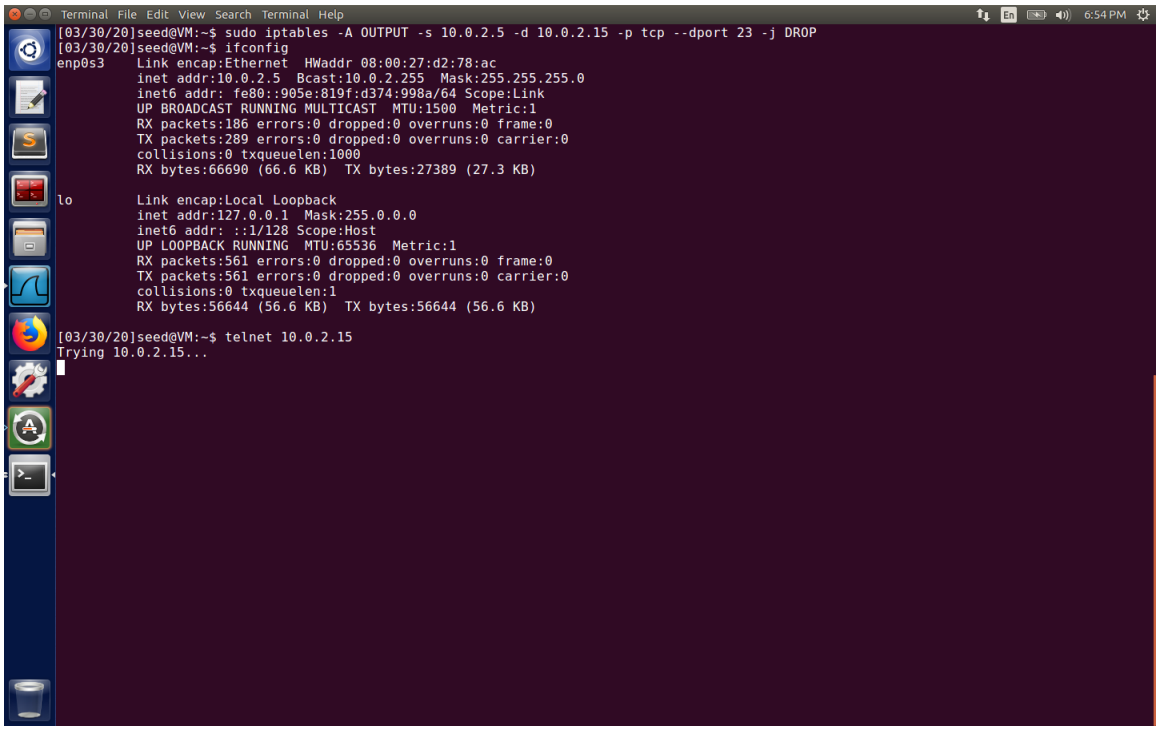
Linux Firewall Exploration Lab

Task 1: Using Firewall on Machine A

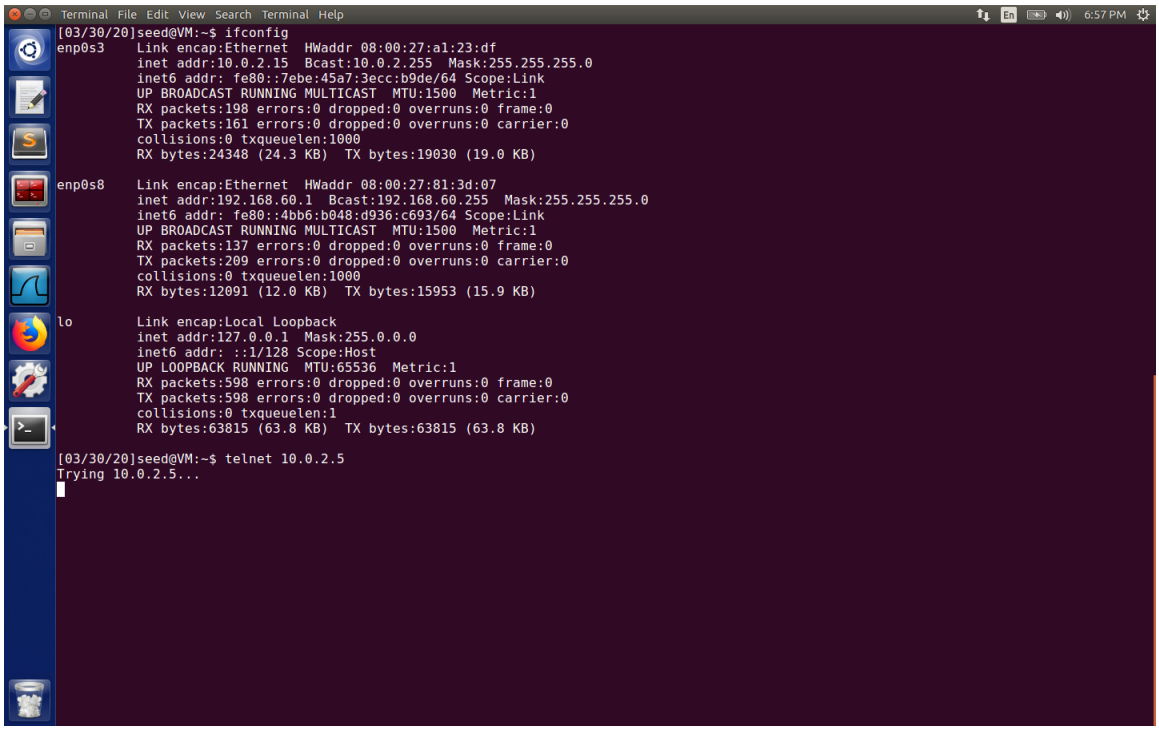
Prevent A from doing telnet to machine B

```
sudo iptables -A OUTPUT -s 10.0.2.5 -d 10.0.2.15 -p tcp –dport 23 -j DROP
```



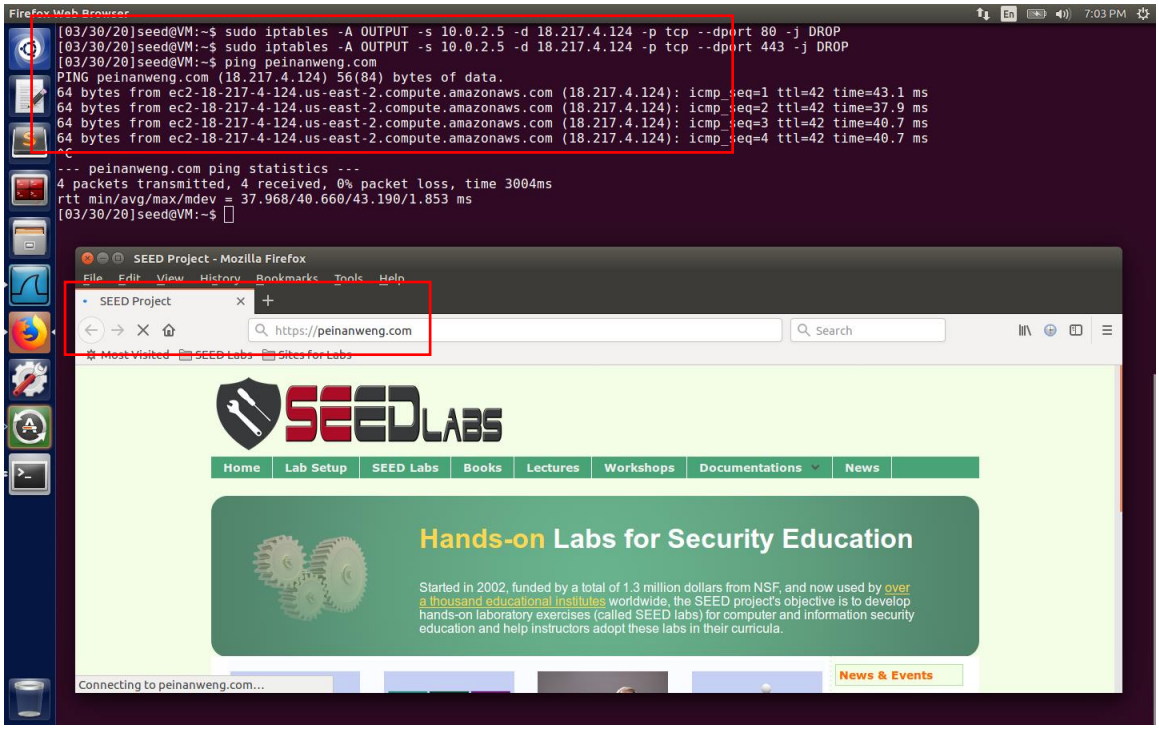
Prevent B from doing telnet to machine A

```
sudo iptables -A INPUT -s 10.0.2.15 -d 10.0.2.5 -p tcp -dport 23 -j DROP
```



Prevent A from visiting an external website (my personal website `peinanweng.com`)

```
sudo iptables -A OUTPUT -s 10.0.2.5 -d 18.217.4.124 -p tcp –dport 80 -j DROP
sudo iptables -A OUTPUT -s 10.0.2.5 -d 18.217.4.124 -p tcp –dport 443 -j DROP
```



Task 2: Implementing a Simple Firewall

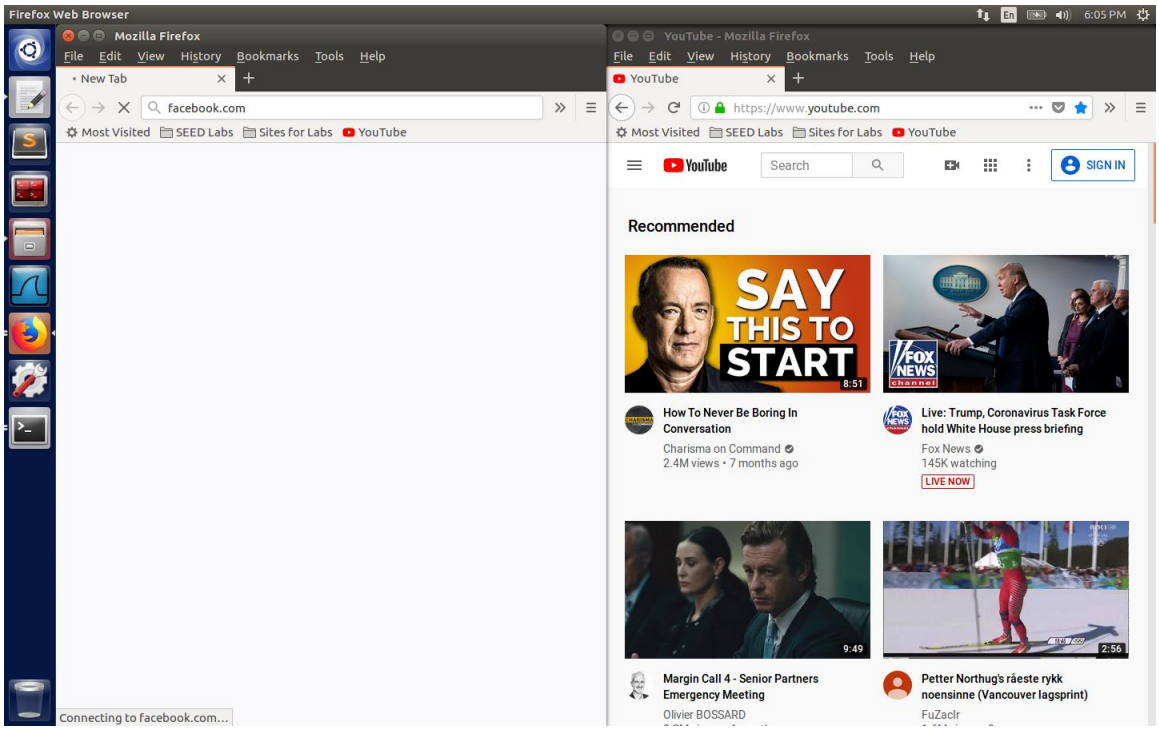
code netfilter.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.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 in_filters_hook;
static struct nf_hook_ops post_routing_filters_hook;

void prn_pkt(char *info, unsigned char* saddr, unsigned char *daddr)
{
      printk(KERN_INFO "%s from %d.%d.%d.%d to %d.%d.%d.%d\n",
            info,
            saddr[0], saddr[1], saddr[2], saddr[3],
            daddr[0], daddr[1], daddr[2], daddr[3]
      );
}

unsigned int in_filter_hook_function(
      void *priv,
      struct sk_buff *skb,
      const struct nf_hook_state *state) {

      struct iphdr *iph;
      struct tcphdr *tcph;
      int host_B_ip = in_aton("10.0.2.15");

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

      // DROP TELNET packets from Machine B to Machine A
      if (iph->protocol == IPPROTO_TCP && ntohs(tcph->dest) == 23 &&
iph->saddr == host_B_ip) {
            prn_pkt("(IN) Dropping TELNET packets:", (unsigned
char*)&iph->saddr, (unsigned char*)&iph->daddr);
            return NF_DROP;
      }
      // DROP SSH packets from Machine B to Machine A
      else if (iph->protocol == IPPROTO_TCP && ntohs(tcph->dest) == 22
&& iph->saddr == host_B_ip) {
            prn_pkt("(IN) Dropping SSH packets:", (unsigned
char*)&iph->saddr, (unsigned char*)&iph->daddr);
            return NF_DROP;
```

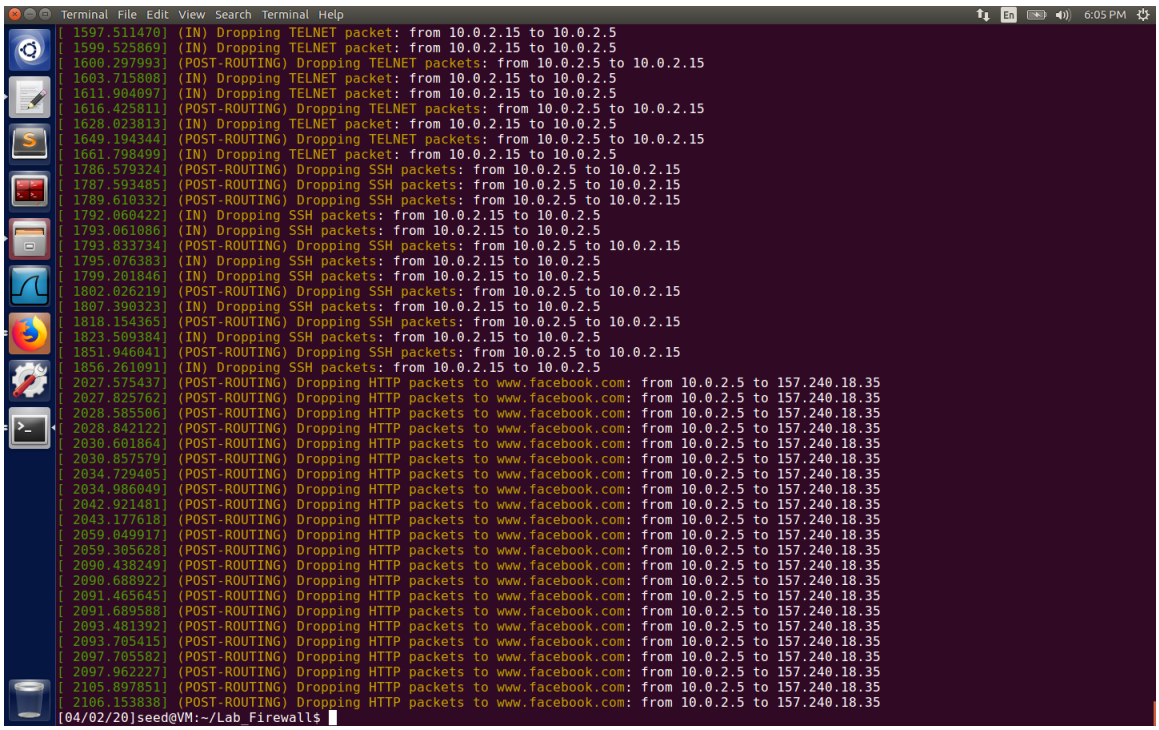
```
	}
	else {
		return NF_ACCEPT;
	}
}

unsigned int post_routing_filter_hook_function(
	void *priv,
	struct sk_buff *skb,
	const struct nf_hook_state *state) {
	struct iphdr *iph;
	struct tcphdr *tcph;
	int host_B_ip = in_aton("10.0.2.15");
	int facebook_ip = in_aton("157.240.18.35");

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

	// DROP TELNET packets from Mahcine A to Machine B
	if (iph->protocol == IPPROTO_TCP && ntohs(tcph->dest) == 23 &&
iph->daddr == host_B_ip) {
		prn_pkt("(POST-ROUTING) Dropping TELNET packets:",
(unsigned char*)&iph->saddr, (unsigned char*)&iph->daddr);
		return NF_DROP;
	}
	// DROP SSH packets from Machine A to Machine B
	else if (iph->protocol == IPPROTO_TCP && ntohs(tcph->dest) == 22
&& iph->daddr == host_B_ip) {
		prn_pkt("(POST-ROUTING) Dropping SSH packets:", (unsigned
char*)&iph->saddr, (unsigned char*)&iph->daddr);
		return NF_DROP;
	}
	// DROP HTTP/HTTPS packets from Machine A to external website
(Facebook)
	else if (iph->protocol == IPPROTO_TCP && (ntohs(tcph->dest) == 80
|| ntohs(tcph->dest) == 443) && iph->daddr == facebook_ip) {
		prn_pkt("(POST-ROUTING) Dropping HTTP/HTTPS packets to
www.facebook.com:", (unsigned char*)&iph->saddr, (unsigned
char*)&iph->daddr);
		return NF_DROP;
	}
	else {
		return NF_ACCEPT;
	}
}

int setUpFilter(void)
{
	printk(KERN_INFO "Registering net filters.\n");
```

```
	in_filters_hook.hook = in_filter_hook_function;
	in_filters_hook.hooknum = NF_INET_LOCAL_IN;
	in_filters_hook.pf = PF_INET;
	in_filters_hook.priority = NF_IP_PRI_FIRST;

	post_routing_filters_hook.hook =
post_routing_filter_hook_function;
	post_routing_filters_hook.hooknum = NF_INET_POST_ROUTING;
	post_routing_filters_hook.pf = PF_INET;
	post_routing_filters_hook.priority = NF_IP_PRI_FIRST;

	nf_register_hook(&in_filters_hook);
	nf_register_hook(&post_routing_filters_hook);

	return 0;
}

void removeFilter(void)
{
	printk(KERN_INFO "Net filters are being removed.\n");
	nf_unregister_hook(&in_filters_hook);
	nf_unregister_hook(&post_routing_filters_hook);
}

module_init(setUpFilter);
module_exit(removeFilter);

MODULE_LICENSE("GPL");

/* END */
```

The result of blocking Telnet and SSH is the same as what we have done in task 1 (Keep trying, then time-out). Here I will show the result of blocking external website.



We can get access to all external websites except `facebook.com` because of our firewall.



Kernel messages: Our packets were blocked by the firewall.

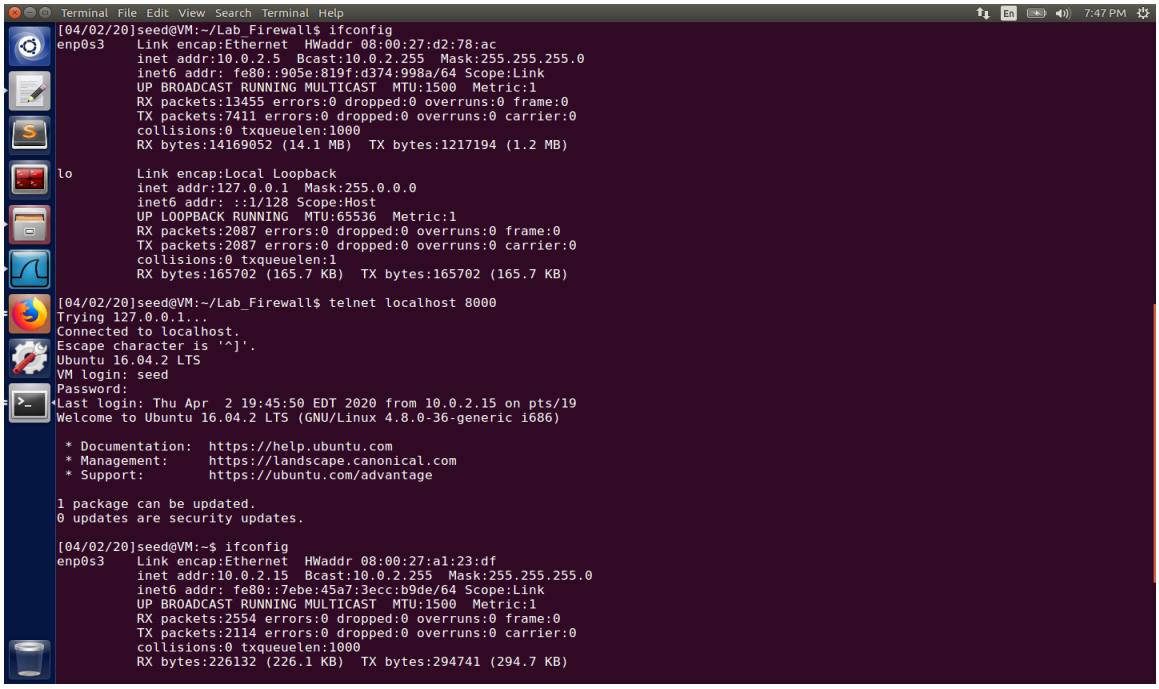
Task 3: Evading Egress Filtering

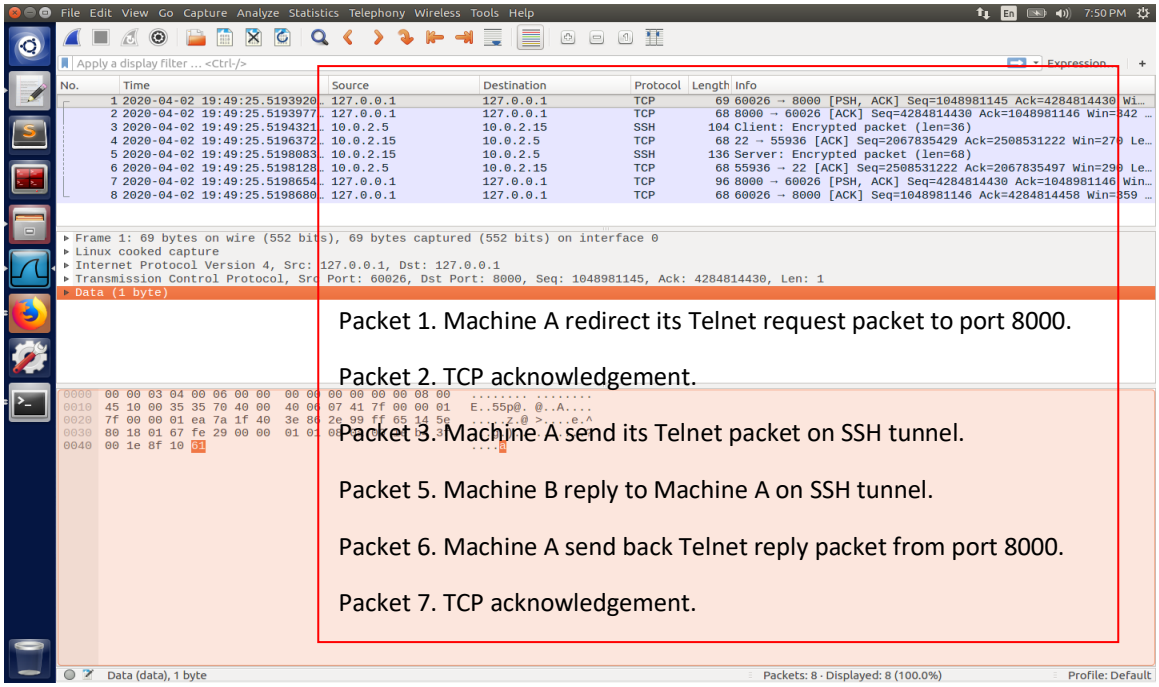
In Task 2, we have already block Telnet packets to our Machine B and HTTP/HTTPS packets to `facebook.com`.

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

`command: ssh -L 8000:10.0.2.15:23 seed@10.0.2.15`

Bind Machine A 8000 port to Machine B 23 port.



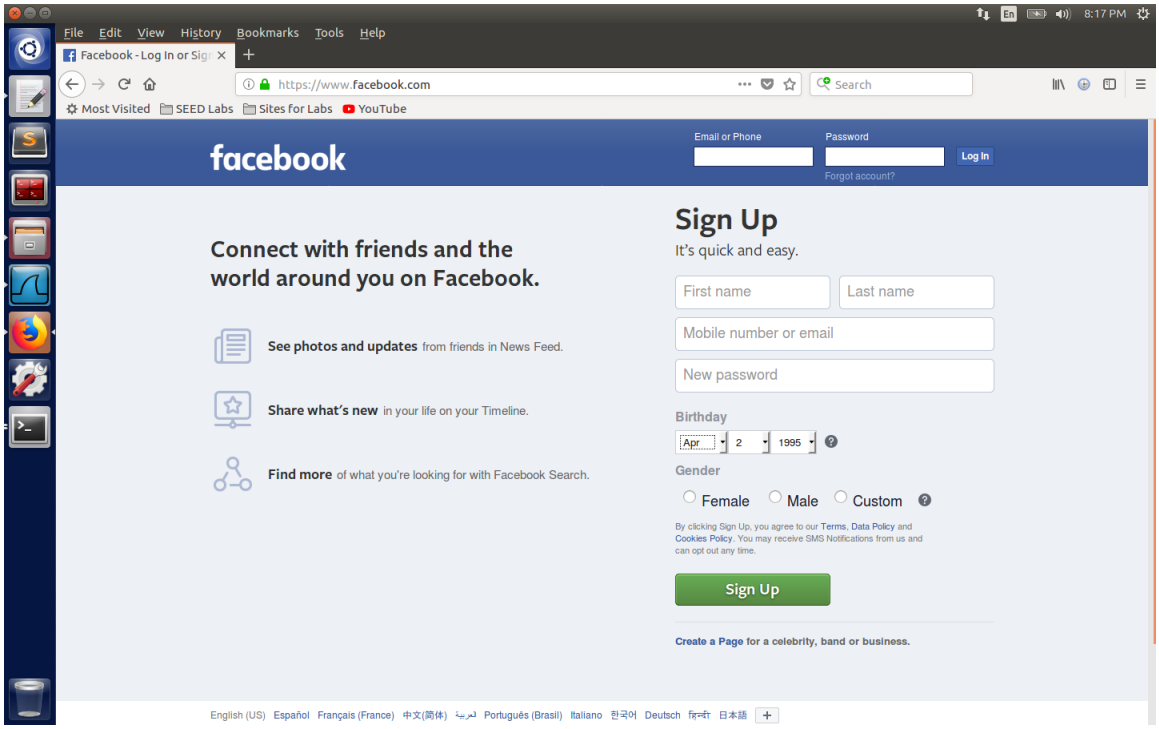


Task 3.b: Connect to Facebook using SSH Tunnel

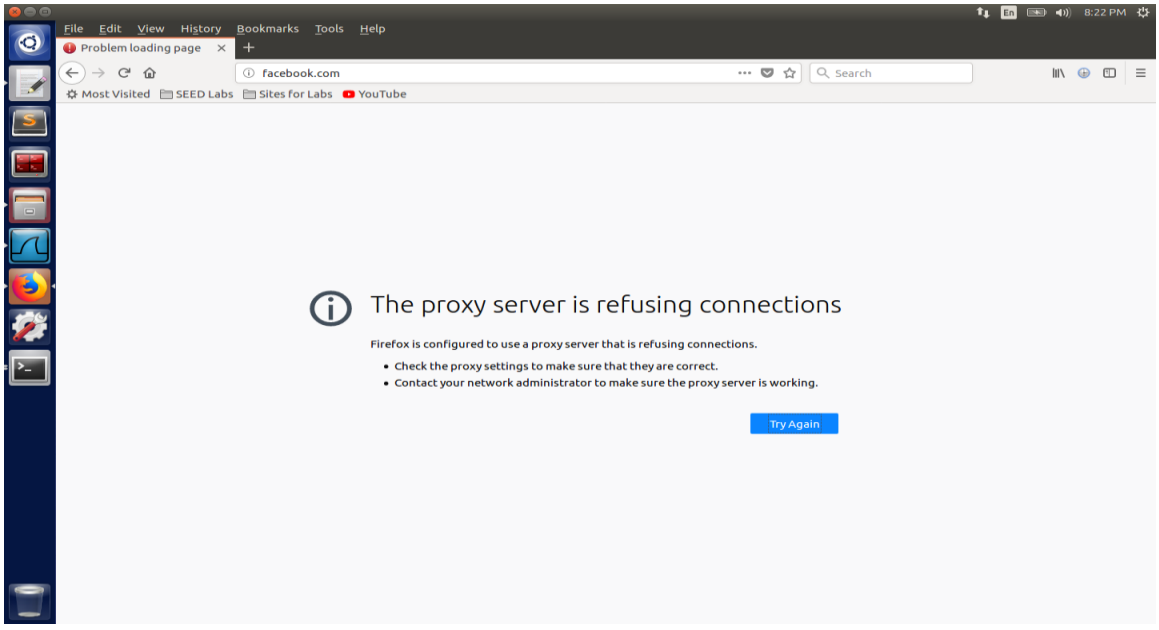
`command: ssh -D 9000 -C seed@10.0.2.15`

Bind Machine A 9000 port to Machine B. Then set proxy server for Firefox.

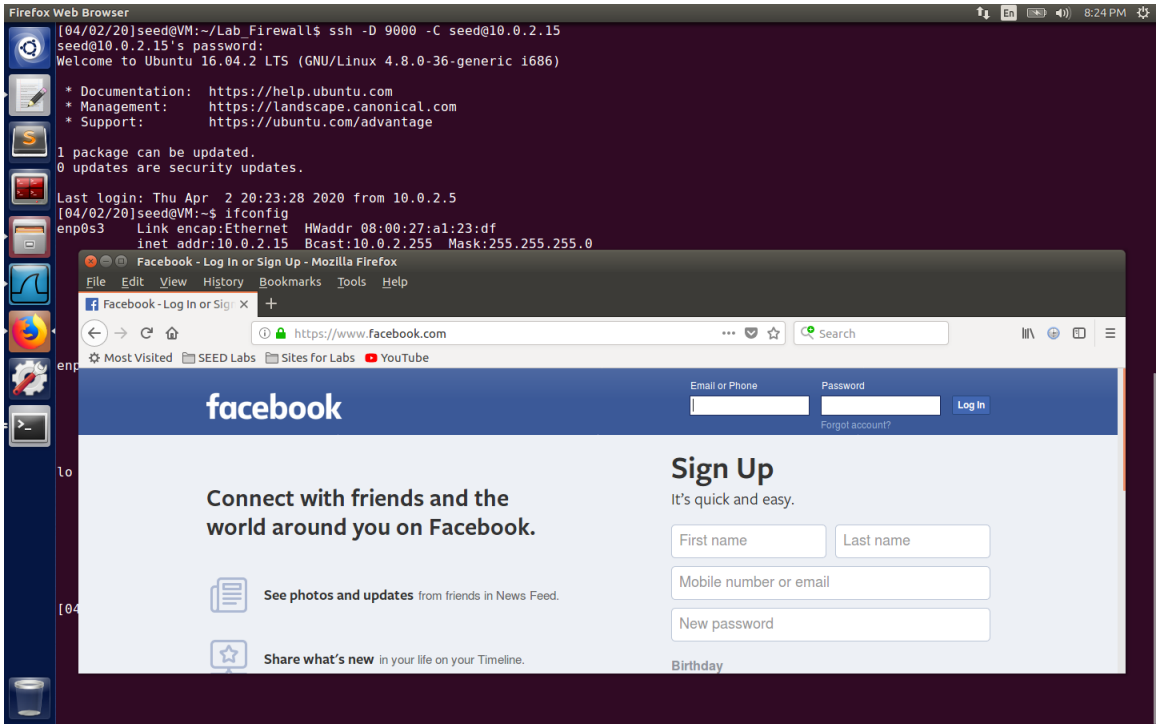
Then we could get access to `facebook.com`.



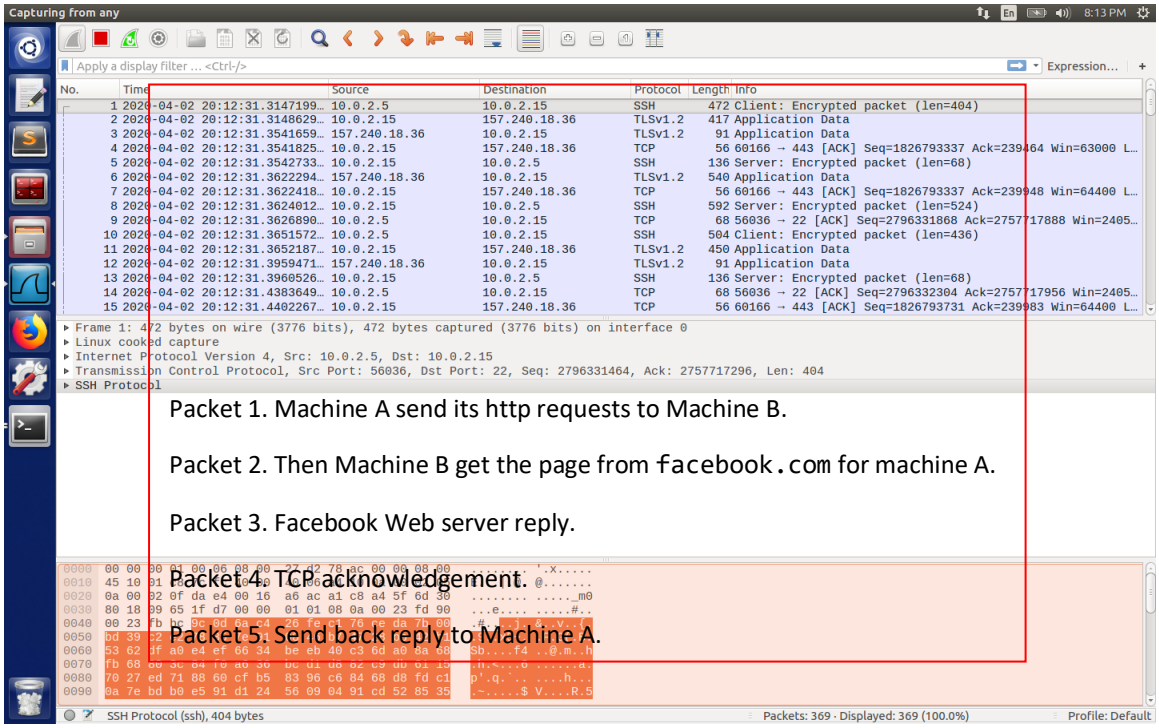
If we break the SSH tunnel, we cannot get access to `facebook.com` (and also other websites), because our proxy server is no longer available.



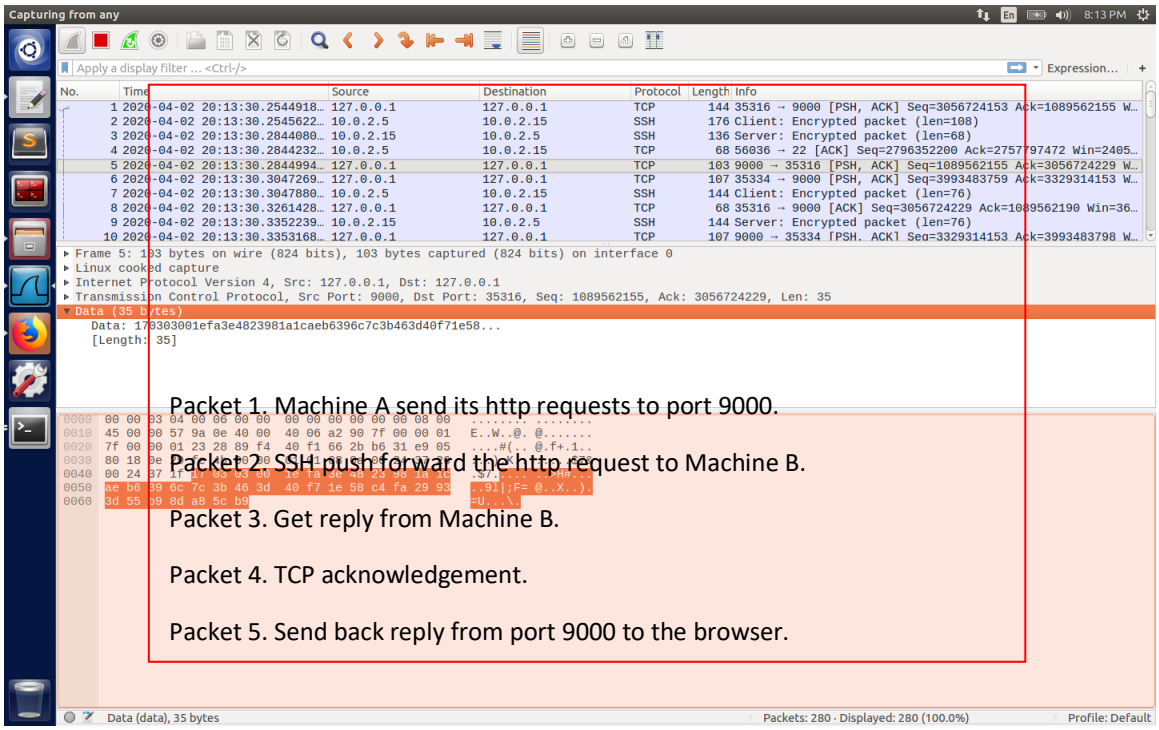
After we established the tunnel again, we can get access to `facebook.com` again.



Wireshark – Machine B side



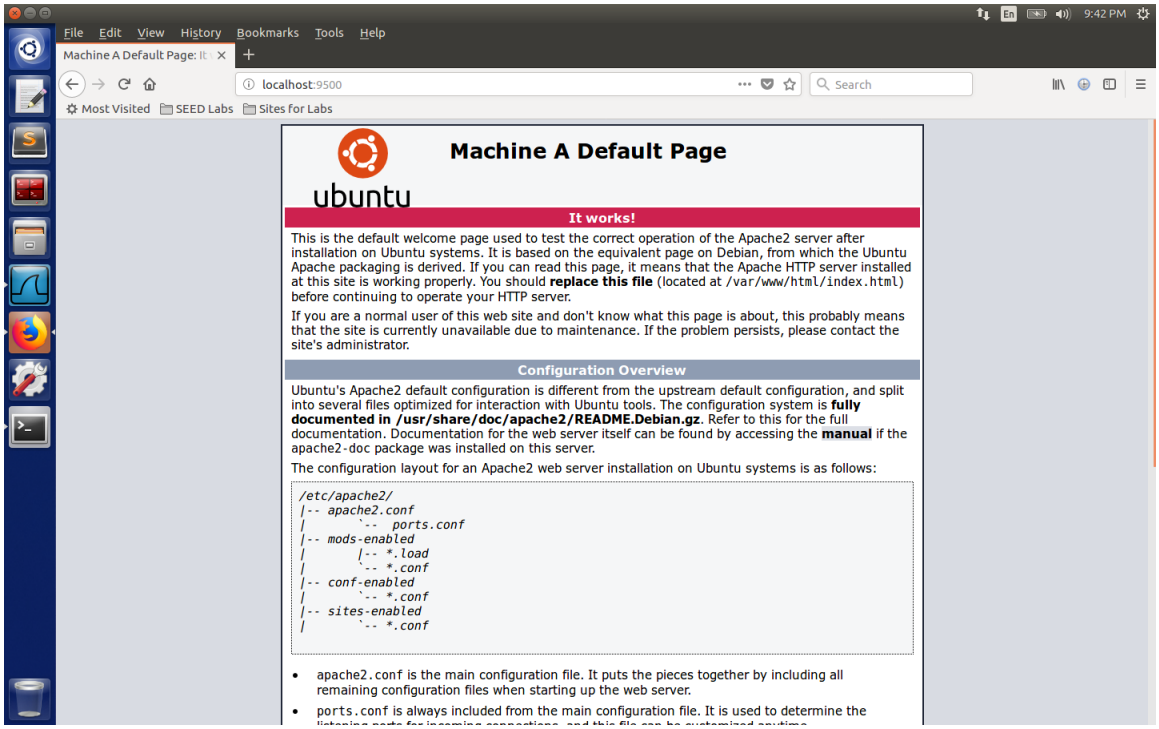
Wireshark – Machine A side



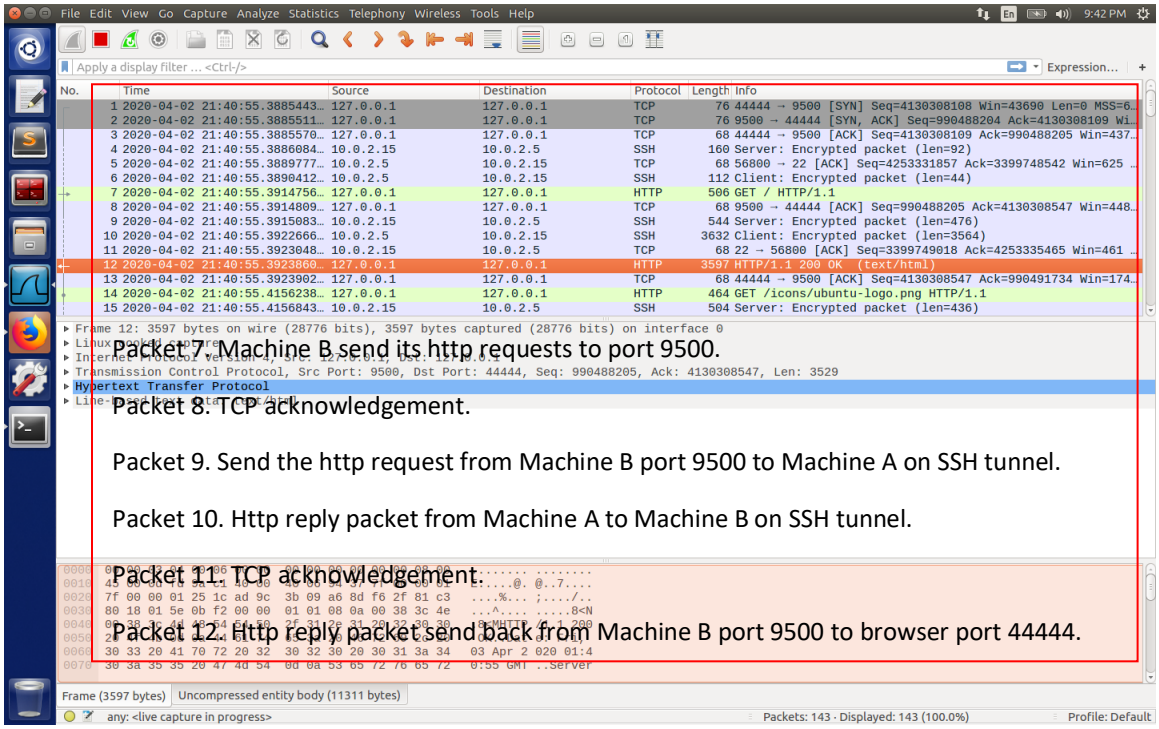
Task 4: Evading Ingress Filtering

`command: ssh -R 9500:10.0.2.5:80 seed@10.0.2.15`

Remote Bind: Bind Machine B 9500 port to Machine A 80 port.



We can see the page on Machine A by searching `localhost:9500` on Machine B.



Packet 7. Machine B send its http requests to port 9500.

Packet 8. TCP acknowledgement.

Packet 9. Send the http request from Machine B port 9500 to Machine A on SSH tunnel.

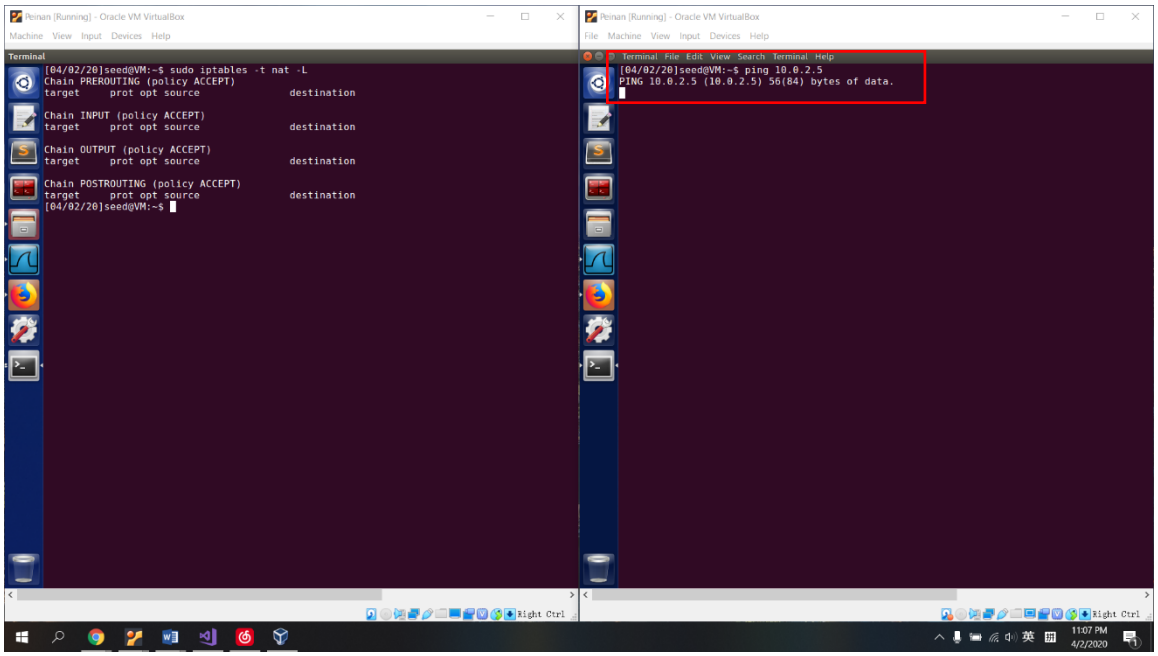
Packet 10. Http reply packet from Machine A to Machine B on SSH tunnel.

Packet 11. TCP acknowledgement.

Packet 12. Http reply packet send back from Machine B port 9500 to browser port 44444.

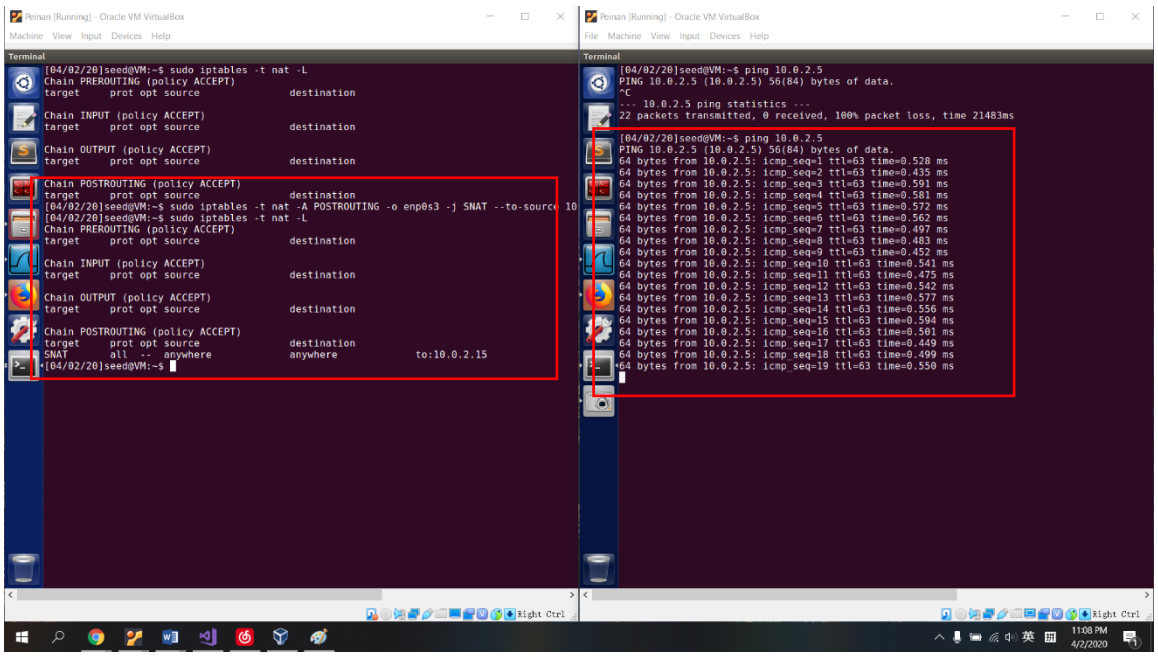
Task 5(SNAT): Use iptables to set up a SNAT. Use Wireshark to prove that your SNAT is working.

Before: Host V cannot get ICMP reply from Host U.

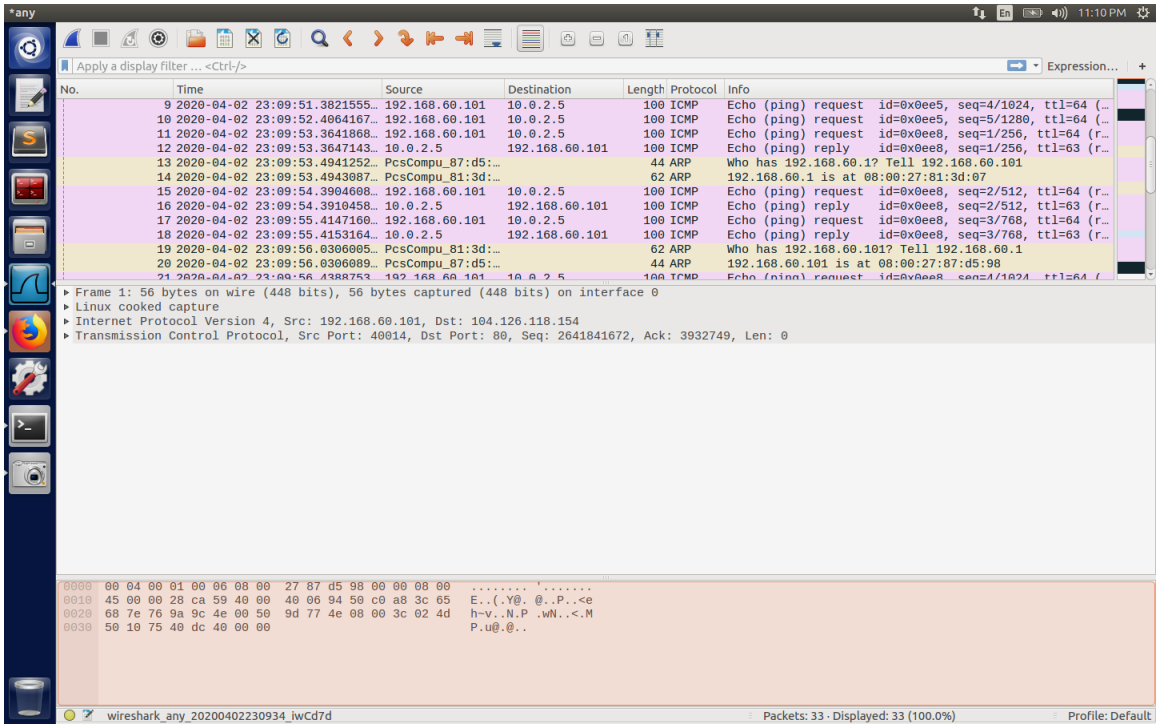


After we set the SNAT on our server machine (Both Internal and NAT Network).

Host V can get ICMP reply from Host U.



Wireshark – Host V

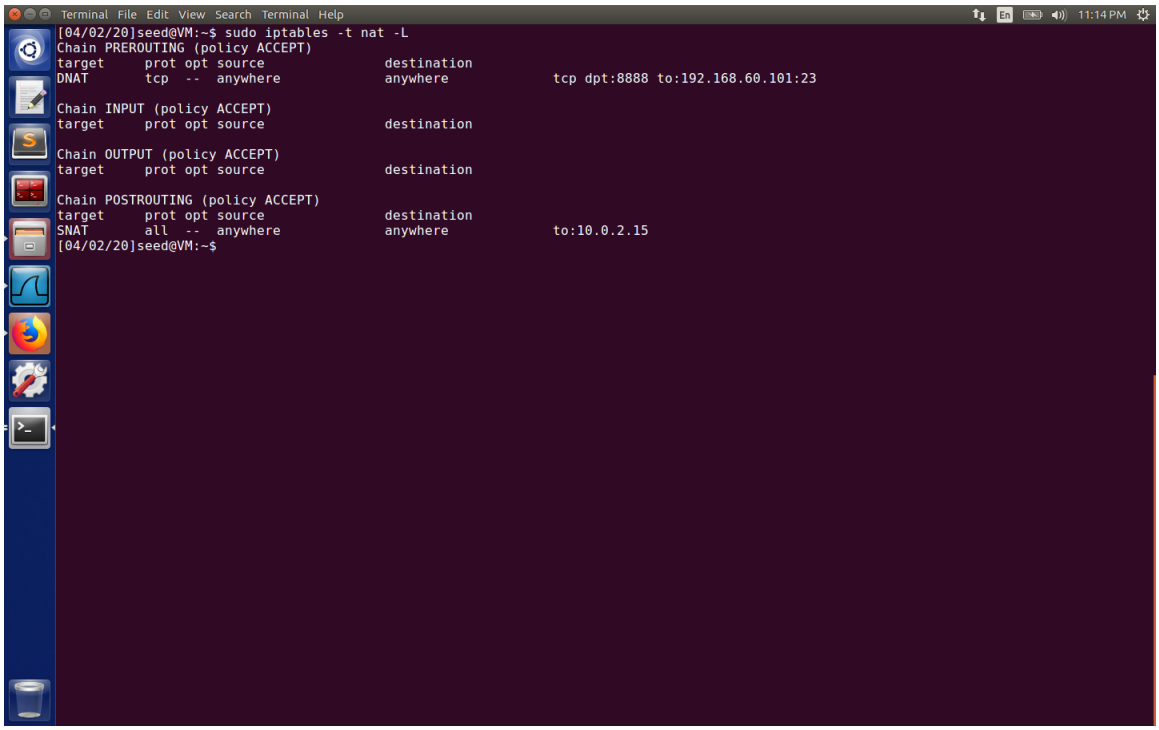


Packet 9 and Packet 10: before setting SNAT.

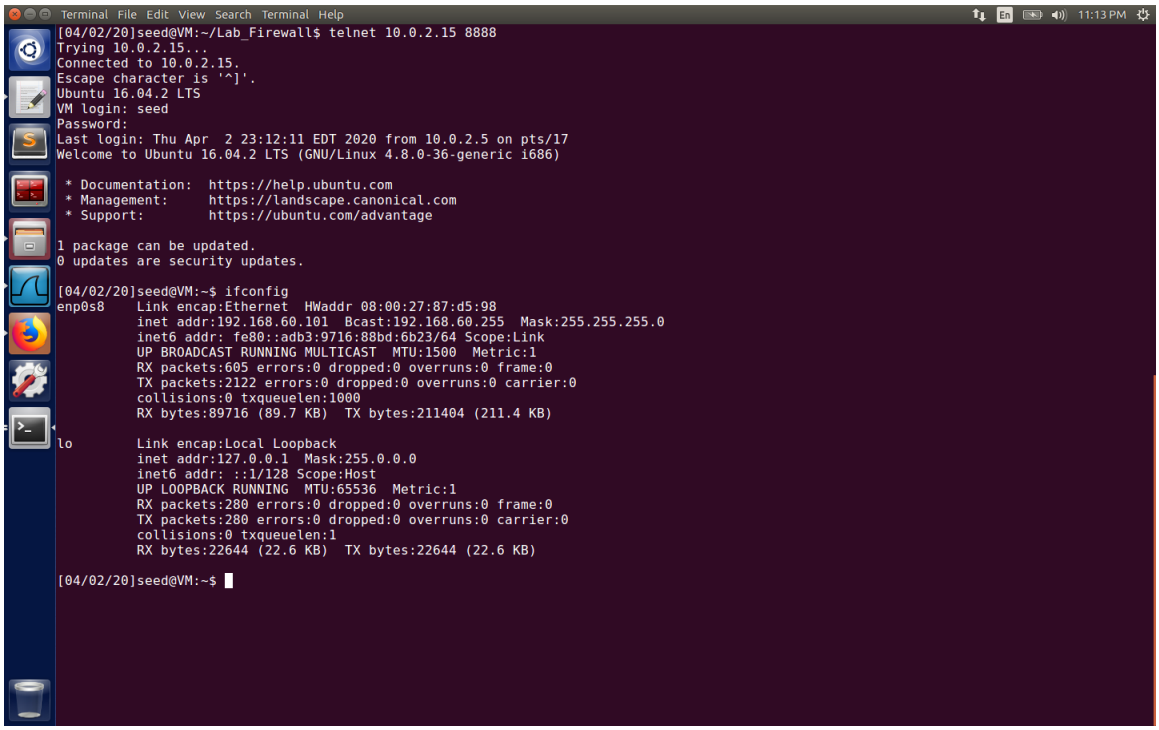
Packet after 11: After setting SNAT.

Task 6(DNAT): Use iptables to set up a DNAT for port forwarding. Use Wireshark to prove that your DNAT is working.

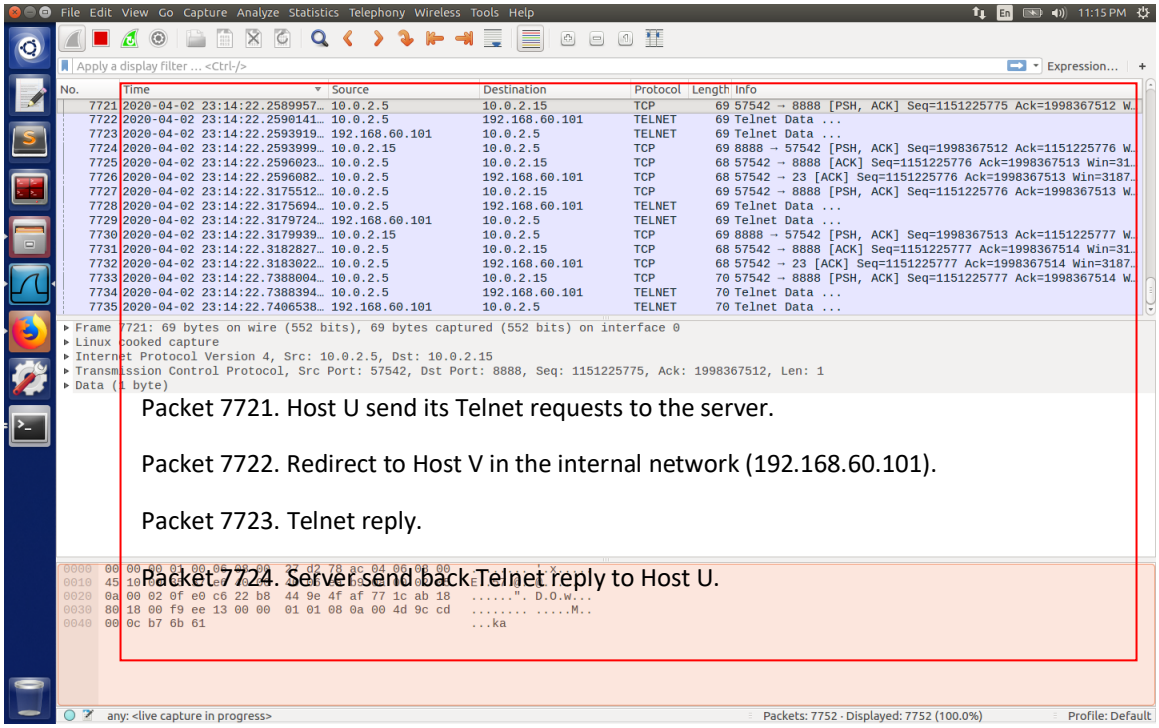
Set DNAT on server. See the following **`iptables`**.



Telnet 10.0.2.15, but actually connected to 192.168.60.101 in the internal network.



Wireshark: Telnet Packets



Packet 7721. Host U send its Telnet requests to the server.

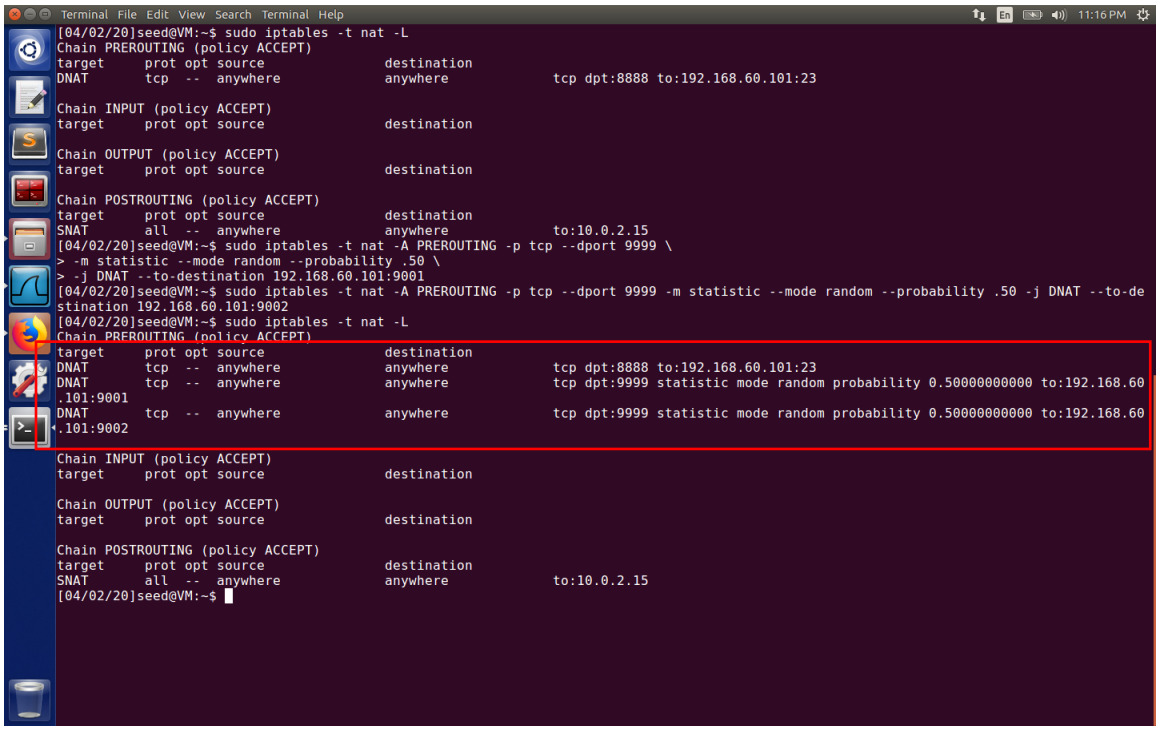
Packet 7722. Redirect to Host V in the internal network (192.168.60.101).

Packet 7723. Telnet reply.

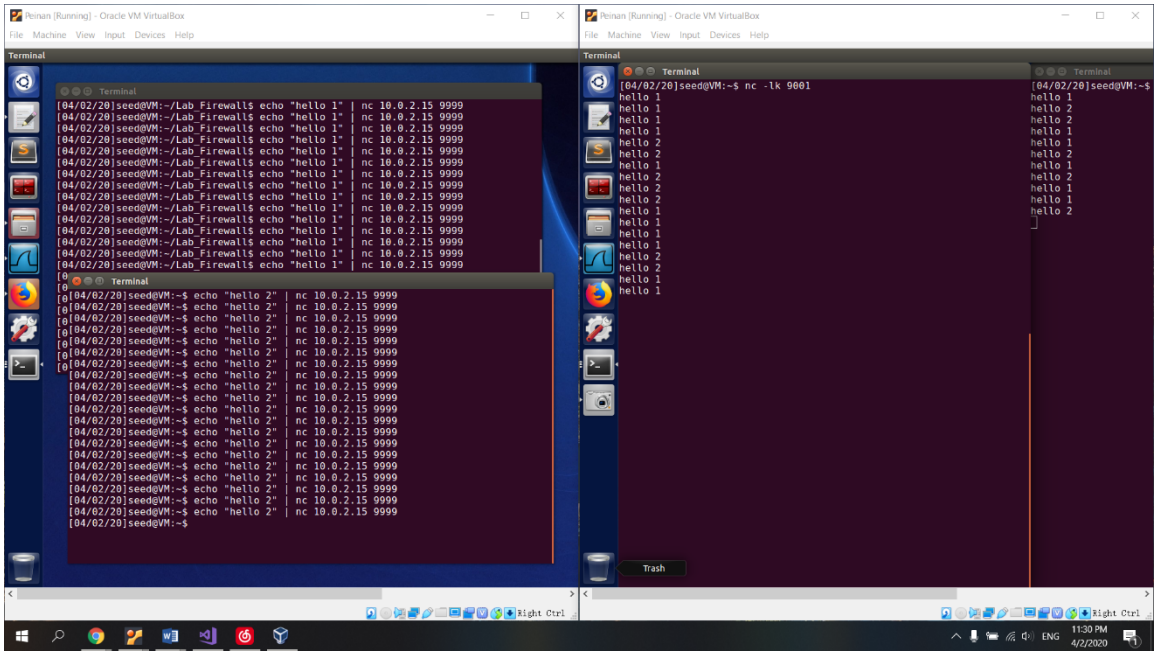
Packet 7724. Server send back Telnet reply to Host U.

Task 7(DNAT): Use iptables to set up a DNAT for load balancing and demonstrate how it works. In my lecture, I didn't get a perfect load balancing. Can you improve my result?

Set rules on server.



Results



How to improve: set the first probability to 0.5 and the second one 1.

If K > 2, then let the probability of N-th server be `[1 / (K - N + 1)]`

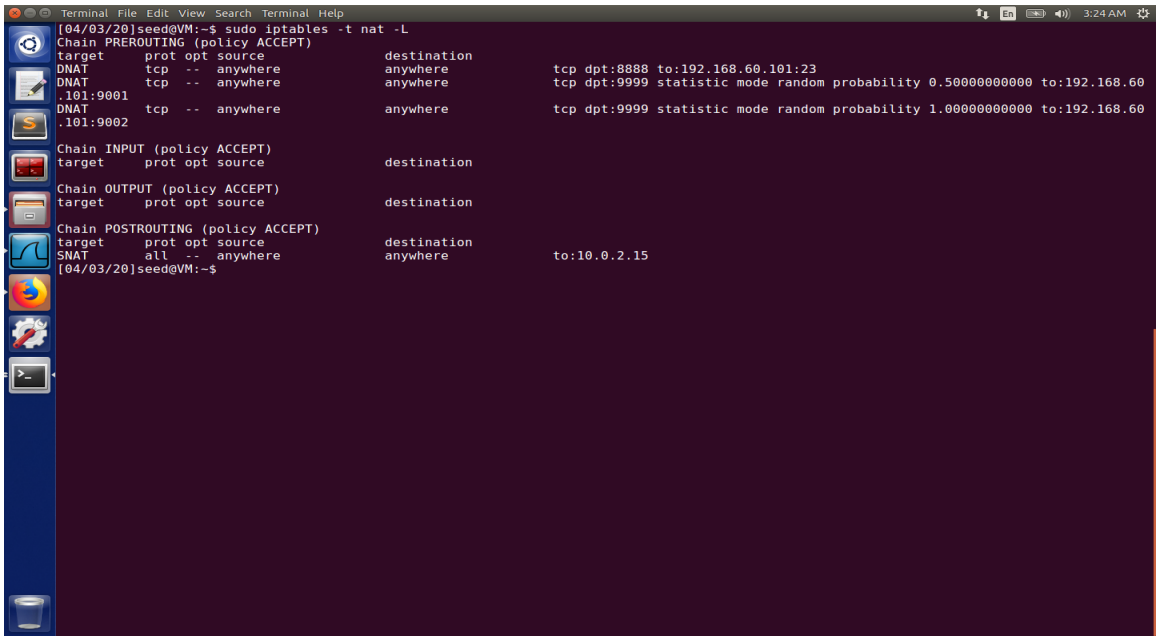
For example, we have 4 servers.

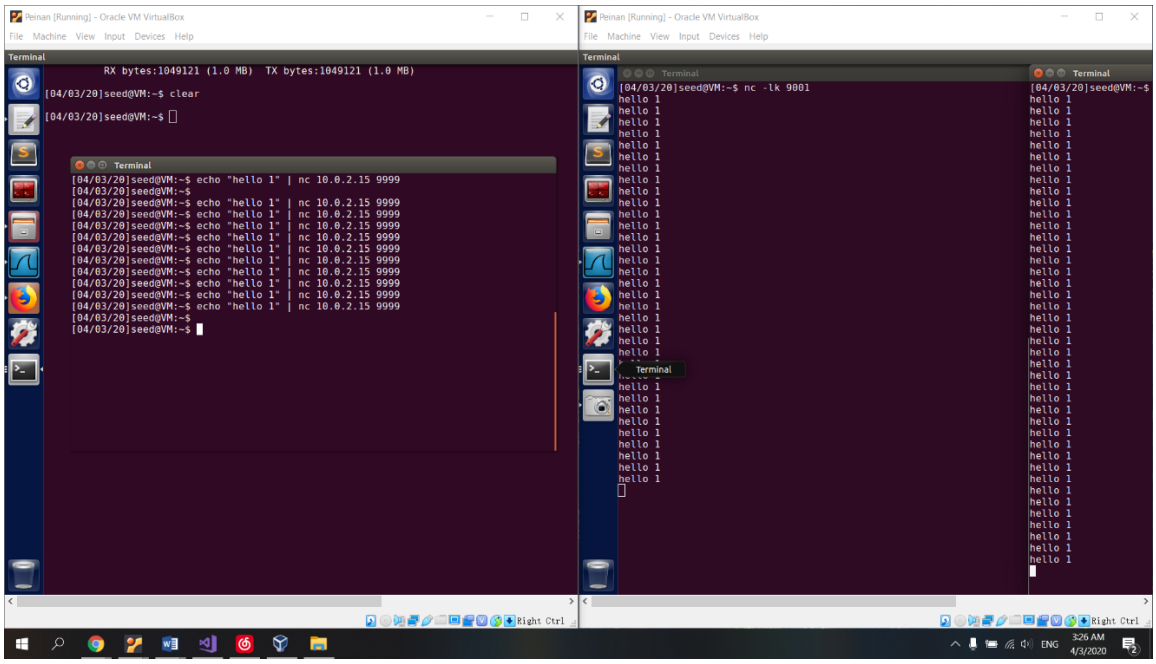
Server 1 Probability = 1 / 4 = 0.25

Server 2 Probability = 1 / 3 = 0.33

Server 3 Probability = 1 / 2 = 0.5

Server 4 Probability = 1 / 1 = 1





Task 8(Connection Track): Set up a firewall rule based on connections.

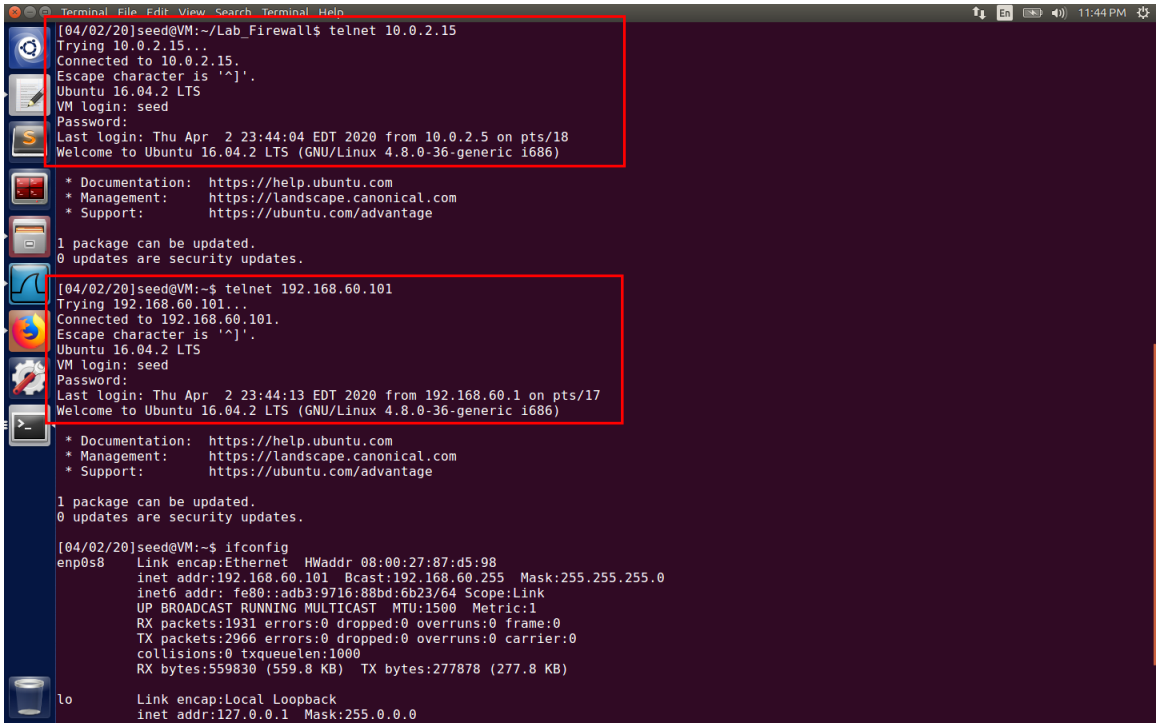
```
Command 1: sudo iptables -A OUTPUT -p tcp -m conntrack –ctstate
ESTABLISHED,RELATED -j ACCEPT
```

```
Command 2: sudo iptables -A OUTPUT -p tcp -j REJECT
```

Only allow TCP packets with ESTABLISHED and RELATED state, no new TCP connections.

See results & screenshots on the next page.

Before: Host U could telnet 10.0.2.15 then telnet 192.168.60.101(host V) in the internal network.



After: Can't do that, because the firewall disallow new TCP connections.