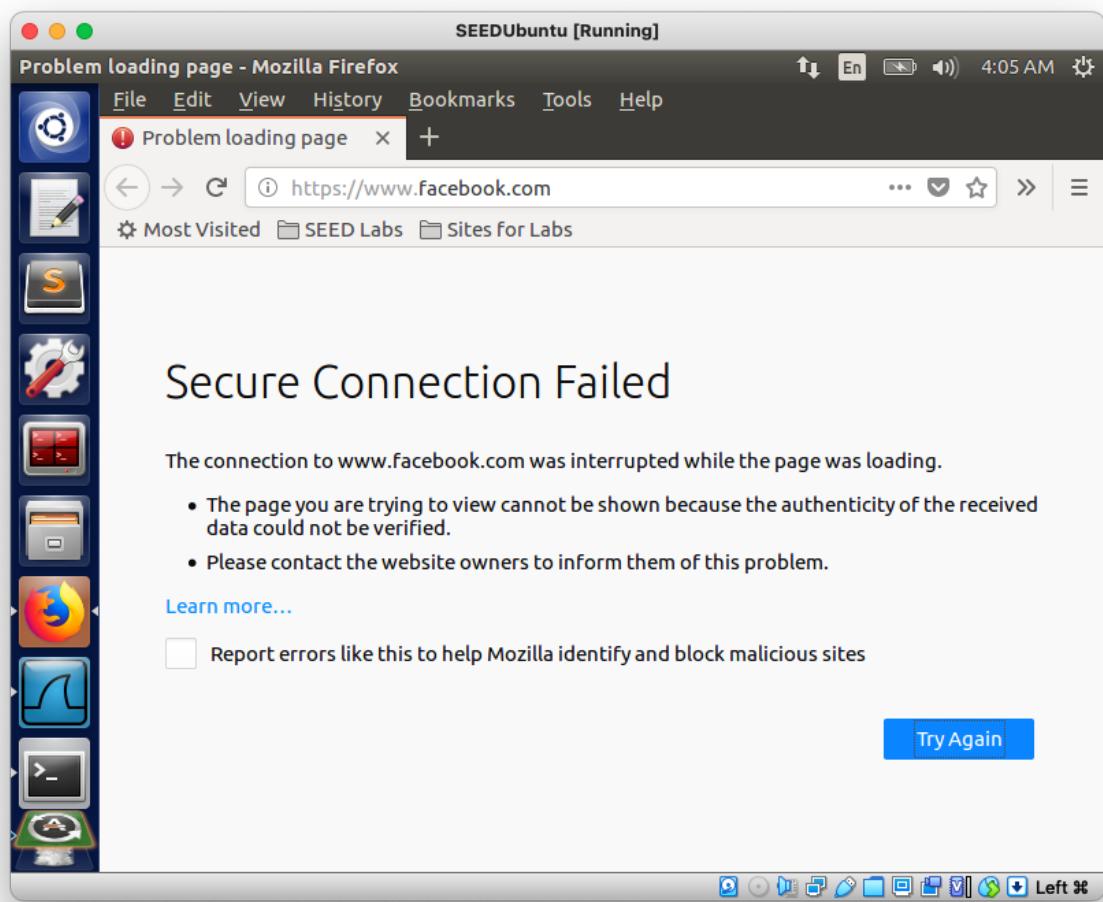


Fig. 1.4 Failure to connect to [www.facebook.com](https://www.facebook.com)

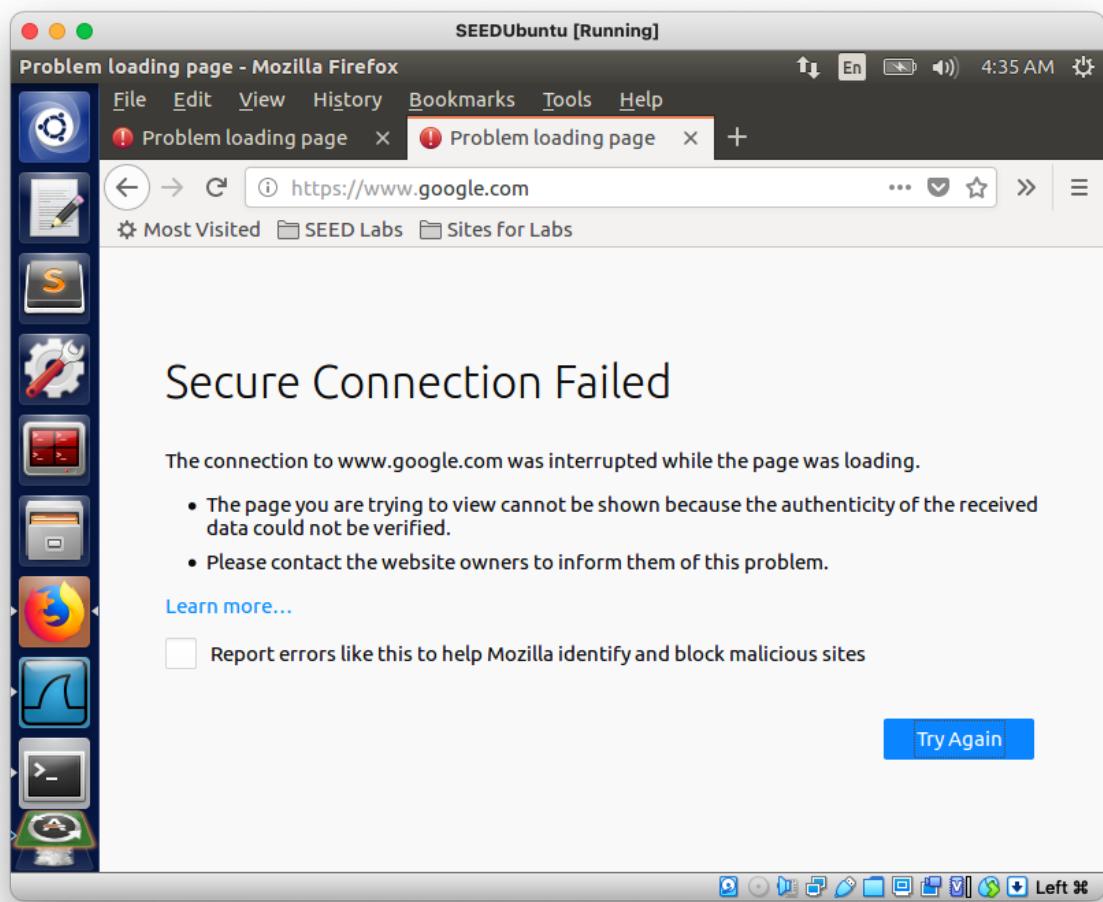


Fig. 1.5 Failure to connect to [www.google.com](https://www.google.com)

## Task 2: Implementing a Simple Firewall

Machine	Name	IP
A	SEEDUbuntu	10.0.2.6
B	SEEDUbuntu Clone	10.0.2.10

I will be using the above configurations and implement the firewall configurations on Machine A for this task.

The five rules implemented in **task2.c** are as follows:

- DROP egress telnet packets
- DROP ingress telnet packets
- DROP egress packets to restricted website (facebook.com)
- DROP ingress ssh packets
- DROP egress ssh packets

Fig. 2.1 shows the code used in **task2.c** to DROP all egress and ingress telnet packets respectively. The code in the green highlighted section in Fig. 2.1 shows the pointer to the current restricted website's (**www.syr.edu**) IP address. We first have to check the restricted website's IP address before compiling this LKM. We can do so with the command **host www.syr.edu** as shown in Fig. 2.3. Fig. 2.2 shows the code used in **task2.c** to DROP all packets destined to the restricted website (**www.syr.edu**) as well as DROP all ingress and egress ssh packets respectively. Fig. 2.4 shows the init and cleanup functions. I specified **NF\_INET\_PRE\_ROUTING** which is triggered by any incoming traffic very soon after entering the network stack. This hook is processed before any routing decisions have been made regarding where to send the packet which satisfies all our rules. I use this so I do not need to compile a separate LKM.

Fig. 2.5 shows the Kernel Ring Buffer. All **printk()** statements in **task2.c** will be logged here. We can view the Kernel Ring Buffer with the command **dmesg**. Fig. 2.5 shows that the 5 rules are in action. Fig. 2.6 shows that Machines A and B are unable to ssh into each other, proving that the firewall is dropping ssh packets. Fig. 2.6 shows that Machines A and B are unable to telnet into each other, proving that the firewall is dropping telnet packets. Fig. 2.7 shows Machine A being unable to visit **www.syr.edu**.

We compile **task2.c** using the Makefile shown in Fig. 2.8 and we do so by typing the command **make**. We then insert the LKM by typing **sudo insmod task2.ko** and we can confirm the module has been inserted by typing **lsmod | grep task2** as shown in Fig. 2.10.

SEEDUbuntu [Running]

Terminal

```
17 #include <linux/kernel.h> /* Needed for KERN_INFO */
18 #include <linux/init.h> /* Needed for the macros */
19 #include <linux/netfilter.h>
20 #include <linux/netfilter_ipv4.h>
21 #include <linux/ip.h>
22 #include <linux/tcp.h>
23 #include <linux/skbuff.h>
24
25 static struct nf_hook_ops nfho;
26
27 unsigned int filter(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)
28 {
29
30     struct iphdr *iph;
31     struct tcphdr *tcph;
32     static unsigned char *ip_address = "\x80\xE6\x12\x3F"; //128.230.18.63 -> www.syr.edu
33     iph = ip_hdr(skb);
34     tcph = (void *)iph+iph->ihl*4;
35
36     if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23))
37     {
38         printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
39             ((unsigned char *)iph->daddr)[0],
40             ((unsigned char *)iph->daddr)[1],
41             ((unsigned char *)iph->daddr)[2],
42             ((unsigned char *)iph->daddr)[3]
43         );
44         return NF_DROP;
45     }
46     else if (iph->protocol == IPPROTO_TCP && tcph->source == htons(23))
47     {
48         printk(KERN_INFO "Dropping telnet packet from %d.%d.%d.%d\n",
49             ((unsigned char *)iph->saddr)[0],
50             ((unsigned char *)iph->saddr)[1],
51             ((unsigned char *)iph->saddr)[2],
52             ((unsigned char *)iph->saddr)[3]
53         );
54         return NF_DROP;
55     }
56 }
```

17,1 22% Left #

Fig. 2.1 Rules to drop outbound and inbound telnet packets

The screenshot shows a dual-terminal window titled "SEEDUbuntu [Running]". The left terminal displays the following kernel code:

```
48     printk(KERN_INFO "Dropping telnet packet from %d.%d.%d.%d\n",
49             ((unsigned char *)&iph->saddr)[0],
50             ((unsigned char *)&iph->saddr)[1],
51             ((unsigned char *)&iph->saddr)[2],
52             ((unsigned char *)&iph->saddr)[3])
53     );
54     return NF_DROP;
55 }
56 else if (iph->saddr == *(unsigned int*)ip_address)
57 {
58     printk(KERN_INFO "User accessing restricted site: www.syr.edu");
59     return NF_DROP;
60 }
61 else if (iph->protocol == IPPROTO_TCP && tcph->source == htons(22))
62 {
63     printk(KERN_INFO "Dropping ssh packet from %d.%d.%d.%d",
64             ((unsigned char *)&iph->saddr)[0],
65             ((unsigned char *)&iph->saddr)[1],
66             ((unsigned char *)&iph->saddr)[2],
67             ((unsigned char *)&iph->saddr)[3])
68     );
69     return NF_DROP;
70 }
71 else if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(22))
72 {
73     printk(KERN_INFO "Dropping ssh packet to %d.%d.%d.%d",
74             ((unsigned char *)&iph->daddr)[0],
75             ((unsigned char *)&iph->daddr)[1],
76             ((unsigned char *)&iph->daddr)[2],
77             ((unsigned char *)&iph->daddr)[3])
78     );
79     return NF_DROP;
80 }
81 else
82 {
83     return NF_ACCEPT;
84 }
```

The right terminal window is currently empty. The desktop environment includes icons for the Dash, Home, Applications, and Help.

Fig. 2.2 Rules to drop packets to www.syr.edu and inbound, outbound ssh packets

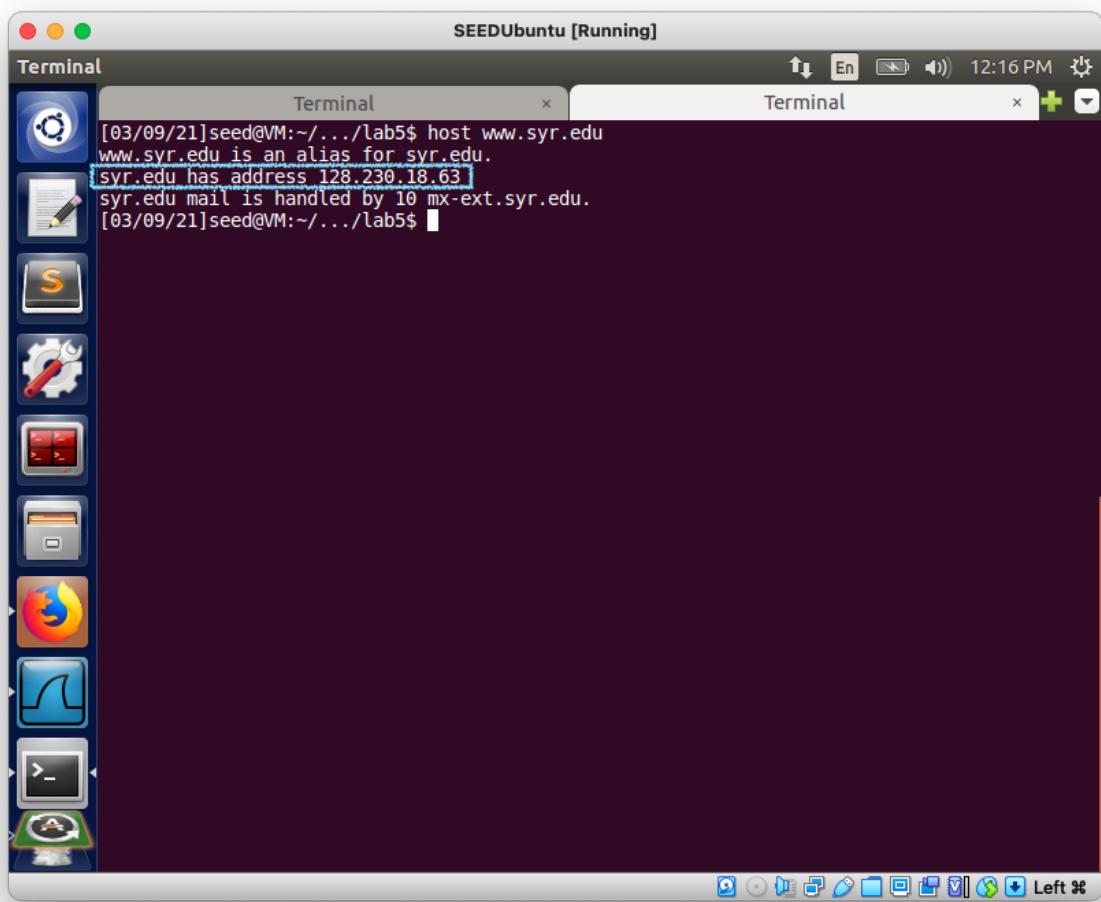


Fig. 2.3 Getting facebook.com's current IP address

The screenshot shows a dual-terminal window titled "SEEDUbuntu [Running]". The left terminal window displays a portion of a C kernel module source code. The code includes a decision point based on a variable 'filter' (line 80), followed by an 'else' block (lines 81-84) that returns NF\_ACCEPT. The main block (lines 72-88) contains printk statements and initializes an nf\_hook\_ops structure named 'nfho'. This structure is configured for NF\_INET\_PRE\_ROUTING, PF\_INET, and NF\_IP\_PRI\_FIRST priority. It also registers a hook function. The right terminal window is currently empty.

```
72     {
73         printk(KERN_INFO "Dropping ssh packet to %d.%d.%d.%d",
74             ((unsigned char *)&iph->daddr)[0],
75             ((unsigned char *)&iph->daddr)[1],
76             ((unsigned char *)&iph->daddr)[2],
77             ((unsigned char *)&iph->daddr)[3]
78         );
79         return NF_DROP;
80     } else
81     {
82         return NF_ACCEPT;
83     }
84 }
85 }

88 int init_module(void)
89 {
90     printk(KERN_INFO "Initialising Task2 filter\n");
91     nfho.hook = filter;
92     nfho.hooknum = NF_INET_PRE_ROUTING;
93     nfho.pf = PF_INET;
94     nfho.priority = NF_IP_PRI_FIRST;
95
96     // Register hook
97     nf_register_hook(&nfho);
98     return 0;
99 }

100 }

101 void cleanup_module(void)
102 {
103     printk(KERN_INFO "Task2 filter is being removed\n");
104     nf_unregister_hook(&nfho);
105 }

106 }

107 }

108 }

109 }
```

Fig. 2.4 init and cleanup functions

SEEDUbuntu [Running]

Terminal

```
[30232.953752] User accessing restricted site: facebook.com
[30233.211139] User accessing restricted site: facebook.com
[30241.012153] User accessing restricted site: facebook.com
[30241.142375] User accessing restricted site: facebook.com
[30241.397473] User accessing restricted site: facebook.com
[30253.618879] User accessing restricted site: facebook.com
[30257.262116] User accessing restricted site: facebook.com
[30257.517993] User accessing restricted site: facebook.com
[30277.900114] Task2 filter is being removed
Initialising Task2 filter
[30281.518345] User accessing restricted site: www.syr.edu
[30312.548196] User accessing restricted site: www.syr.edu
[30312.834654] User accessing restricted site: www.syr.edu
[30313.297586] User accessing restricted site: www.syr.edu
[30313.553441] User accessing restricted site: www.syr.edu
[30315.145315] User accessing restricted site: www.syr.edu
[30315.312539] User accessing restricted site: www.syr.edu
[30315.568597] User accessing restricted site: www.syr.edu
[30315.651088] User accessing restricted site: www.syr.edu
[30319.439805] User accessing restricted site: www.syr.edu
[30319.694557] User accessing restricted site: www.syr.edu
[30325.653994] Dropping ssh packet from 10.0.2.10
[30326.653877] Dropping ssh packet from 10.0.2.10
[30326.667637] Dropping ssh packet from 10.0.2.10
[30327.257953] User accessing restricted site: www.syr.edu
[30328.669341] Dropping ssh packet from 10.0.2.10
[30329.379647] Dropping telnet packet from 10.0.2.10
[30330.396312] Dropping telnet packet from 10.0.2.10
[30330.409343] Dropping telnet packet from 10.0.2.10
[30332.411550] Dropping telnet packet from 10.0.2.10
[30332.699623] Dropping ssh packet from 10.0.2.10
[30336.538165] Dropping telnet packet from 10.0.2.10
[30336.560274] Dropping ssh packet to 10.0.2.6
[30337.560771] Dropping ssh packet to 10.0.2.6
[30339.867896] User accessing restricted site: www.syr.edu
[30340.886737] Dropping ssh packet from 10.0.2.10
[30343.339361] Dropping telnet packet to 10.0.2.6
[30344.341276] Dropping telnet packet to 10.0.2.6
[30344.725989] Dropping telnet packet from 10.0.2.10
[03/09/21]seed@VM:~/lab5$
```

Fig. 2.5 Output Log of Kernel Ring Buffer with *dmesg*

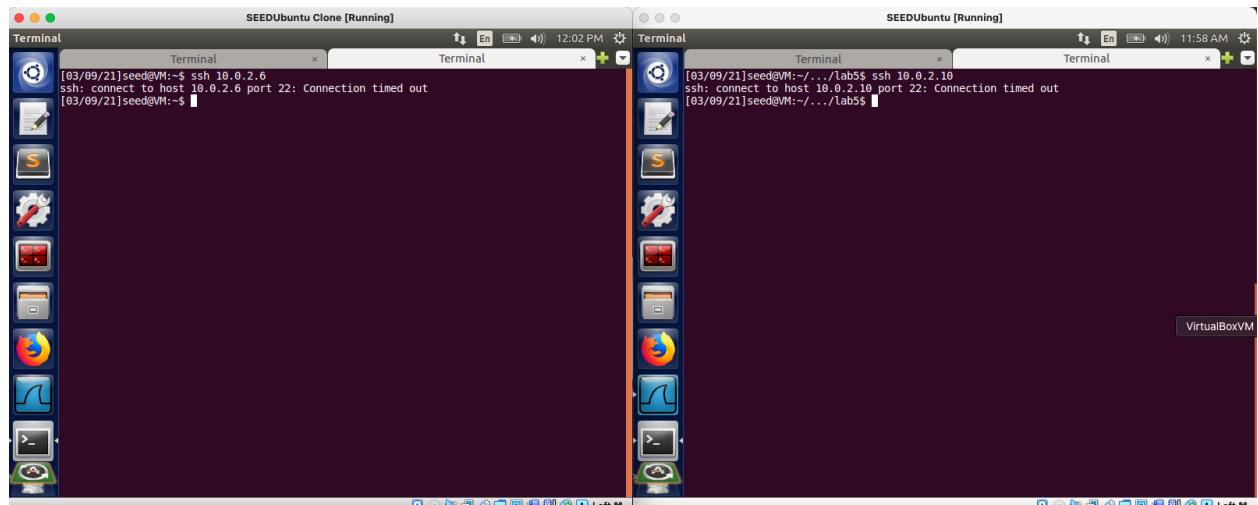


Fig. 2.6 Machine A and B are unable to ssh into each other

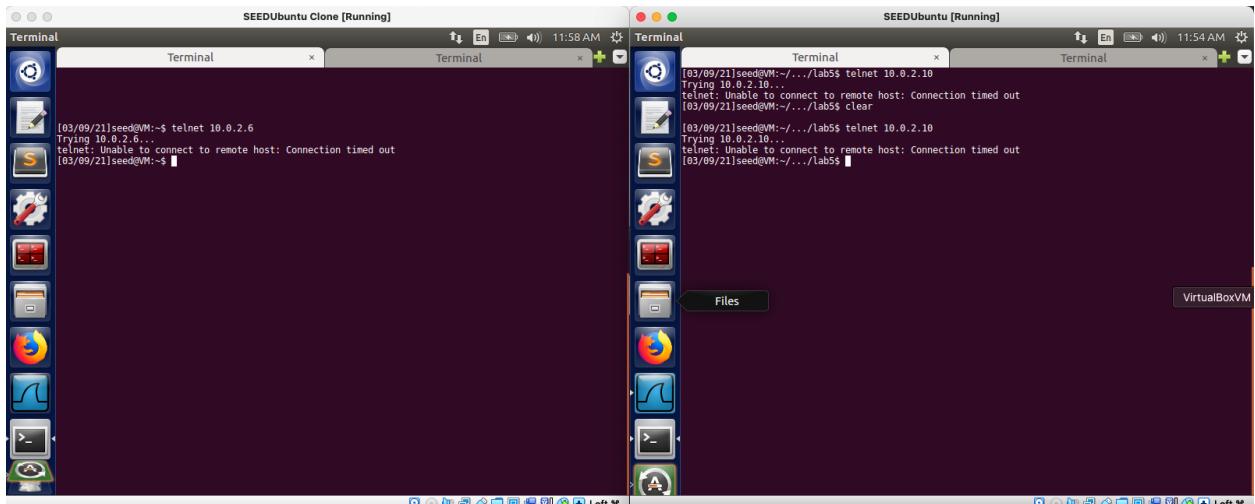


Fig. 2.7 Machine A and B are unable to telnet into each other

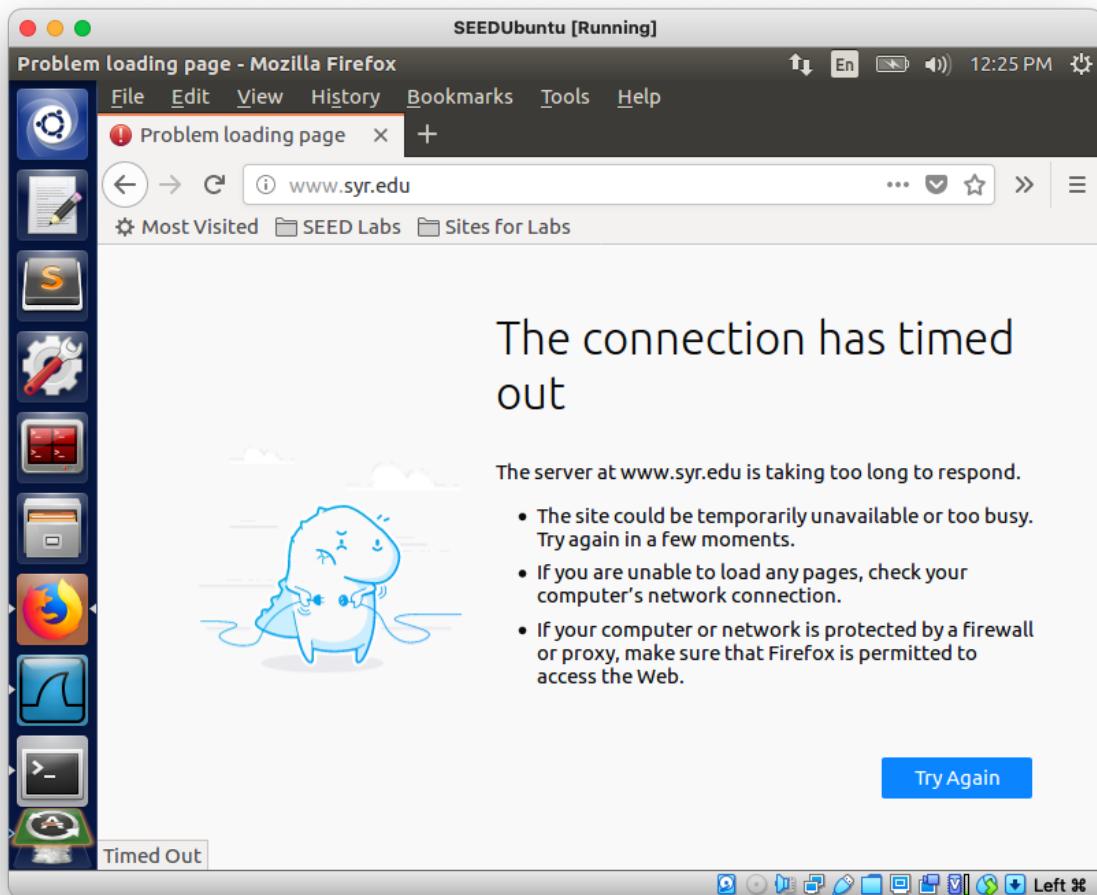


Fig. 2.8 Machine A unable to access [www.syr.edu](http://www.syr.edu)

Fig. 2.9 Makefile

SEEDUbuntu [Running]

Terminal

```
Terminal Terminal
```

```
..: directory
[03/10/21]seed@VM:~/Desktop$ folder .
The program 'folder' can be found in the following packages:
* mailutils-mh
* nmh
Try: sudo apt install <selected package>
[03/10/21]seed@VM:~/Desktop$ ls
lab1 lab2 lab3 lab5
[03/10/21]seed@VM:~/Desktop$ mv lab5 ~/host/
[03/10/21]seed@VM:~/Desktop$ ls ~/host
lab1 lab2 lab3 lab4 lab4_1003331.zip lab5
[03/10/21]seed@VM:~/Desktop$ ls
lab1 lab2 lab3
[03/10/21]seed@VM:~/Desktop$ cp ~/host/lab5
lab5/ lab5_1003331.zip
[03/10/21]seed@VM:~/Desktop$ cp ~/host/lab5
lab5/ lab5_1003331.zip
[03/10/21]seed@VM:~/Desktop$ cp ~/host/lab5 ;
cp: omitting directory '/home/seed/host/lab5';
[03/10/21]seed@VM:~/Desktop$ cp -r ~/host/lab5 .
[03/10/21]seed@VM:~/Desktop$ ls
lab1 lab2 lab3 lab5
[03/10/21]seed@VM:~/.../lab5$ ls
cleanup.sh Makefile Module.symvers task2.c task2.mod.c task2.o task4.sh
Lab 5-1003331.pdf modules.order task1.sh task2.ko task2.mod.o task3.sh ufw-cleanup.sh
[03/10/21]seed@VM:~/.../lab5$ vim Makefile
[03/10/21]seed@VM:~/.../lab5$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Desktop/lab5 modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
CC [M] /home/seed/Desktop/lab5/task2.o
Building modules, stage 2.
MODPOST 1 modules
LD [M] /home/seed/Desktop/lab5/task2.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[03/10/21]seed@VM:~/.../lab5$ sudo insmod task2.ko
[03/10/21]seed@VM:~/.../lab5$ lsmod | grep task2
task2 16384 0
[03/10/21]seed@VM:~/.../lab5$
```

Fig. 2.10 Inserting LKM

## Task 3: Evading Egress Filtering

Machine	Name	IP
A	SEEDUbuntu	10.0.2.6
B	SEEDUbuntu Clone	10.0.2.10
C	SEEDUbuntu Clone 1	10.0.2.7

I will be using the above configurations and implement the firewall configurations on Machine A for this task.

We were tasked to setup 2 firewall rules:

- Block all the outgoing traffic to external telnet servers
- Block all the outgoing traffic to [www.facebook.com](http://www.facebook.com)

I have written the code to do so in **task3.sh** shown in Fig. 3.1. Since it's a subset of task 1, I shall not explain the code. Fig. 3.2 shows the failed attempt on Machine A to visit **facebook.com**.

The screenshot shows a dual-terminal window titled "SEEDUbuntu [Running]". The left terminal window is active and displays the following bash script content:

```
1 #!/bin/bash
2
3 # reject egress telnet packets
4 iptables -A OUTPUT -p tcp --dport telnet -j REJECT
5
6 # block egress packets to external website - www.facebook.com
7 iptables -A OUTPUT -p tcp -m string --string "facebook.com" --algo kmp -j DROP
8
```

The right terminal window is inactive and shows a blank command line interface.

At the bottom of the left terminal window, the status bar indicates: "task3.sh" 8L, 237C written. The desktop environment includes a dock with icons for various applications like a terminal, file manager, browser, and system tools.

Fig. 3.1 iptables configurations in **task3.sh**

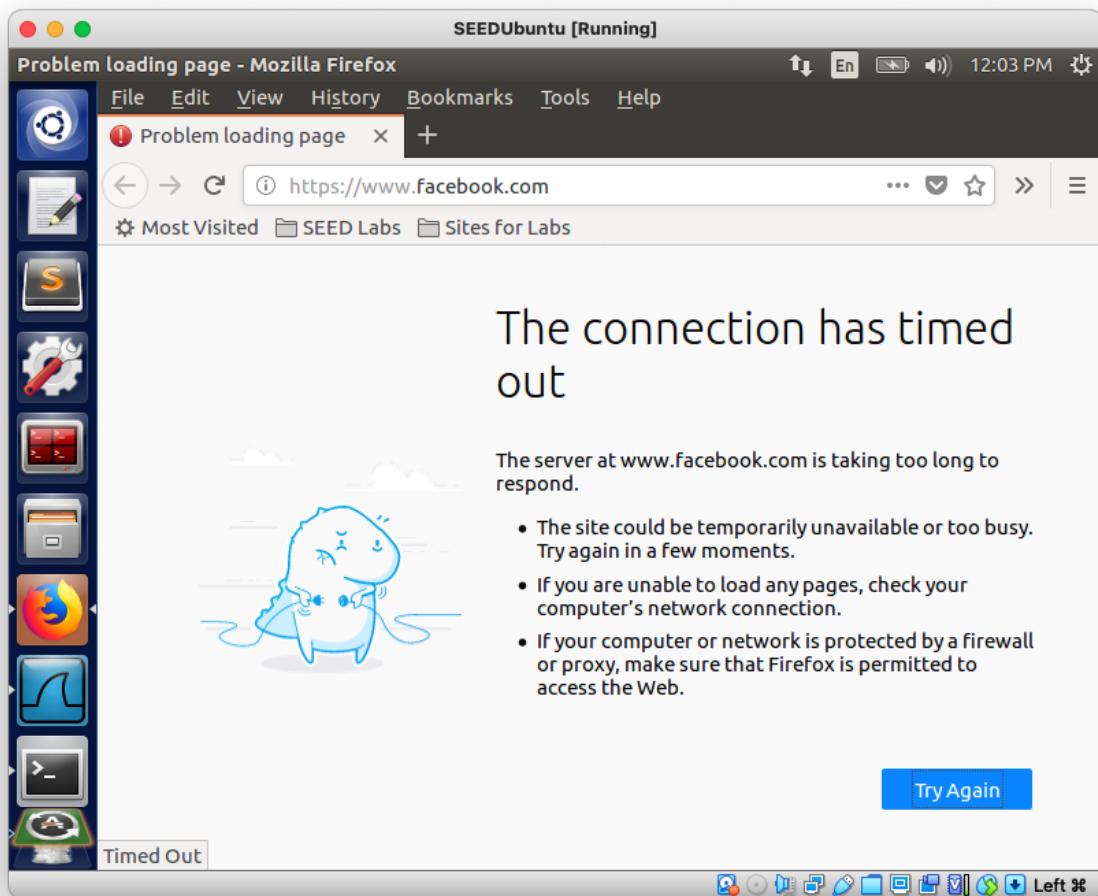


Fig. 3.2 Machine A unable to access ***facebook.com***

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

To get a clean slate of our iptables, I ran the clean up script shown in Fig. 1.0 (Task 1) with the command ***sudo bash cleanup.sh***. After which, I ran the iptable configuration script ***task3.sh*** shown in Fig. 3.1 (Task 3) with the command ***sudo bash task3.sh***. We can see the following illustrated in the first red highlighted box in Fig. 3.a.1 along with the updated iptables.

Next, to establish the SSH tunnel between Machine A and B, and port forward packets coming out of B to Machine C's port 23, I ran the command ***ssh -f -N -L 8000:10.0.2.7:23 seed@10.0.2.10***. The use of flags are explained as such:

- **-f** : Requests **ssh** to go to background just before command execution.
- **-N** : Do not execute a remote command. This is useful for just forwarding ports
- **-L** :Specifies that the given port on the local (client) host is to be forwarded to the given host and port on the remote side.

Adding these flags will send the ssh connection into the background, allowing me to use the current shell after typing in Machine B's password. This means the current shell after establishing the SSH tunnel still belongs to Machine A (**10.0.2.6**). We can confirm the establishment of the SSH tunnel with the command ***netstat -tna*** as shown in the blue highlighted box in Fig. 3.a.1.

Now, we telnet to Machine C (**10.0.2.7**) via the SSH tunnel with the command ***telnet localhost 8000*** as shown in Fig. 3.a.2. After entering the login details on Machine C, we have successfully established a telnet session between A and C via B. We can see that this is true as the output of ***hostname -I*** on Machine A changes from Machine A's IP address to Machine C's IP address after establishing the telnet session. This can be seen in the first and last blue highlighted box in Fig. 3.a.2.

I will now discuss my findings in Wireshark for Task 3a. In Fig. 3.a.3, we see the establishment of SSH session between Machine A (**10.0.2.6**) and Machine B (**10.0.2.10**) and we can confirm this is the case by looking at the packets' **source** and **dest** addresses as well as the negotiation of cipher suite and acquiring client keys during this period.

In Fig. 3.a.4, we see the attempt by Machine A to establish a telnet session to Machine C via Machine B. We can confirm this behavior by looking at the order of packets in the wireshark capture in Fig. 3.a.4 as such:

1. A → B (**10.0.2.6** → **10.0.2.10**)
2. B → C (**10.0.2.10** → **10.0.2.7**)
3. C → B (**10.0.2.7** → **10.0.2.10**)
4. B → A (**10.0.2.10** → **10.0.2.6**)

Furthermore, the encrypted telnet packet from C → B (#3 in the above order) shows its payload as the telnet login session. We see that the next packet is from B → A (#4 in the above order) and because it is encrypted (SSH protocol), we are unable to view its contents. However, we know that this packet is the packet (#3 in the above order) is forwarded to A via the SSH tunnel.

After entering the login credentials via the telnet session, Machine A is able to bypass the firewall restrictions. We can see this in Fig. 3.a.6 where the first red highlighted box shows the welcome message from Machine C to A. The next green highlighted box is the ACK from A to B and the following telnet packets are the rest of the standard information one would see when successfully establishing a telnet session (VM Information).

This method works because the firewall restriction on Machine A only restricts egress telnet packets but not bidirectional ssh packets. We are able to leverage this by creating an SSH tunnel between A and B through which B forwards packets coming out of this tunnel to the telnet port on Machine C. This means the firewall only sees SSH packets going in and out of Machine A. It can not see the telnet packets between B and C. Furthermore, the SSH packets are encrypted.

[03/10/21]seed@VM:~/.../lab5\$ sudo bash cleanup.sh  
[03/10/21]seed@VM:~/.../lab5\$ sudo bash task3.sh  
[03/10/21]seed@VM:~/.../lab5\$ sudo iptables -L  
Chain INPUT (policy ACCEPT)  
target prot opt source destination  
  
Chain FORWARD (policy ACCEPT)  
target prot opt source destination  
  
Chain OUTPUT (policy ACCEPT)  
target prot opt source destination  
REJECT tcp -- anywhere anywhere tcp dpt:telnet reject-with icmp-port-unreachable  
DROP tcp -- anywhere anywhere STRING match "facebook.com" ALGO name km  
o TO 65535  
[03/10/21]seed@VM:~/.../lab5\$ ssh -f -N -L 8000:10.0.2.7:23 seed@10.0.2.10  
seed@10.0.2.10's password:  
[03/10/21]seed@VM:~/.../lab5\$ netstat -tana  
Active Internet connections (servers and established)  
Proto Recv-Q Send-Q Local Address Foreign Address State  
tcp 0 0 10.0.2.6:53 0.0.0.0:\*

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	10.0.2.6:53	0.0.0.0:*	LISTEN
tcp	0	0	127.0.1.53	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:53	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:23	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:953	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:8000	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:3306	0.0.0.0:*	LISTEN
tcp	0	0	10.0.2.6:42738	10.0.2.10:22	ESTABLISHED
tcp	0	0	10.0.2.6:42730	10.0.2.10:22	TIME_WAIT
tcp6	0	0	:::80	:::*	LISTEN
tcp6	0	0	:::53	:::*	LISTEN
tcp6	0	0	:::21	:::*	LISTEN
tcp6	0	0	:::22	:::*	LISTEN
tcp6	0	0	:::3128	:::*	LISTEN
tcp6	0	0	:::1:953	:::*	LISTEN
tcp6	0	0	:::1:8000	:::*	LISTEN

[03/10/21]seed@VM:~/.../lab5\$

Fig. 3.a.1 Establishment of SSH Tunnel between A and B

The screenshot shows a dual-terminal window titled "SEEDUbuntu [Running]". The left terminal displays a netstat -an output, listing various TCP and UDP ports and their states. The right terminal shows a telnet session connected to localhost port 8000, displaying the Ubuntu 16.04.2 LTS login screen.

```
tcp        0      0 10.0.2.6:53          0.0.0.0:*          LISTEN
tcp        0      0 127.0.1.1:53          0.0.0.0:*          LISTEN
tcp        0      0 127.0.0.1:53          0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:22           0.0.0.0:*          LISTEN
tcp        0      0 0.0.0.0:23           0.0.0.0:*          LISTEN
tcp        0      0 127.0.0.1:953         0.0.0.0:*          LISTEN
tcp        0      0 127.0.0.1:8000         0.0.0.0:*          LISTEN
tcp        0      0 127.0.0.1:3306         0.0.0.0:*          LISTEN
tcp        0      0 10.0.2.6:42738        10.0.2.10:22       ESTABLISHED
tcp        0      0 10.0.2.6:42730        10.0.2.10:22       TIME_WAIT
tcp6       0      0 ::1:80              ::*:               LISTEN
tcp6       0      0 ::1:53              ::*:               LISTEN
tcp6       0      0 ::1:21              ::*:               LISTEN
tcp6       0      0 ::1:22              ::*:               LISTEN
tcp6       0      0 ::1:3128             ::*:               LISTEN
tcp6       0      0 ::1:953              ::*:               LISTEN
tcp6       0      0 ::1:8000             ::*:               LISTEN
[03/10/21]seed@VM:~/.../lab5$ hostname -I
10.0.2.6
[03/10/21]seed@VM:~/.../lab5$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Mar 10 01:04:50 EST 2021 from 10.0.2.10 on pts/4
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[03/10/21]seed@VM:~$ hostname -I
10.0.2.7
[03/10/21]seed@VM:~$
```

Fig. 3.a.2 Establishment of Telnet Session via SSH Tunnel

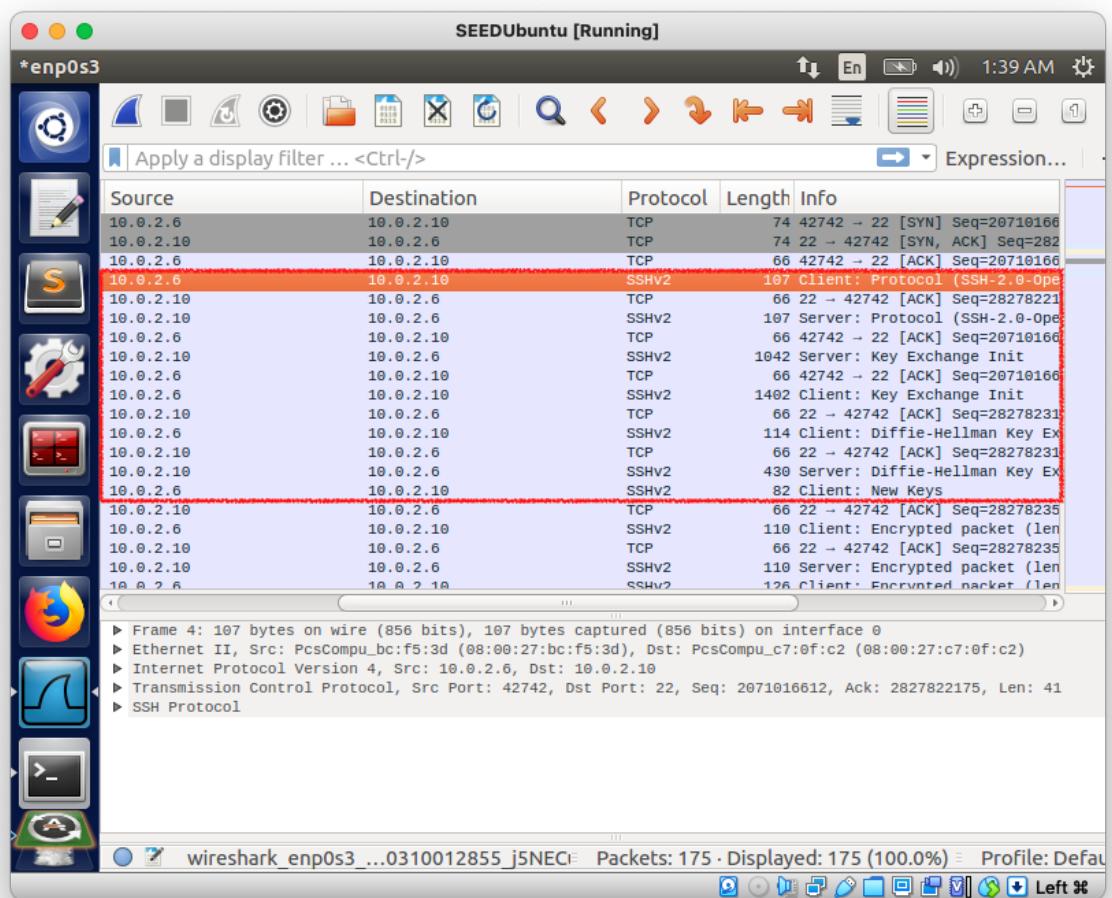


Fig. 3.a.3 Establishment of SSH Session between Machine A and B

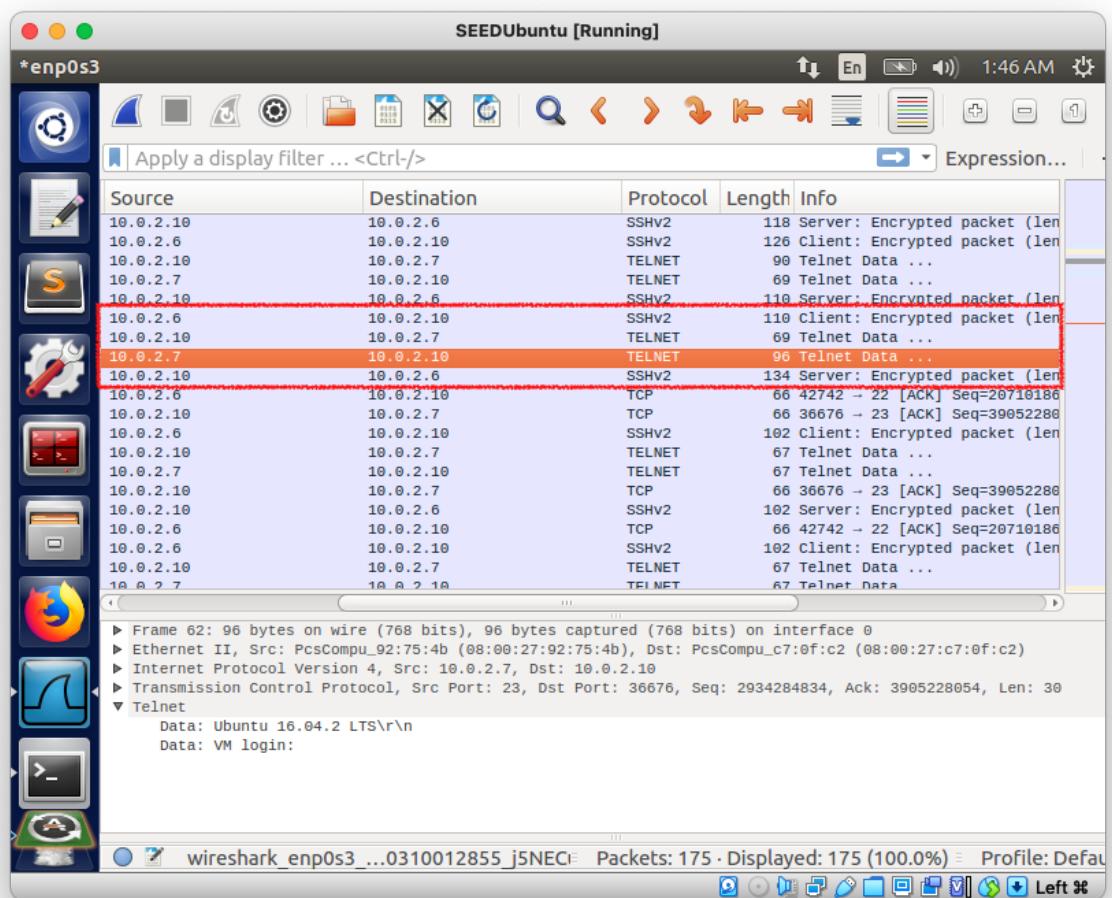


Fig. 3.a.4 Machine A telnet to Machine C via Machine B

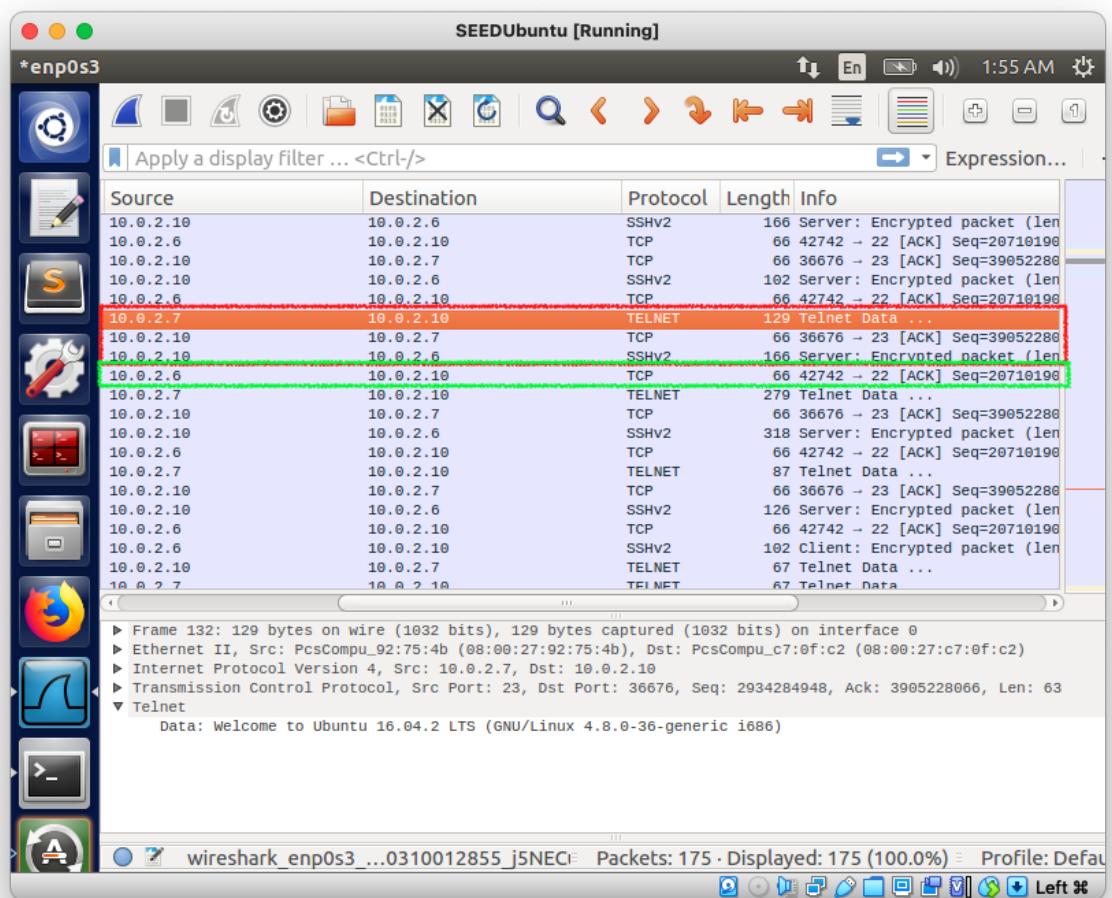


Fig. 3.a.5 Successful telnet session from A to C via B

## Task 3.b: Connect to Facebook using SSH Tunnel

This task was quite interesting because I ran into unexpected results which I will detail in the following paragraphs. The task was to bypass the firewall restrictions (restricting Machine A to visit **facebook.com**). To set up the restriction, I entered in the bash script **iptables -A OUTPUT -p tcp -m string --string "facebook.com" --algo kmp -j DROP** as shown in the last line of Fig. 3.1. Fig. 3.b.2 shows the iptables configuration after running **task3.sh** (Fig. 3.1). This way, the firewall will account for all IP addresses of facebook.com. From the firewalls' perspective, this is good.

Next, we set the dynamic SSH Tunnel between Machines A and B as shown in the green highlighted box in Fig. 3.b.3. We then proceed to configure the SOCKS Proxy on Firefox on Machine A as shown in Fig. 3.b.1. Now that we have everything setup, I will now visit **facebook.com** via Firefox and we fail to do so as illustrated in Fig. 3.b.4. This is not the expected behavior. To diagnose this problem, I set up wireshark on both Machines A and B before visiting **facebook.com** on Machine A. The results are shown in Fig. 3.b.5. The wireshark capture of Machine A (right VM in Fig. 3.b.5) shows packets in the following order:

- Loopback interface 127.0.0.1 (TCP)
- 10.0.2.6 → 10.0.2.10 (SSH)
- 10.0.2.10 → 10.0.2.6 (SSH)

This is because when we visit **facebook.com** via Firefox browser, the TCP packets are first sent to the SOCKS proxy via the loopback interface on port 9000. Since there is the existing SSH Tunnel that port forwards these TCP packets to Machine B (10.0.2.10), we see SSH packets from 10.0.2.6 to 10.0.2.10. From which, we receive replies. We can assume that this is the application data of **facebook.com**. The wireshark capture of Machine B (left VM in Fig. 3.b.6) shows packets in the following order:

- 10.0.2.6 → 10.0.2.10 (SSH)
- 10.0.2.10 → 157.240.217.35 (one of facebook's IP addresses) (TCP)
- 157.240.217.35 → 10.0.2.10 (TCP)
- 10.0.2.10 → 10.0.2.6 (SSH)

This is because Machine B (10.0.2.10) first receives packets SSH packets from 10.0.2.6 via the SSH Tunnel. It then processes the request visit **facebook.com** and receives a reply from it. Machine B then forwards this reply via the SSH Tunnel to Machine A. I would expect Machine's A Firefox Browser to serve **facebook.com** but it did not. My hypothesis is that the firewall rule which uses the kpm algorithm to match the hostname is causing this.

To test my hypothesis, I will set up two rules and see if we are able to bypass these rules via SSH Tunnel:

- Block static website ([www.syr.edu](http://www.syr.edu)) via iptables
- Block one ip address of **facebook.com**

I have done this by adding two new lines of code in **task3.sh** and commented out the previous iptable rule that used the kpm algo. This is illustrated in Fig. 3.b.6:

- **iptables -A OUTPUT -p tcp --destination 157.240.217.35 -j DROP**
- **iptables -A OUTPUT -p tcp --destination 128.230.18.63 -j DROP**

**157.240.217.35** is one of facebook's IP addresses and **128.230.18.63** is the static ip address of [www.syr.edu](http://www.syr.edu). After running the updated **task3.sh** with the command **sudo bash task3.sh**, we see the new iptable configurations in Fig. 3.b.7.

With the SSH Tunnel and SOCKS proxy set up, we now visit both facebook.com and [www.syr.edu](http://www.syr.edu). Fig. 3.b.8 shows the successful attempt in visiting [www.syr.edu](http://www.syr.edu) and Fig. 3.b.9 shows the successful attempt in visiting **facebook.com**. Since both attempts illustrate that we are able to bypass iptable firewall rules, I will describe the process for **facebook.com** with Fig. 3.b.10.

In Fig. 3.b.10, the right VM (A) shows a wireshark capture of the following order:

- Loopback interface 127.0.0.1 (TCP)
- 10.0.2.6 → 10.0.2.10 (SSH)
- 10.0.2.10 → 10.0.2.6 (SSH)

This is similar to the case when we were trying to visit facebook.com but with the iptable rule using the kmp algorithm. Basically speaking, TCP packets are routed through the SOCKS proxy to Machine B via the SSH Tunnel.

In Fig. 3.b.10, the left VM (B) shows a wireshark capture of the following order:

- 10.0.2.6 → 10.0.2.10 (SSH)
- 10.0.2.10 → 157.240.7.35 (one of facebook's IP addresses) (TCP)
- 157.240.217.35 → 10.0.2.10 (TCP)
- 10.0.2.10 → 10.0.2.6 (SSH)

This is also similar to the case when we tried to visit facebook.com with the iptable rule using the kmp algorithm. The only difference now is that Machine is actually A to load up facebook.com as seen in Fig. 3.b.9.

After breaking the SSH Tunnel, clearing Firefox cache, Machine A is unable to visit **facebook.com** and this is expected as these packets are dropped by the iptable configuration. Instead, we get served a page stating that the Proxy will not connect to **facebook.com** as shown in Fig. 3.b.11.

We now reestablish our SSH Tunnel and Machine A is able to visit **facebook.com** once more and this is expected. Essentially, the TCP packets to **facebook.com** are routed via the SSH tunnel which iptables rules are unable to pick out. Hence, Machine A is able to connect to the restricted website **facebook.com**.

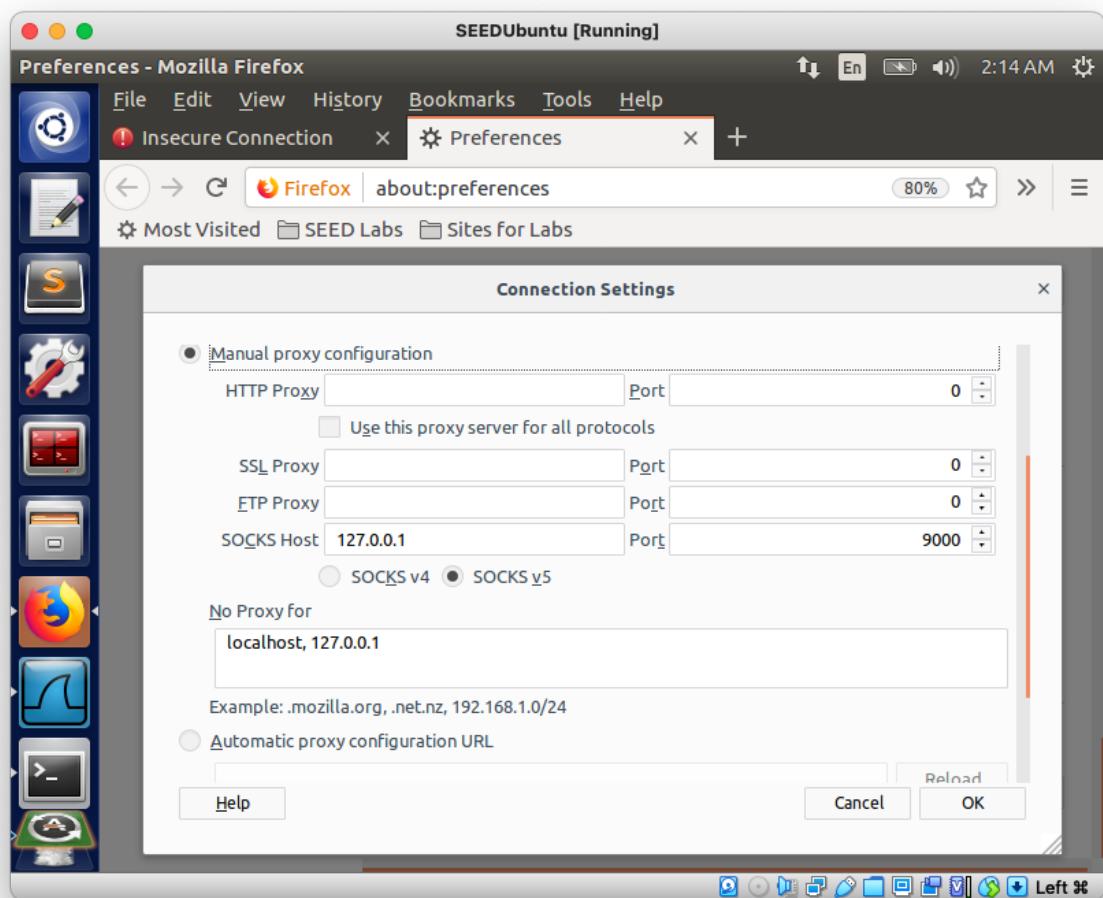


Fig. 3.b.1 Configuring Firefox SOCKS Proxy

```
[03/10/21]seed@VM:~/.../lab5$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
REJECT    tcp  --  anywhere             anywhere            tcp dpt:telnet reject-with icmp-port-unre
achable
DROP      tcp  --  anywhere             anywhere            STRING match  "facebook.com" ALGO name km
p TO 65535
[03/10/21]seed@VM:~/.../lab5$
```

Fig. 3.b.2 iptable configuration on Machine A

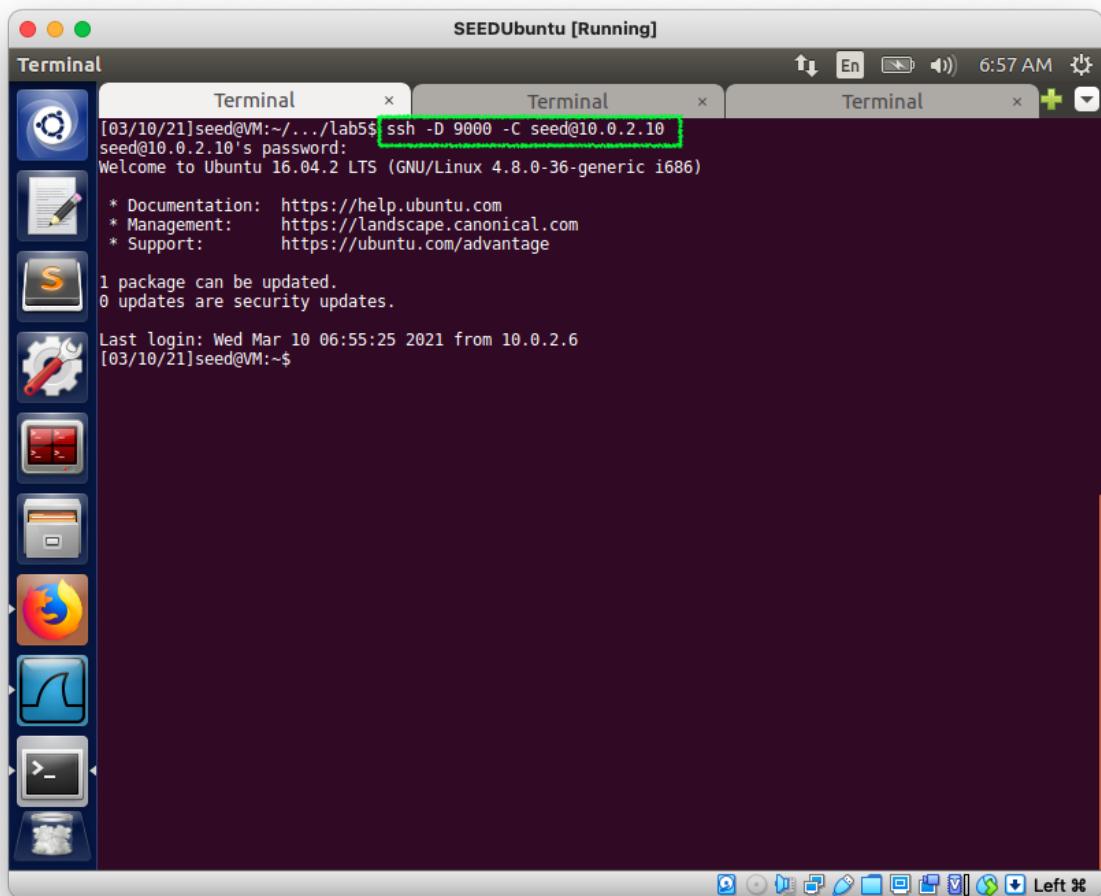


Fig. 3.b.3 Setting up Dynamic SSH Tunnel between A and B

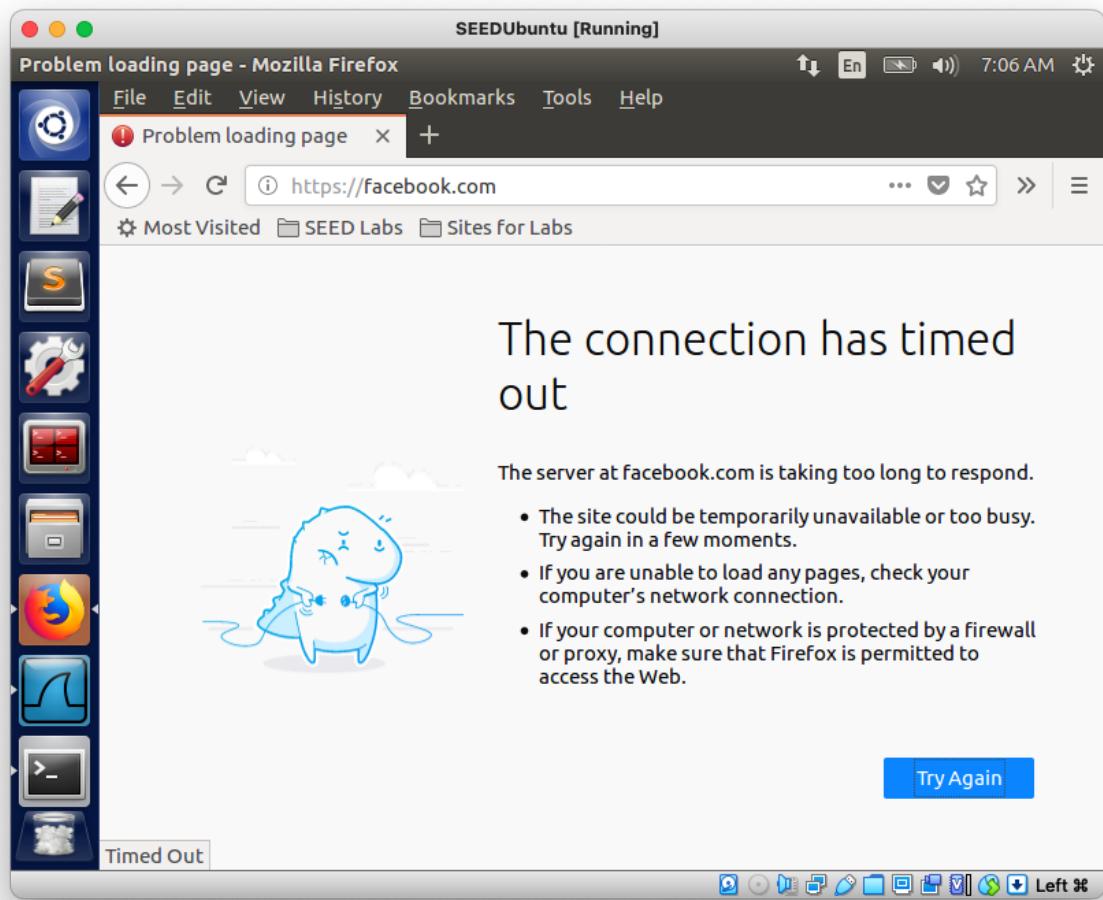


Fig. 3.b.4 Failure to connect to facebook.com on Machine A

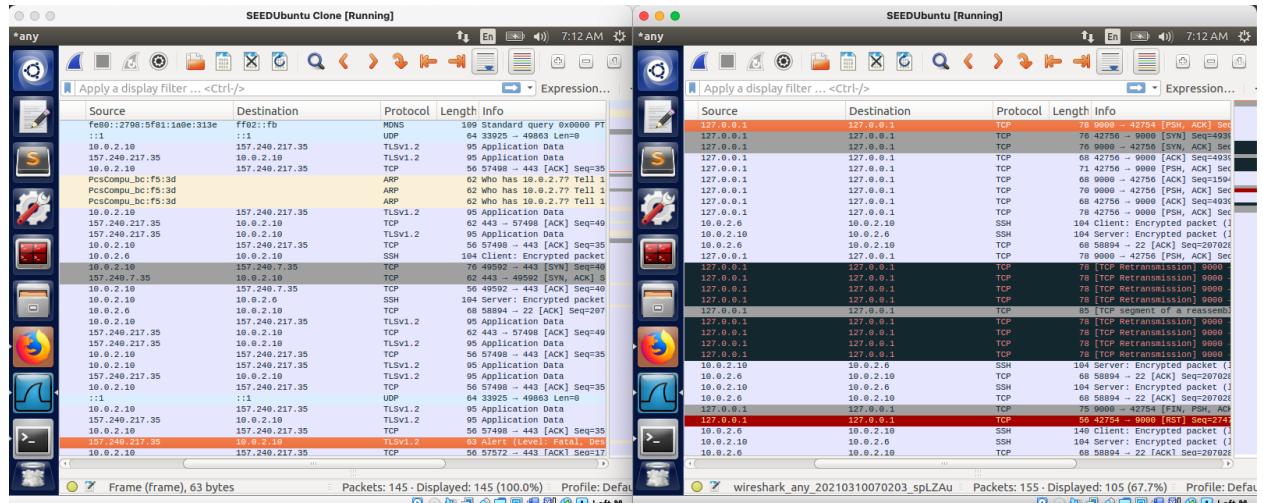


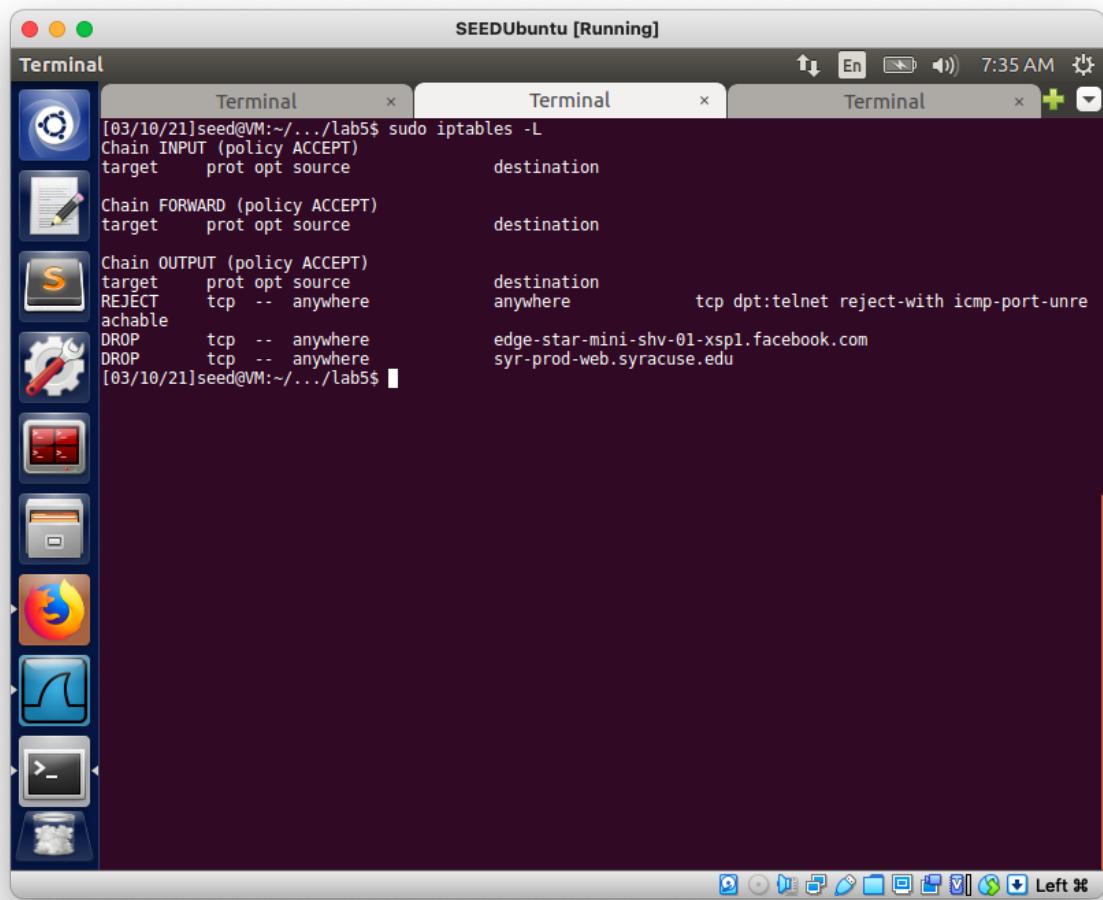
Fig. 3.b.5 Wireshark Capture of Machine A (right) and B (left)

The screenshot shows a desktop environment with a dark theme. On the left is a vertical dock containing icons for various applications: Dash, Nautilus, System Settings, Dash, Home, and a terminal window. Three terminal windows are open in a horizontal stack at the top of the screen. The central terminal window contains the following text:

```
1 #!/bin/bash
2
3 # reject egress telnet packets
4 iptables -A OUTPUT -p tcp --dport telnet -j REJECT
5
6 # block egress packets to external website - www.facebook.com
7 #iptables -A OUTPUT -p tcp -m string --string "facebook.com" --algo kmp -j DROP
8
9 # block egress packet to 1 of the many IPs of external website - www.facebook.com (157.240.217.35)
10 iptables -A OUTPUT -p tcp --destination 157.240.217.35 -j DROP
11
12 # block egress packet to static website - www.syr.edu (128.230.18.63)
13 iptables -A OUTPUT -p tcp --destination 128.230.18.63 -j DROP
```

The bottom of the screen features a dock with icons for various applications, including a file manager, terminal, browser, and system settings. The status bar at the bottom right shows the time as 7:28 AM and battery level as Left 3%.

Fig. 3.b.6 new task3.sh



```
[03/10/21]seed@VM:~/.../lab5$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
REJECT    tcp  --  anywhere             anywhere            tcp dpt:telnet reject-with icmp-port-unre
achable
DROP      tcp  --  anywhere             edge-star-mini-shv-01-xsp1.facebook.com
DROP      tcp  --  anywhere             syr-prod-web.syracuse.edu
[03/10/21]seed@VM:~/.../lab5$
```

Fig. 3.b.7 New iptable configuration

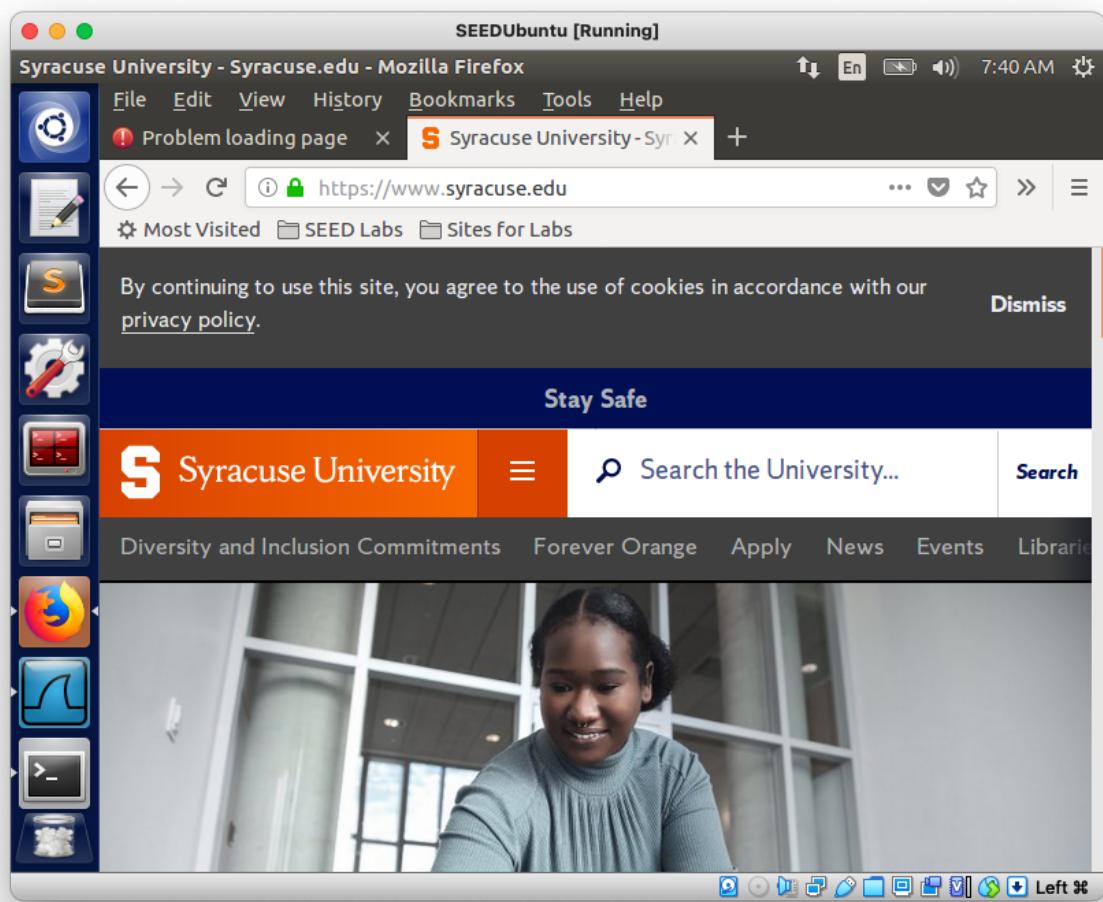


Fig. 3.b.8 Successfully able to visit [www.syr.edu](https://www.syracuse.edu) on Machine A

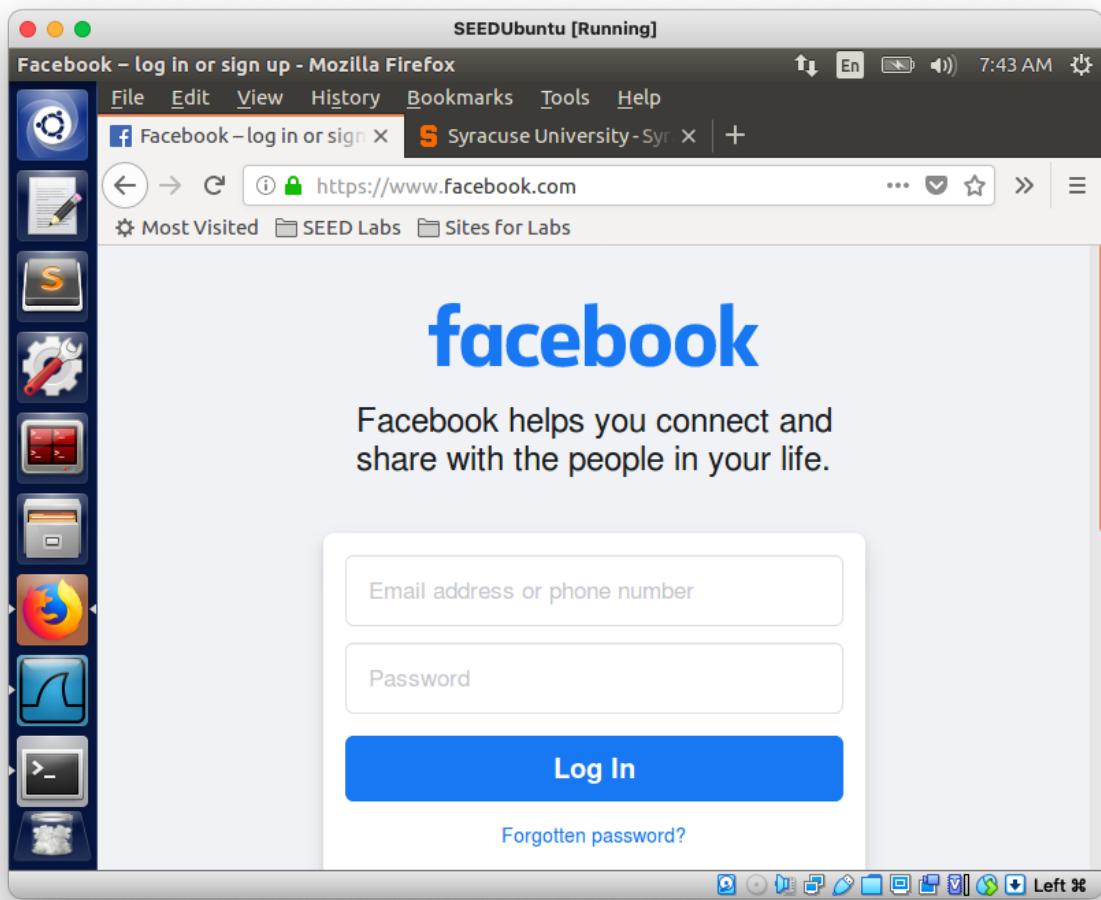


Fig. 3.b.9 Successfully able to visit facebook.com on Machine A

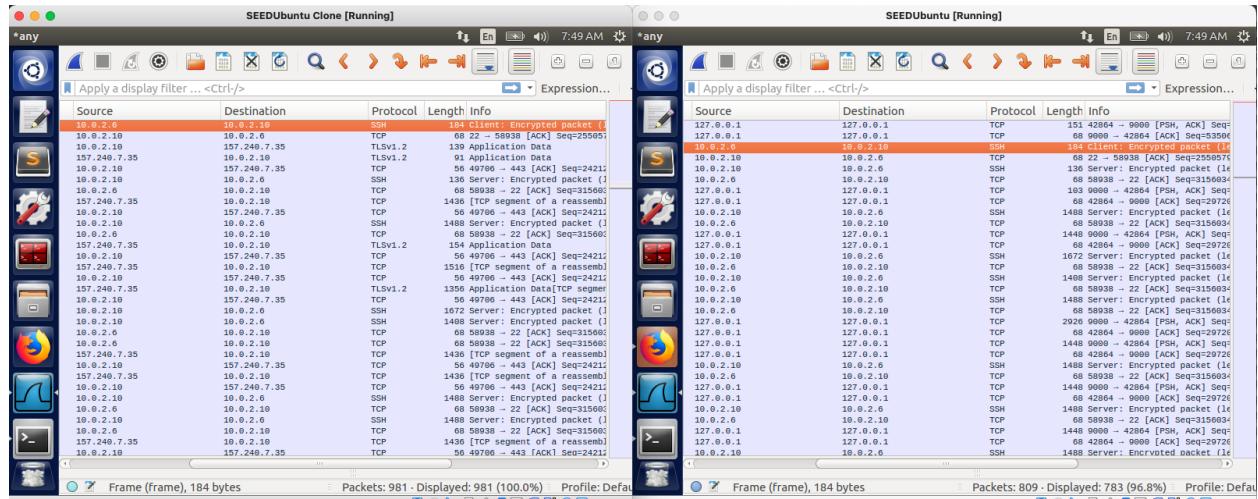


Fig. 3.b.10 Wireshark Capture of successful visit to facebook.com on Machine A

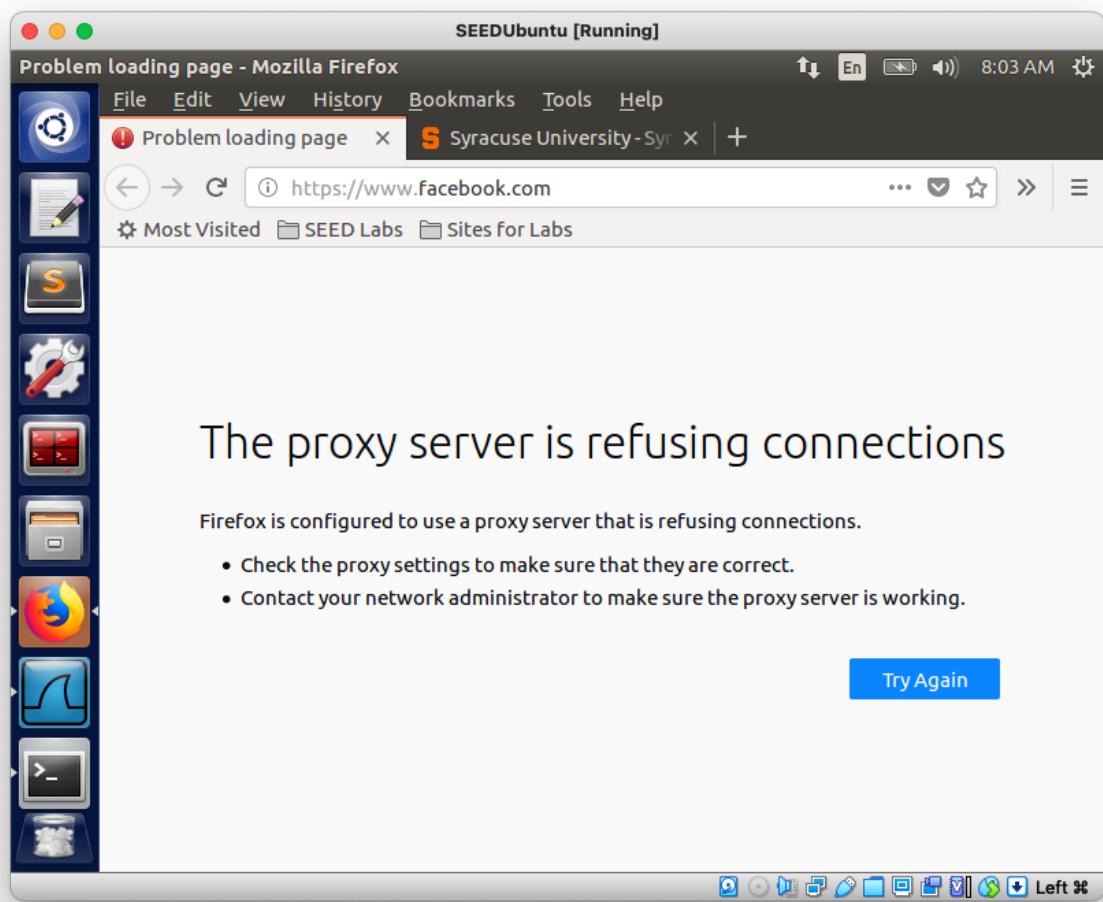


Fig. 3.b.11 SOCKS Proxy refuse to visit facebook.com

## Task 4: Evading Ingress Filtering

Machine	Name	IP
A	SEEDUbuntu	10.0.2.6
B	SEEDUbuntu Clone	10.0.2.10

I will be using the above configurations and implement the firewall configurations on Machine A for this task.

We first configure the firewall on Machine A to block Machine B from accessing its port 80 (web server) and 22 (SSH server). This is illustrated in the output of ***sudo iptables -L*** in Fig. 4.1 after running ***task4.sh***. We can see the contents of ***task4.sh*** in Fig. 4.2. Fig. 4.3 shows the successful results of rejecting B's attempt to access A's web server and ssh server via the wget and ssh utilities respectively.

Reverse SSH tunneling relies on the remote computer using the established connection to listen for new connection requests from the local computer.

To create the reverse SSH Tunnel, we can specify the command on the remote machine (Machine A) ***sudo -R 9000:localhost:22 seed@10.0.2.10*** as seen in Fig. 4.4. Now, we can access Machine A's web server by visiting ***localhost:9000*** in Firefox browser as shown in Fig. 4.5.

SEEDUbuntu [Running]

Terminal

```
[03/10/21]seed@VM:~/.../lab5$ rm index.html
[03/10/21]seed@VM:~/.../lab5$ vim task4.sh
[03/10/21]seed@VM:~/.../lab5$ sudo bash cleanup.sh
[03/10/21]seed@VM:~/.../lab5$ sudo bash task4.sh
[03/10/21]seed@VM:~/.../lab5$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
REJECT    tcp  --  10.0.2.10        anywhere      tcp dpt:http reject-with icmp-port-unreachable
REJECT    tcp  --  10.0.2.10        anywhere      tcp dpt:ssh reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
[03/10/21]seed@VM:~/.../lab5$
```

Fig. 4.1 SOCKS Proxy refuse to visit facebook.com

SEEDUbuntu [Running]

Terminal

```
1 #!/bin/bash
2
3 # reject ingress tcp packets for port 80 from 10.0.2.10
4 iptables -A INPUT -s 10.0.2.10 -p tcp --dport 80 -j REJECT
5
6 # reject ingress tcp packets for port 22 from 10.0.2.10
7 iptables -A INPUT -s 10.0.2.10 -p tcp --dport ssh -j REJECT
8
```

"task4.sh" 8L, 245C

1,1 All

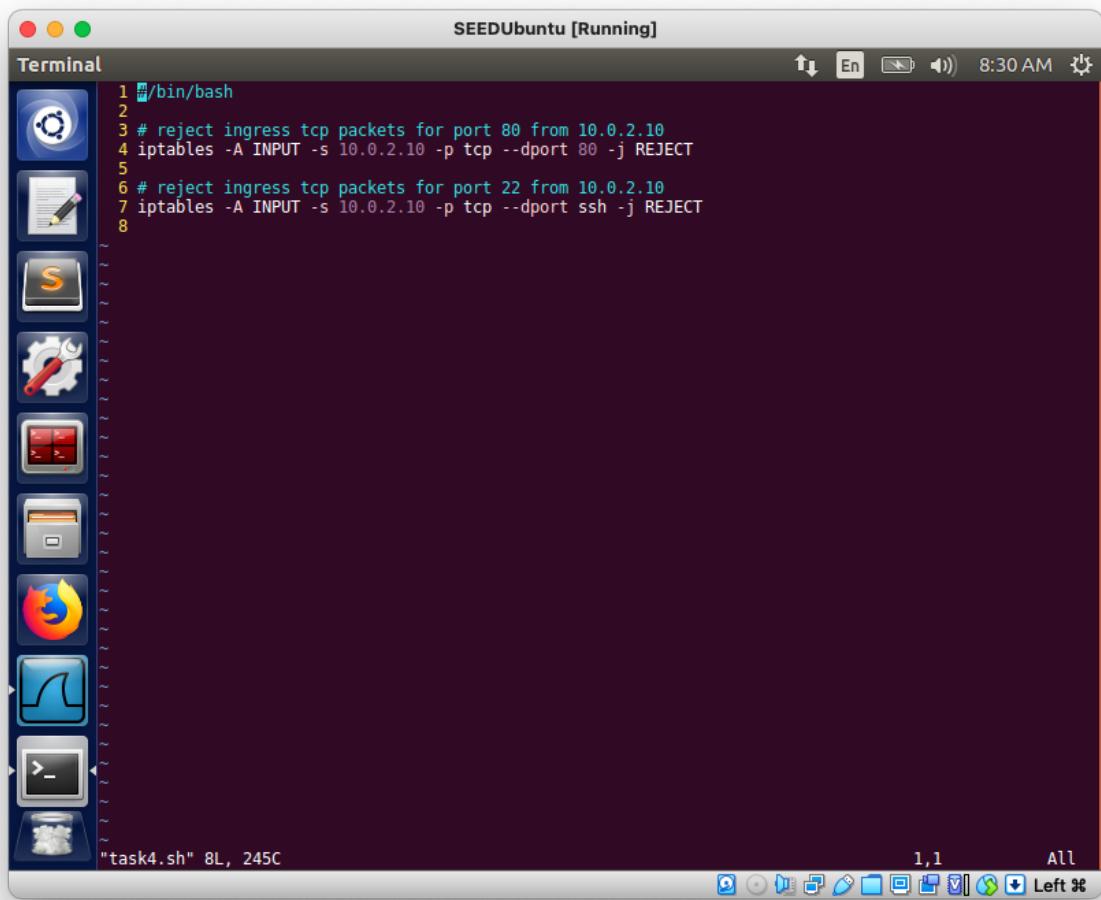


Fig. 4.2 **task4.sh**

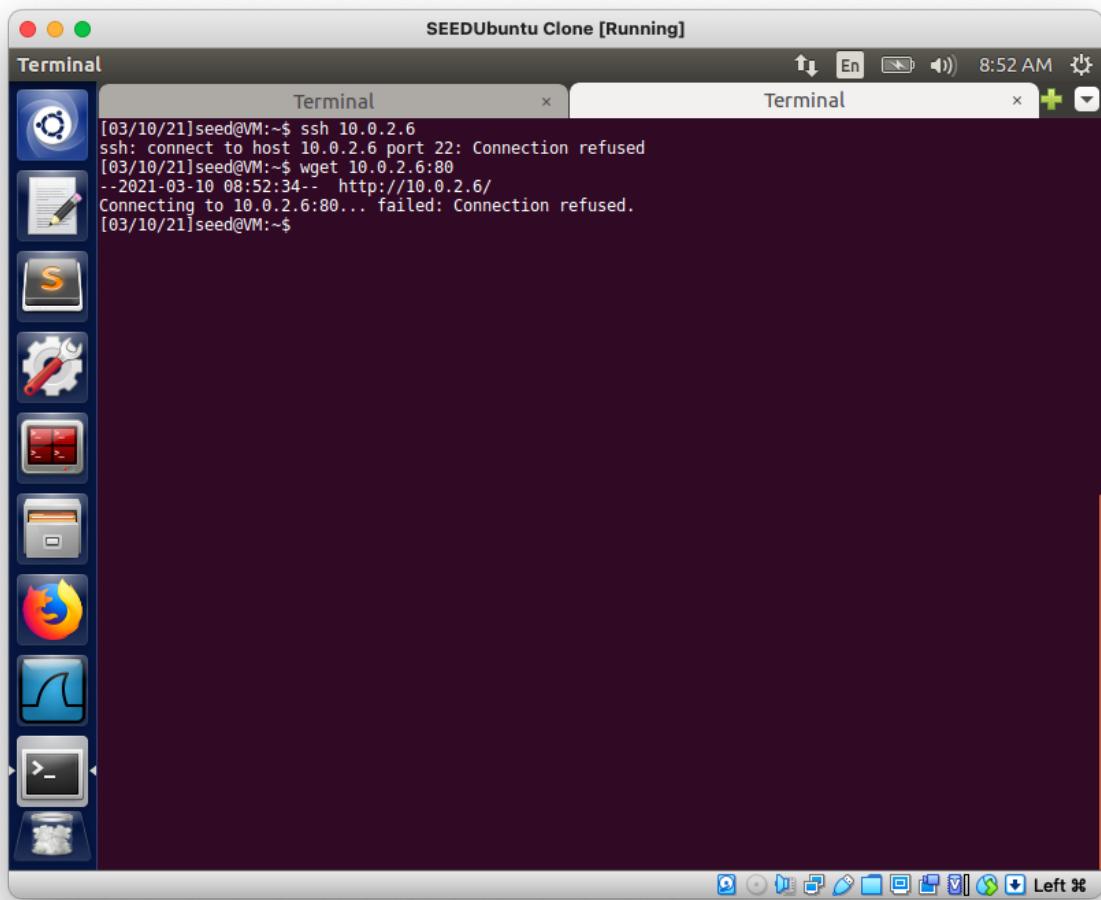


Fig. 4.3 Failure to ssh and wget from Machine B to Machine A

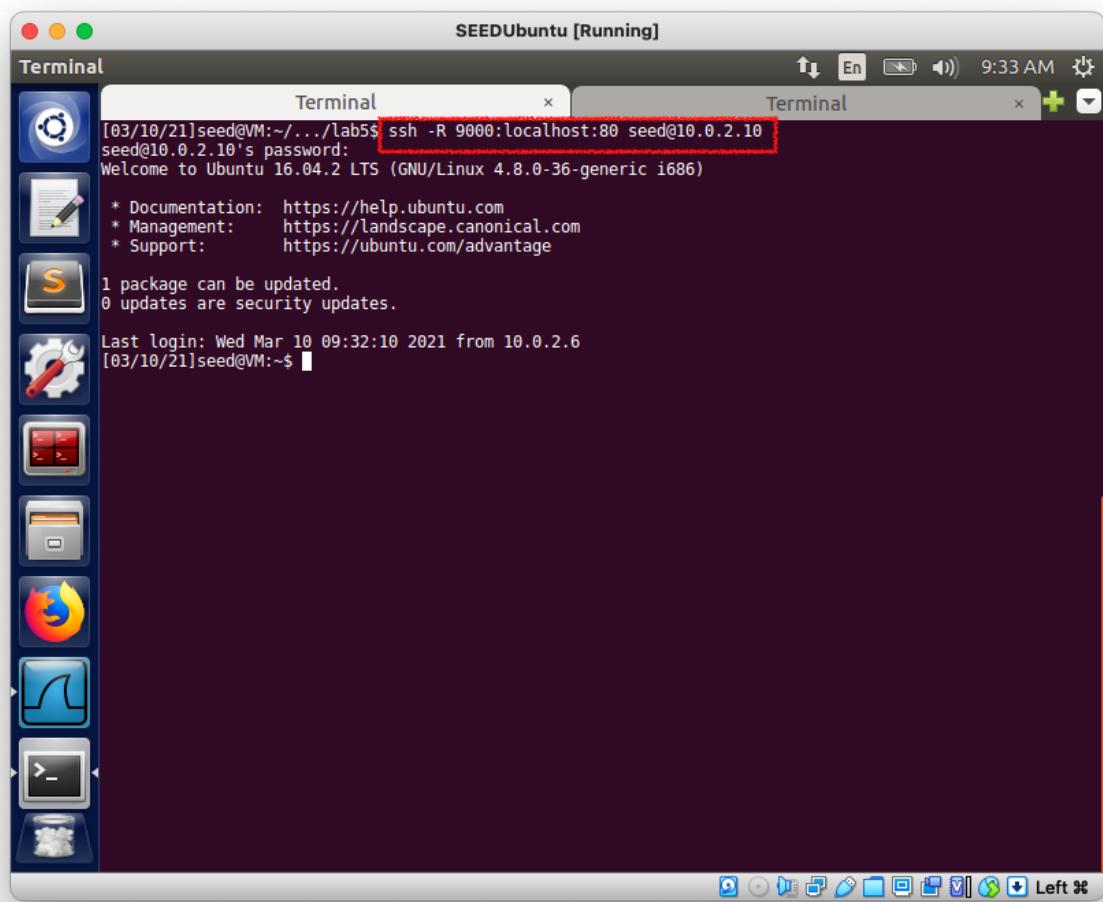


Fig. 4.4 Setting up reverse SSH Tunnel on Machine A

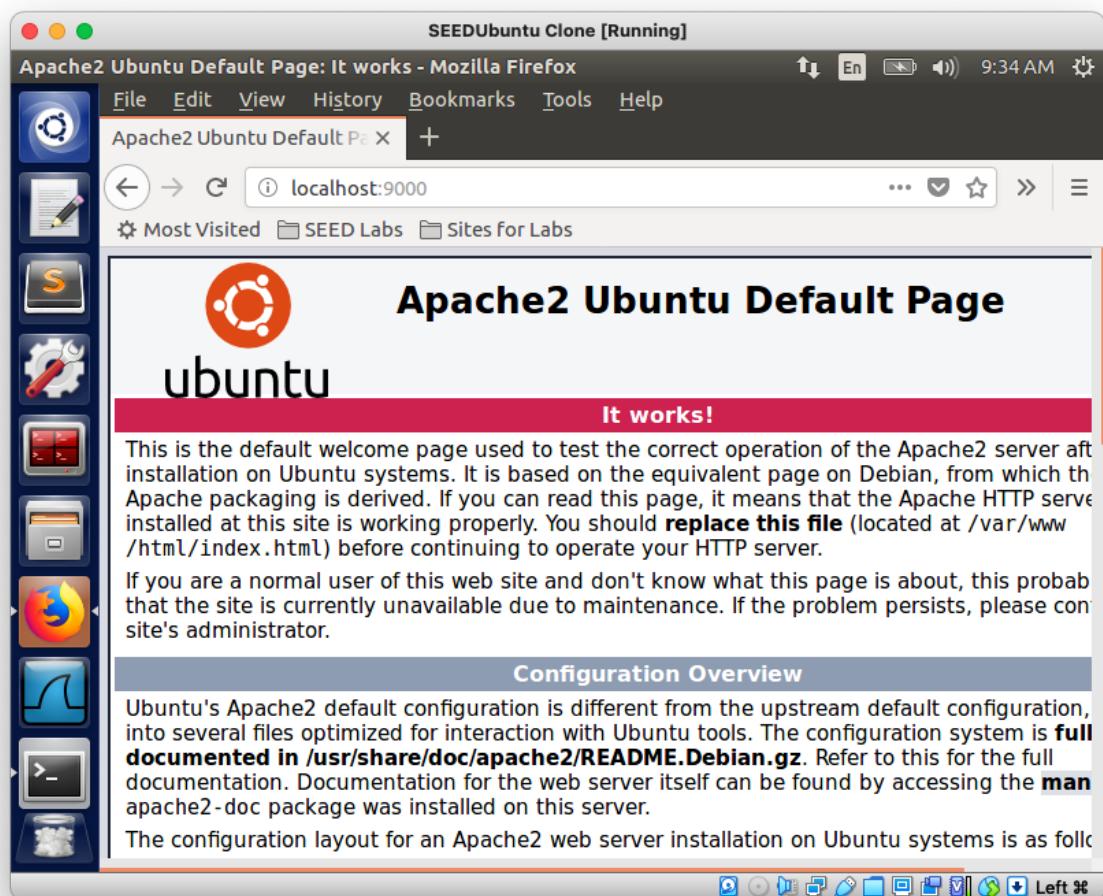


Fig. 4.5 Successful access of Machine A's web server from Machine B via reverse SSH Tunnel