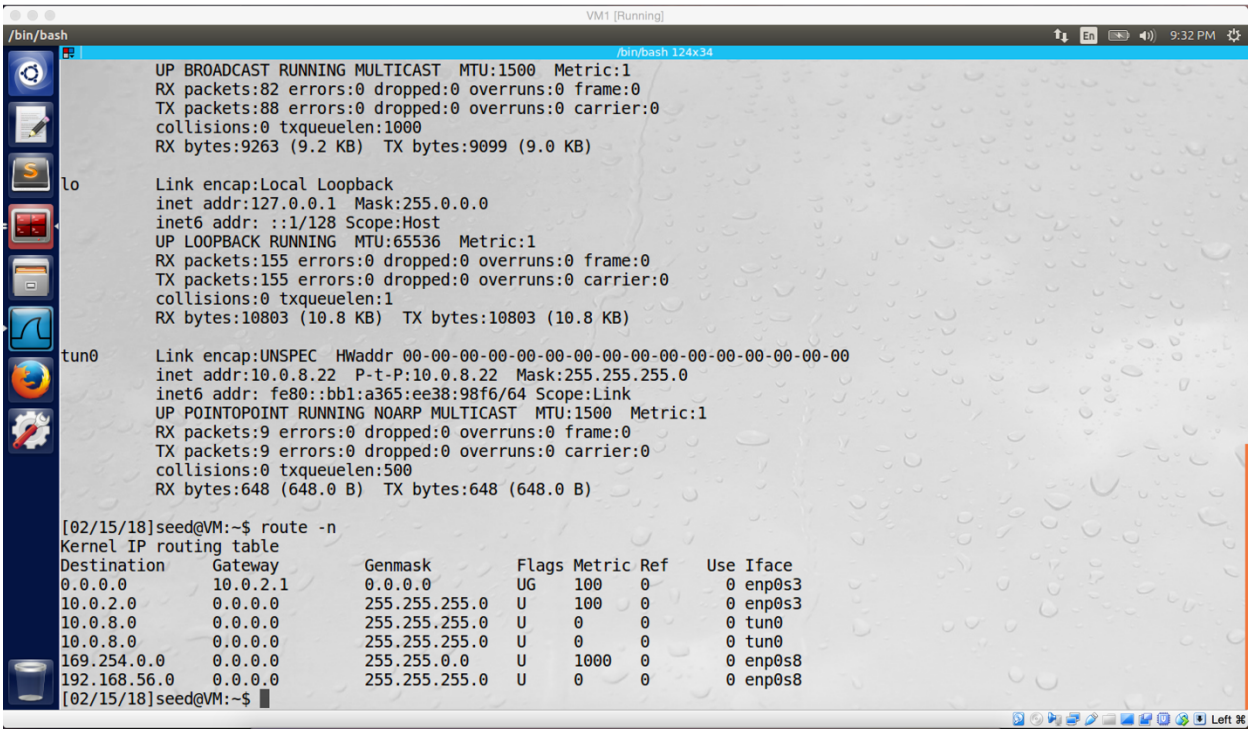
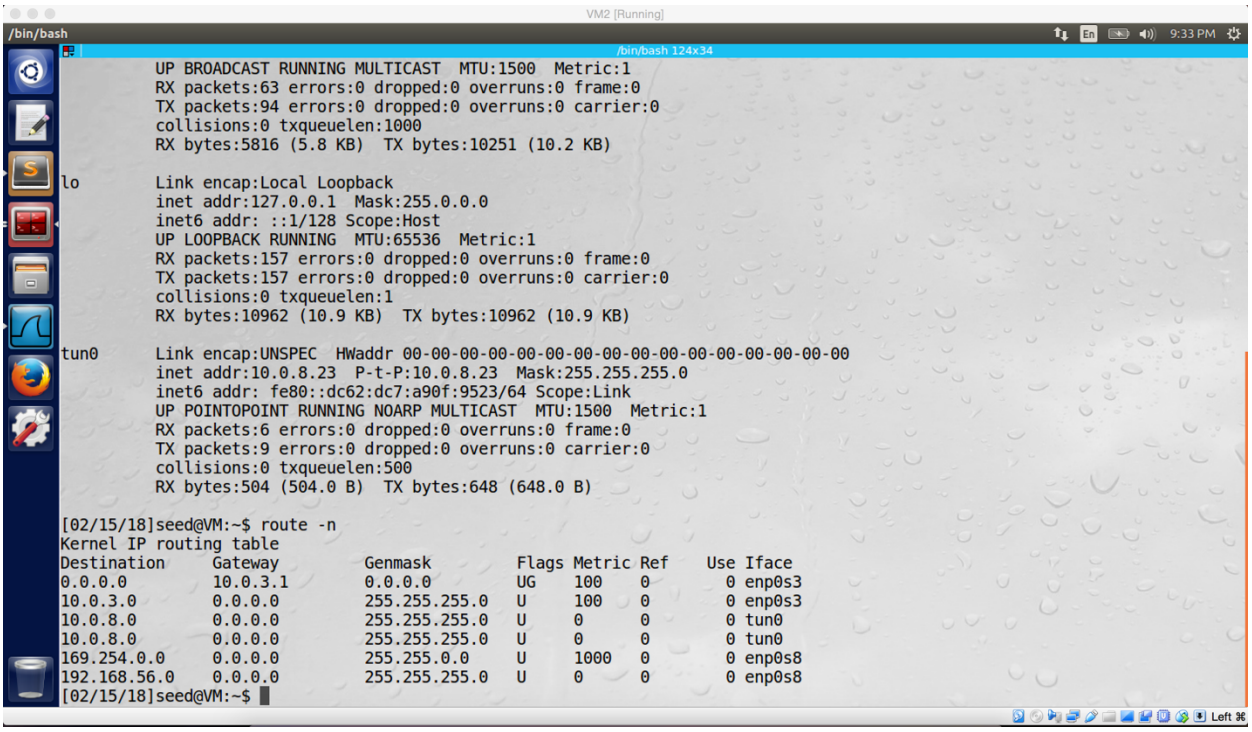
CSE644 Lab3
Yishi Lu
2/20/2018

In this task, I use three machines, they are VM1 (VM A), VM2(VM B), and VM3(VM C).
VM1 (VM A) running VPN server program, IP address is 10.0.2.20, tun0 IP address is 10.0.8.22, host-only IP address is 192.168.56.101
VM2 (VM B) running VPN client program, IP address is 10.0.3.10, tun0 IP address is 10.0.8.23, host-only IP address is 192.168.56.102 (in the program, we set SERVER_IP “192.168.56.101”)
VM3 (VM C) running VPN client program, IP address is 10.0.2.25, tun0 IP address is 10.0.8.24 (in the program, we set SERVER_IP “10.0.2.20”)

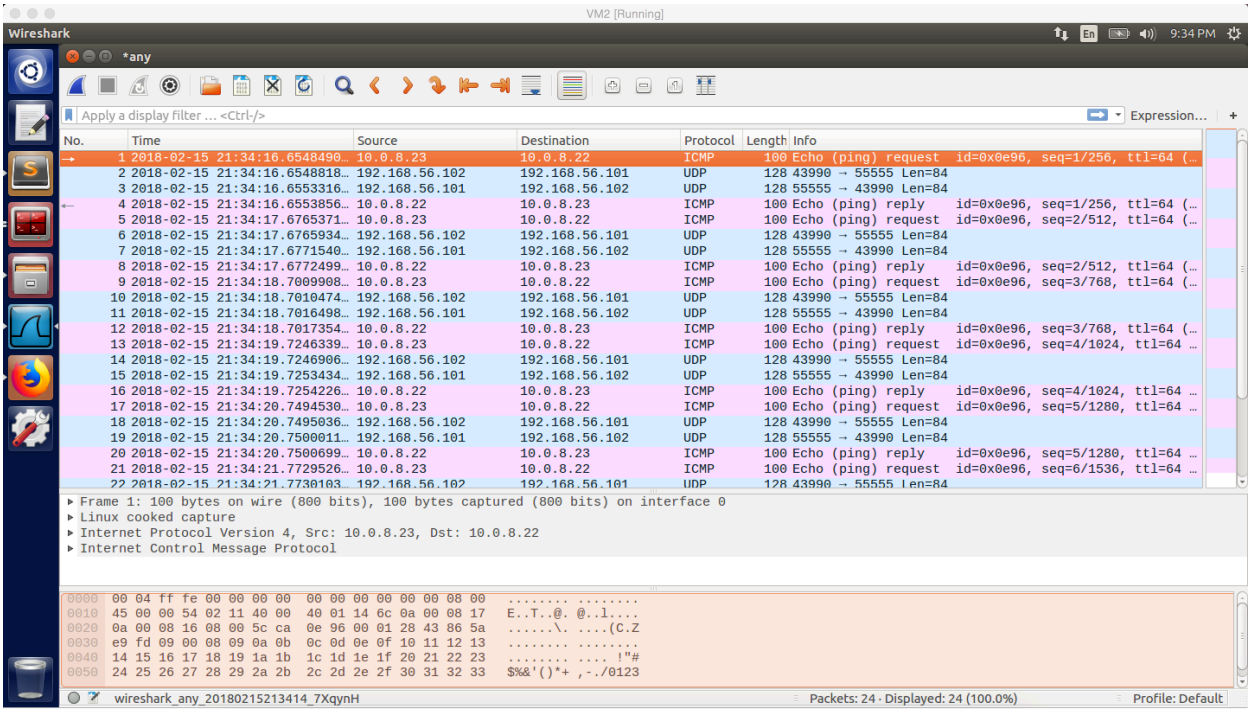
Task 1



screenshot 1 on VM A, successfully register TUN interface, and assigned IP address 10.0.8.22 to tun0



screenshot 2 on VM B, successfully register TUN interface, and assigned IP address 10.0.8.23 to tun0.



screenshot 3 on VM B. Tunnel established successfully between VM A and VM B.

Observation and Explanation:

In this task, I setup environment of two VM as the lab description requirements. VM A and VM B are running in the same host-only network (IP address 192.168.56.0). And they also have different NatNetwork. For VM A, it's 10.0.2.20. For VM B, it's 10.0.3.10.

After the environment setup, I run VPN server program on VM A and VPN client program on VM B (with SERVER_IP = "192.168.56.101"). And then we need to configure the TUN interface by command "sudo ifconfig tun0 10.0.8.X/24 up". This command did three tasks. First, it specified what network the interface is connected to; second, it assigned IP address to the interface (For VM A, I assigned 10.0.8.22; for VM B, I assigned 10.0.8.23); and finally, it activated the interface. Afterwards, we need to add rule to routing table to make sure that packets with destination IP address of 10.0.8.0/24 will go to the TUN interface, so we use command "sudo route add –net 10.0.8.0/24 tun0" on both VMs. And then we also need to turn on IP forwarding on VM A by command "sudo sysctl –w net.ipv4.ip_forward=1". If we do not turn on IP forwarding, VM A will simply drop packets if these packet are not for it. After we turn on this mode, the machine will work as a router and forward other machine's packets.

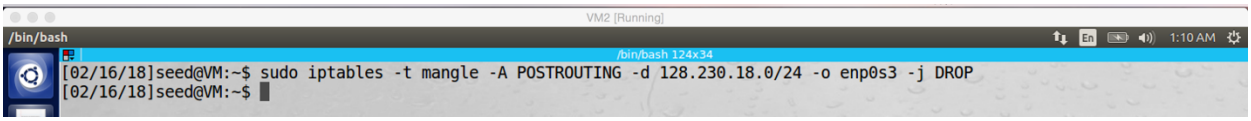
After I finished setup, I try to ping VM A from VM B by using TUN interface. As screenshot 3 shows, it works. The first ICMP packet is sent out from 10.0.8.23 (VM B tun0), and it successfully received by VM A TUN interface 10.0.8.22. We also can see there are UDP packets with IP address of 192.168.56.101 and 192.168.56.102. These two IP addresses represent two endpoints of the VPN tunnel, which are VM A and VM B. Actually, the ICMP packet is pass through the VPN tunnel from VM B to VM A; after VM A receive it, it sends ICMP reply back in the same path but reverse order. We will talk more detail in the task4.

Task2

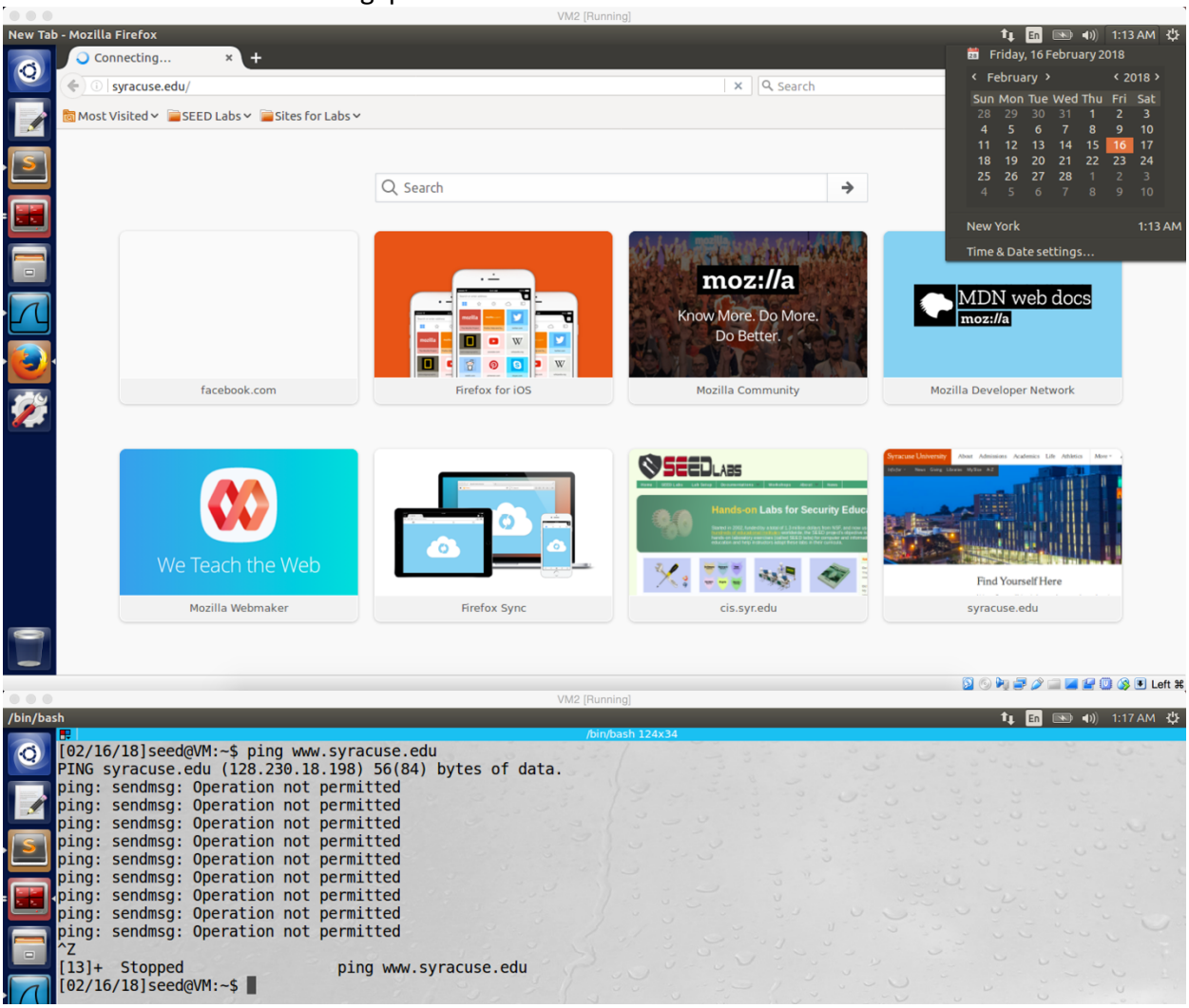
This task will be done in the **Task4**

Task3

In this task, I choose to block SU website which is www.syracuse.com, and it has IP address 128.230.18.198



screenshot 1 on VM B. Adding iptable rule to block SU website



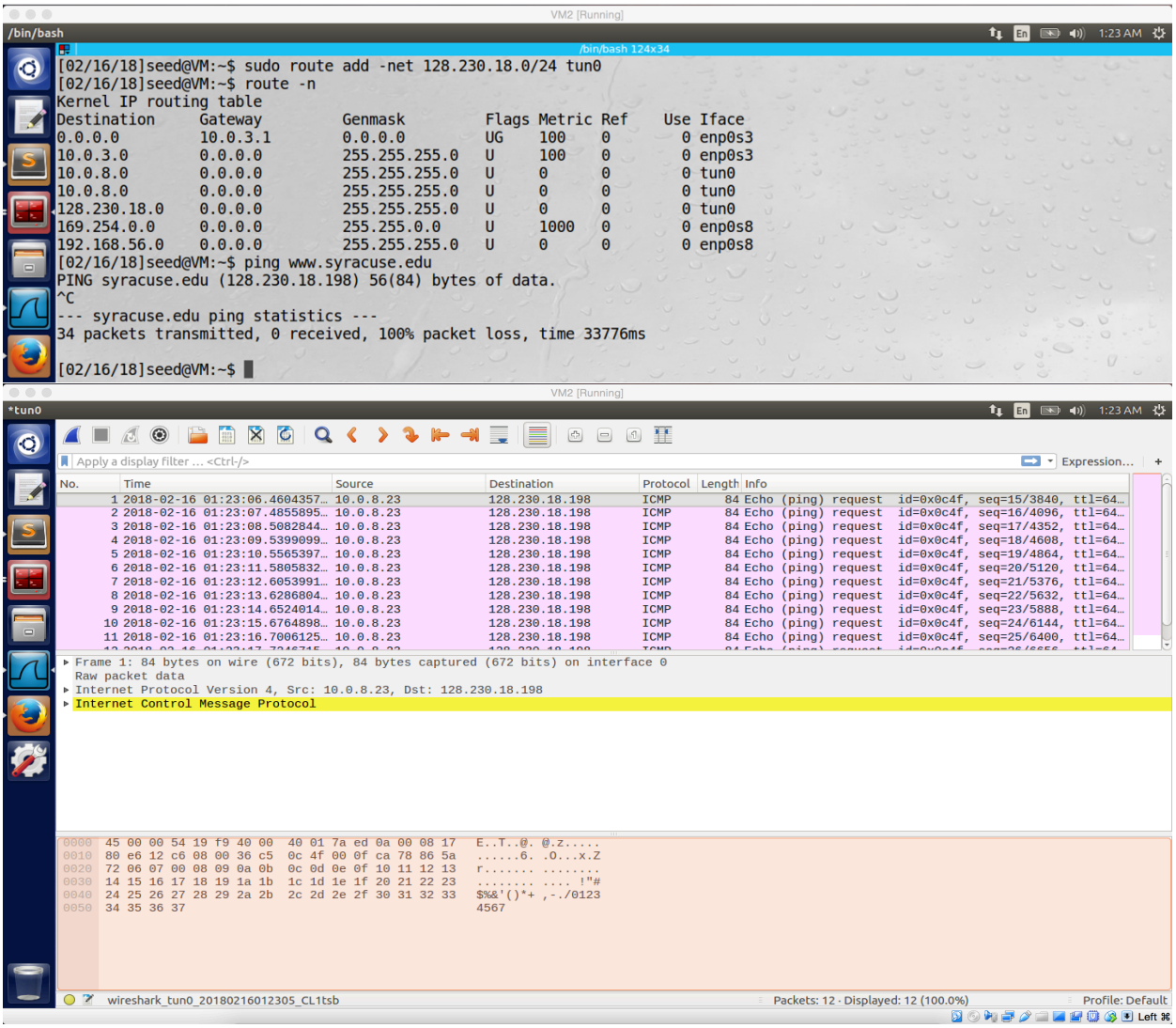
screenshot 2 on VM B. after we added rule to block SU website, VM B cannot reach SU website anymore.

Observation and Explanation:

In this task, I add rule on VM B to block SU website. Because we need to make sure that our TUN interface will not be blocked, we need to set rule on VM B's real network interface. The iptables command (screenshot 1) blocks packets to the 128.230.18.0/24 network going through the VM B's real network interface enp0s3. In screenshot 2, we can see SU website is blocked.

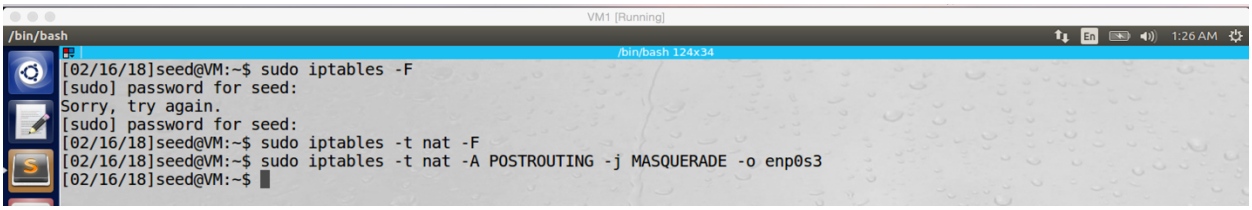
Task 4 Part1: Bypass firewall, reach blocked website

In this task, I will use 2 VMs, VM A and VM B. VM A is running the VPN server program. VM B is running the VPN client program. They are all set in task1, and VM B is blocking to visit SU website in task3. Therefore, the goal is to use VPN to make VM B bypass firewall and visit SU website.



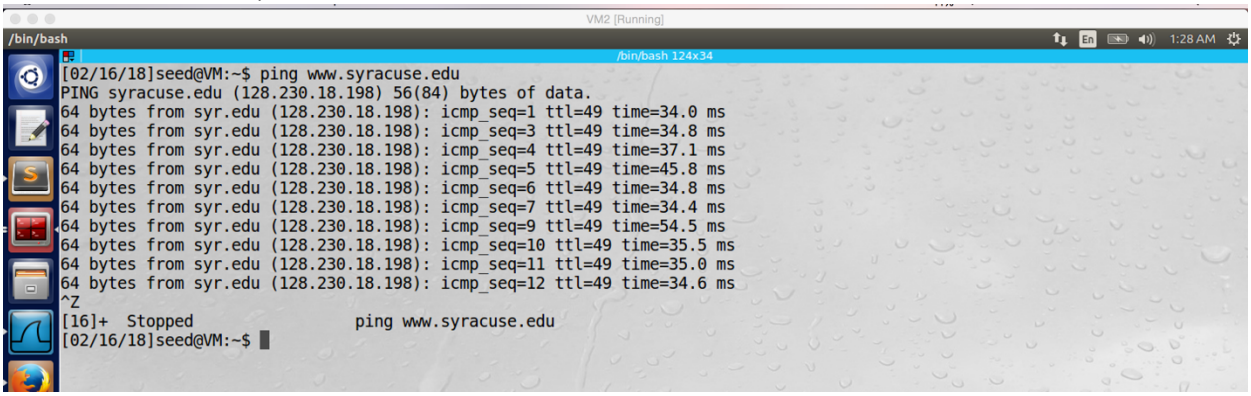
screenshot 1 on VM B. Add rule to routing table, so when we visit SU website, the packets will go through the VPN tunnel. Now we can send ICMP request to SU, but we cannot get reply.

We do **Task2** here

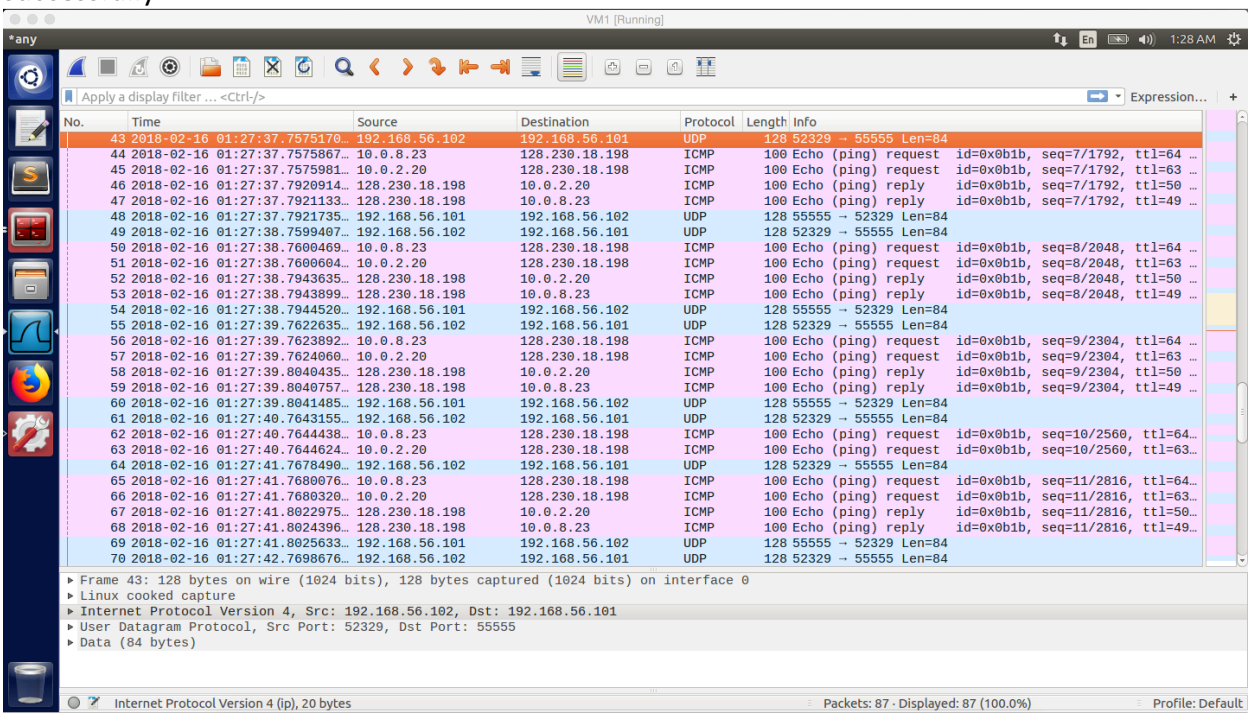


screeshot2 on VM A. We run a command on VM A, we will talk about this detail in the observation and explanation part.

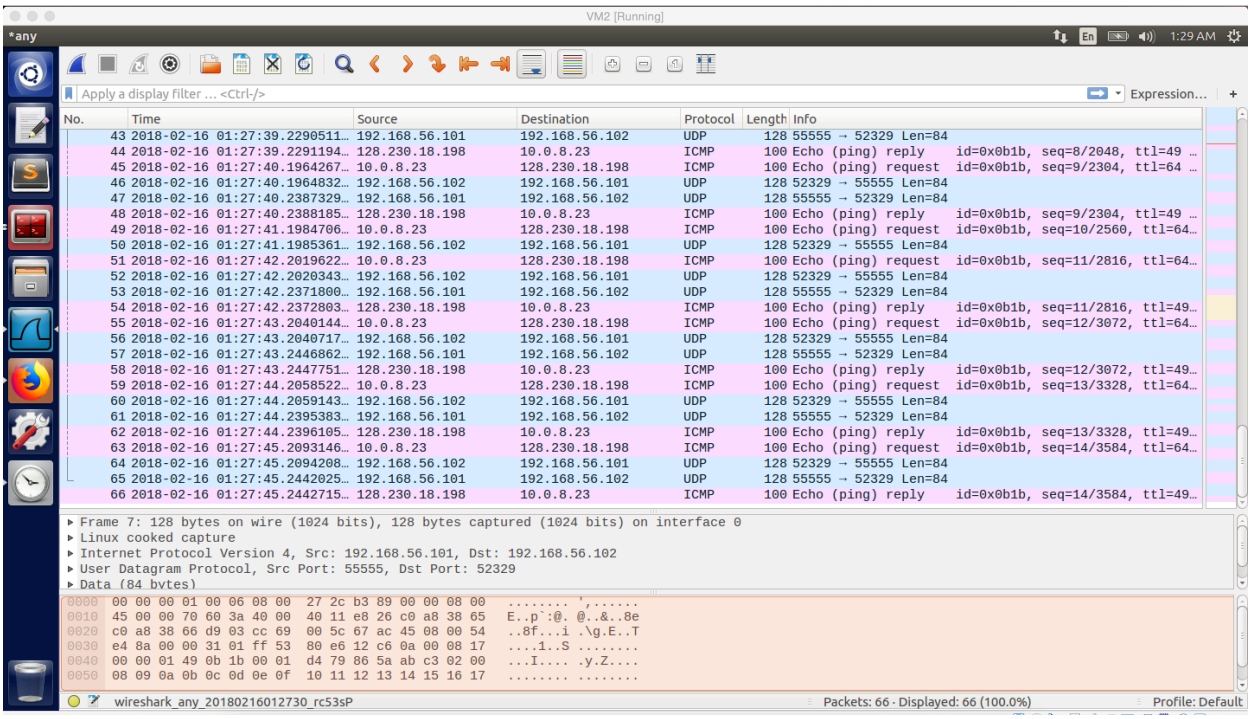
Come back to VM B, now we can access SU website



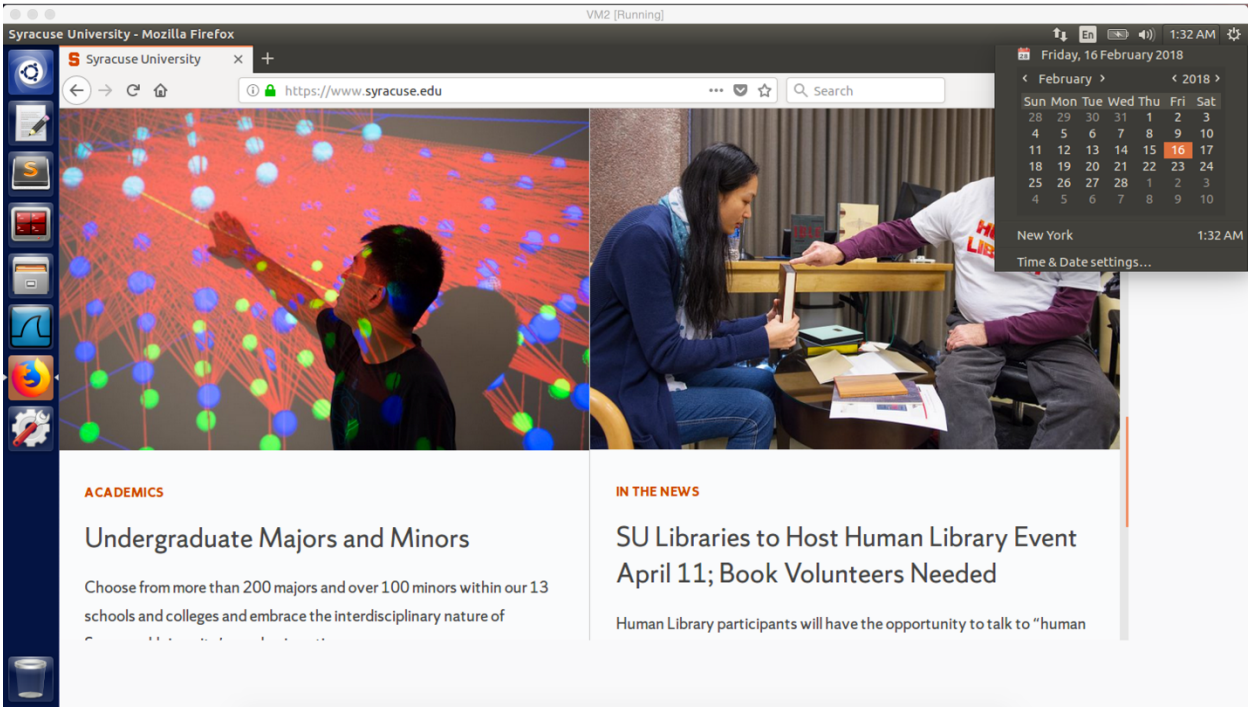
screenshot 3.1 on VM B, sending ICMP request successfully, and receiving ICMP reply from SU successfully



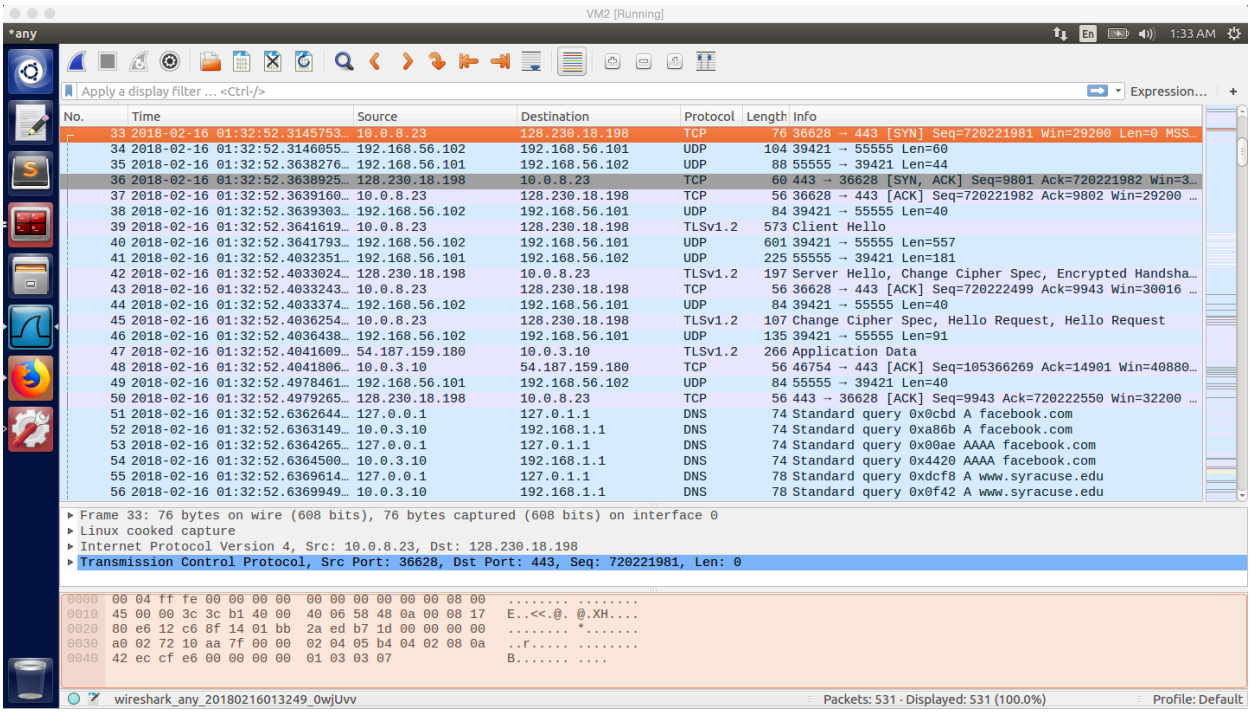
screenshot 3.2 on VM A. Wireshark captured the whole traffic



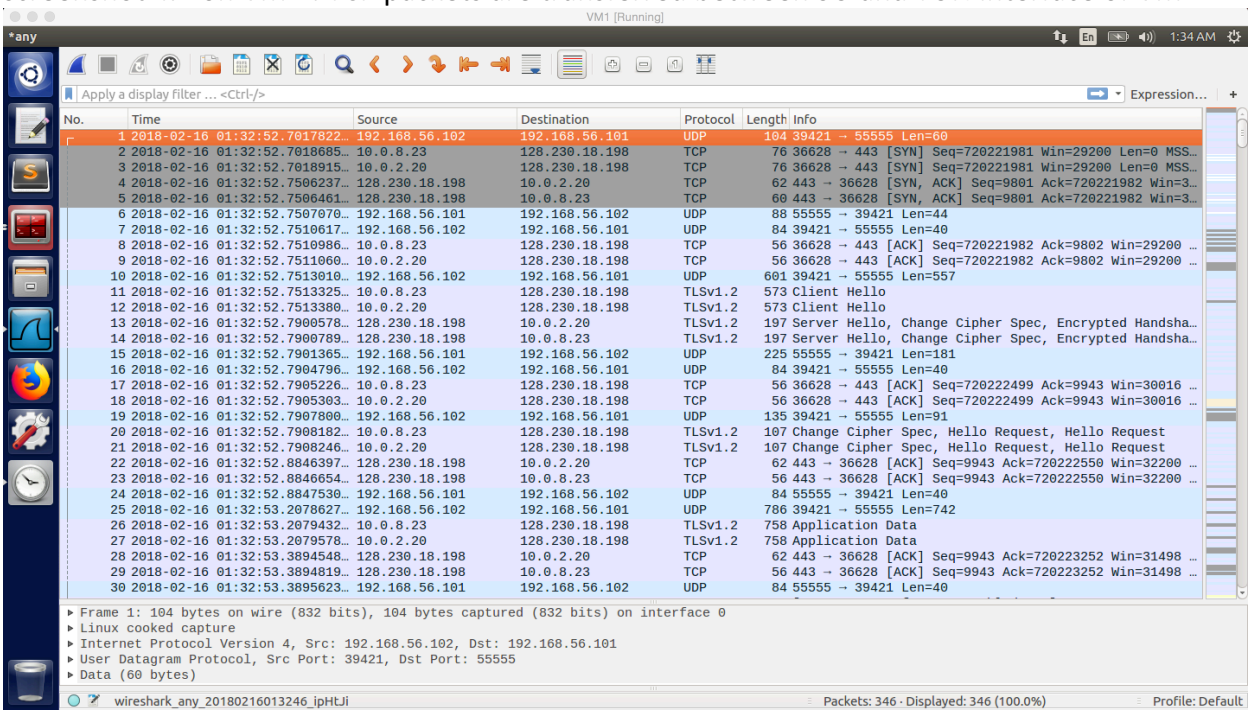
screenshot 3.3 on VM B. ICMP packet is transferred by the tun0 interface



screenshot 4.1 on VM B. we can use browser to visit SU website



screenshot 4.2 on VM B. TCP packets are transferred between SU and TUN interface of VM B



screenshot 4.3 on VM A. VM A and VM B using the VPN tunnel to transfer UDP packets.

Observation and Explanation:

After we setup VM A and VM B, we can bypass firewall now.

The first step is to add rule to the routing table of VM B. “sudo route add –net 192.168.56.0/24 tun0”, this rule will make sure that all packets with IP address of 128.230.18.0/24 will not go to the real NIC, instead they will go to tun0. After adding this rule, I

try to ping SU again; at this time, our ICMP packet send out successfully, but we cannot get reply from SU. (screenshot 1)

This problem is discussed in task 2 of lab description. In fact, packets are successfully go through the VPN tunnel from VM B to VM A, and VM A send packets to SU successfully as well. However, because the packet has source IP address of TUN interface of VM B, which is 10.0.8.23. Therefore, when SU send reply back, it will use this IP address as destination IP address. VM A's NAT server does not know this IP address, so it will drop the packet. After we run the command in the task2 from the lab description. This command will replace the source IP address of tun0 of VM B with real IP address of VM A (10.0.2.20), so we can receive ICMP reply from SU now (screenshot 2).

Go back to VM B, now we can receive ICMP reply from SU (screenshot 3.1). In the screenshot 3.2 and 3.3 we can see the whole traffic (they describe same traffic, but because 3.3 includes more information (packet send out from 10.0.2.20 to SU), I use 3.3 as example), the ICMP packet (No.44) is sent out from 10.0.8.23 to 128.230.18.198. This means the packet is sent out by tun0 of VM B. And then, the packet (No.45) is sent out from 10.0.2.20 to 128.230.18.198, this means the packet is sent out from VM A (because we add command on task2, so the source IP becomes VM A's IP address) to SU. Afterwards, SU send reply back (No.46) to VM A, and VM A forward the packet (No.47) to tun0 of VM B. Finally, we received the packet on VM B. Except these ICMP packets, we can also see UDP packets are transferred between 192.168.56.101 and 192.168.56.102. In fact, when we ping SU on VM B, the ICMP packet is routed to tun0 10.0.8.23 (because the rule we added before). And then the VPN program will put the ICMP packet into UDP packet (No.43) and sent it out by the VPN tunnel. Because the tunnel is established on the host-only network, so the UDP packet has source IP address of 192.168.56.102 (VM B) and destination IP address 192.168.56.101 (VM A). after VM A received this packet, it will take out the ICMP packet and send it out to SU. After SU received ICMP request packet, it sends ICMP reply packets back goes in the same path with reverse order. The ICMP reply reaches VM A first, and the packet will be routed to TUN interface of VM A, and then the VPN program puts it in a UDP packet and send it out to VM B by VPN tunnel (No.48). Finally, the ICMP reply arrives tun0 of VM B, and the tun0 gives the packet to kernel.

After ICMP request success, we should able to visit SU website now. As screenshot 4.1 shows the website can be opened normally. As the screenshot 4.2 and 4.3 shows (they are screenshot on different VMs, but describe same traffic), TCP packet transfer in the same path as ICMP packet. It is routed to tun0 of VM B (No.2 in 4.3), and then the VPN program put it into UDP packet. Afterwards, the TCP packet is transferred to VM A by the VPN tunnel (No.1 in 4.3). After VM A received it, the VM just takes the TCP packet out and forward it to SU (No.3 in 4.3). Afterwards, SU also send packets back with the same path, but reverse order. VM A received packet from SU (No.4 in 4.3), and then VM A puts the TCP packet into UDP packet and send it to VM B by VPN tunnel (No.6 in 4.3). Finally, VM B received it (No.5 in 4.3).

Task 4 Part2: Bypass firewall, doing telnet

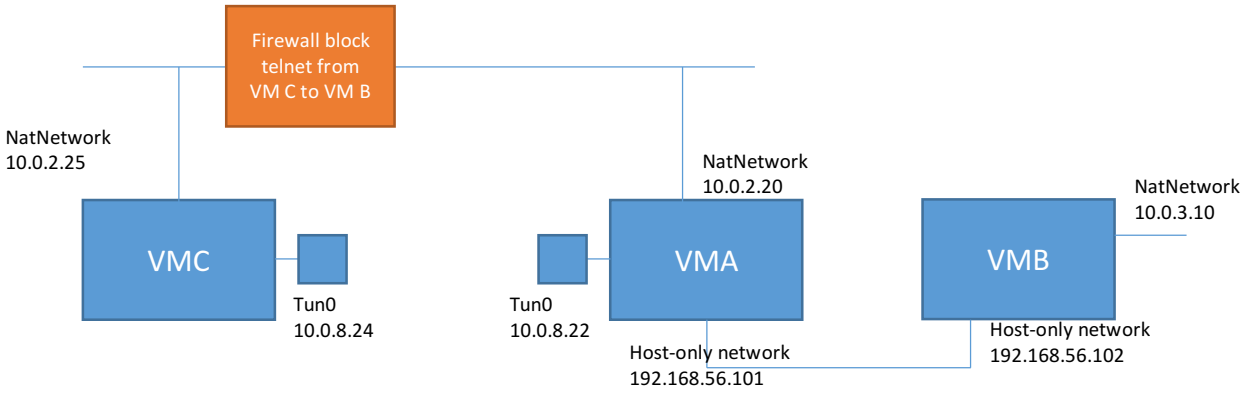
For this task, I change the environment setting for VMs.

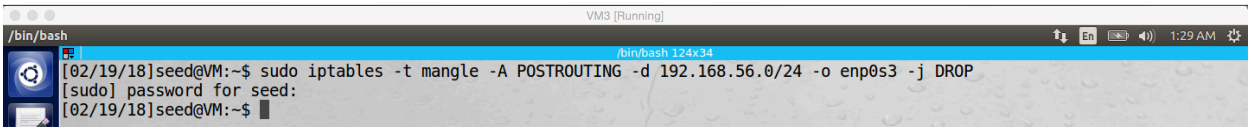
For VM A, I did not change anything, and it still run VPN server program.

For VM B, its NatNetwork and Host-only network is not being changed. But this time, it will not run VPN client program. It is a host machine, but VM C is blocked to reach it.

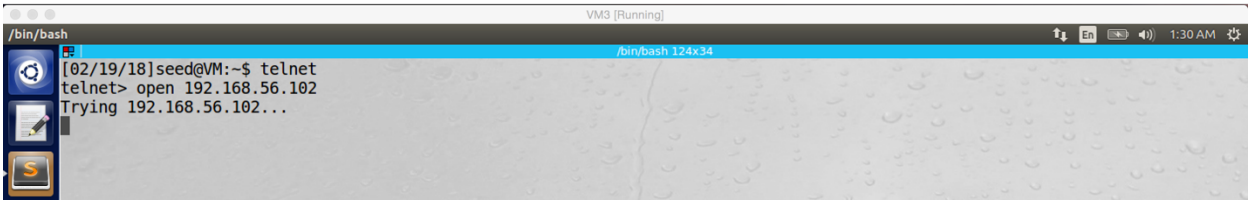
For VM C, it is new machine, and it will run VPN client program (with SERVER_IP = “10.0.2.20”) in this task. It’s real IP address is 10.0.2.25, and its tun0 IP address is 10.0.8.24.

In this task, my goal is to do telnet from VM C to VM B.

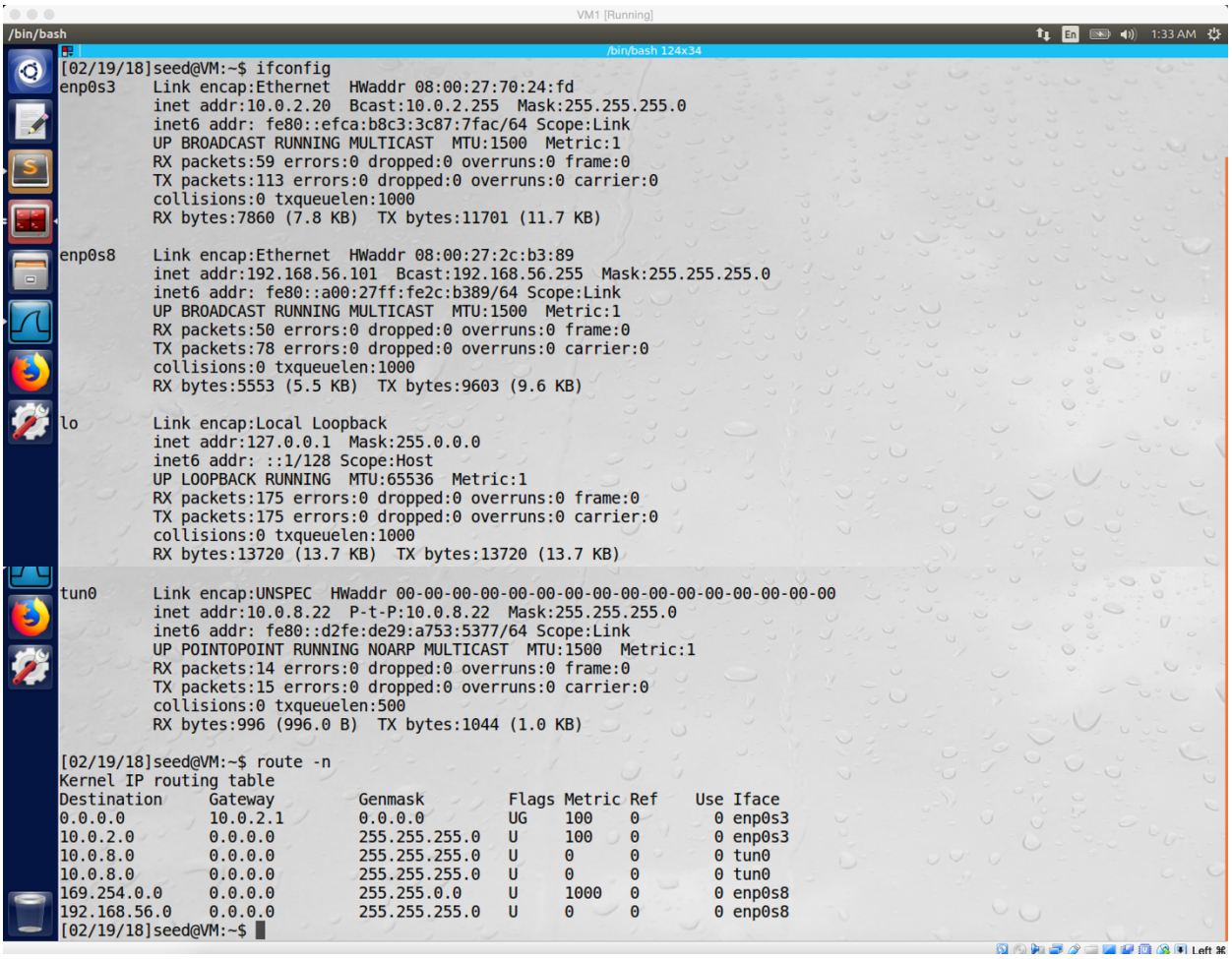




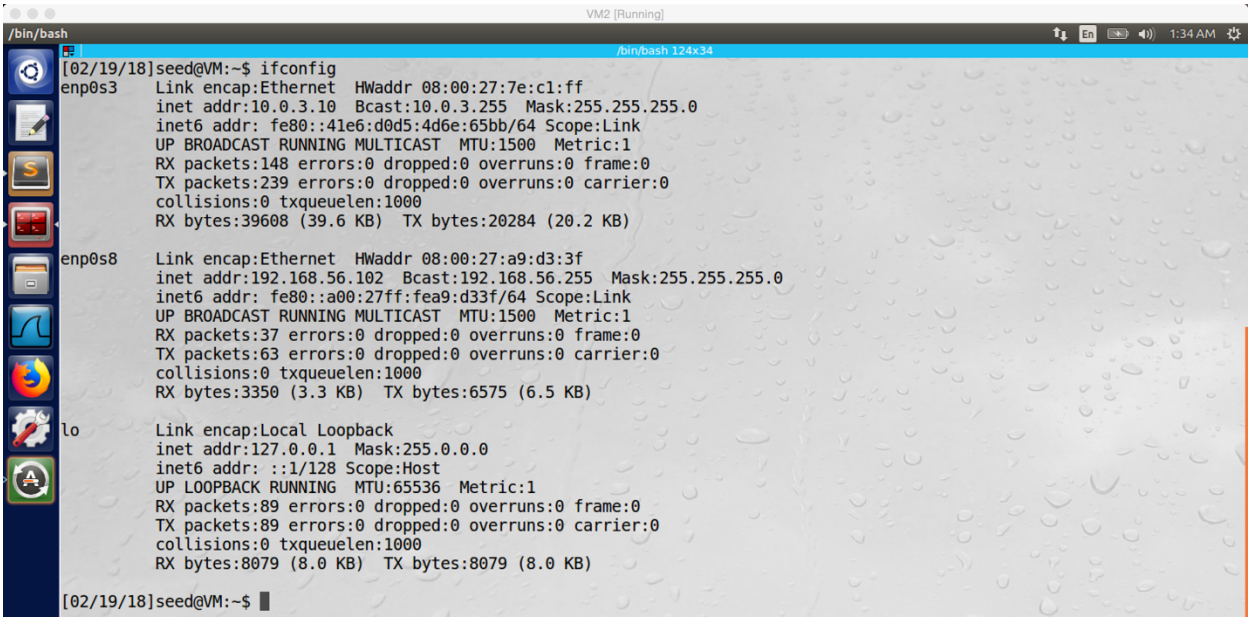
screenshot 1 on VM C. we add the iptables rule to make sure VM C cannot visit VM B.

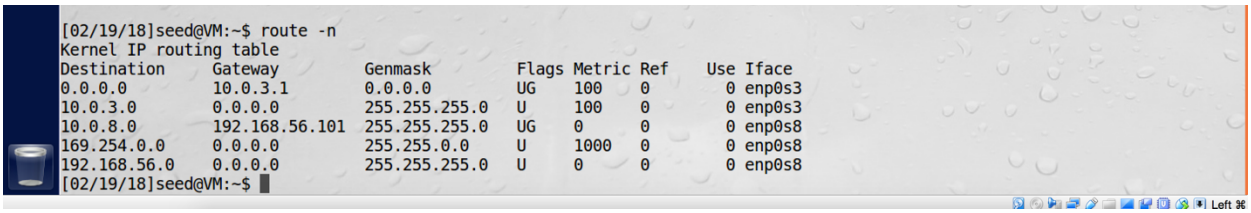


screenshot 2 on VM C. VM C cannot telnet VM B.

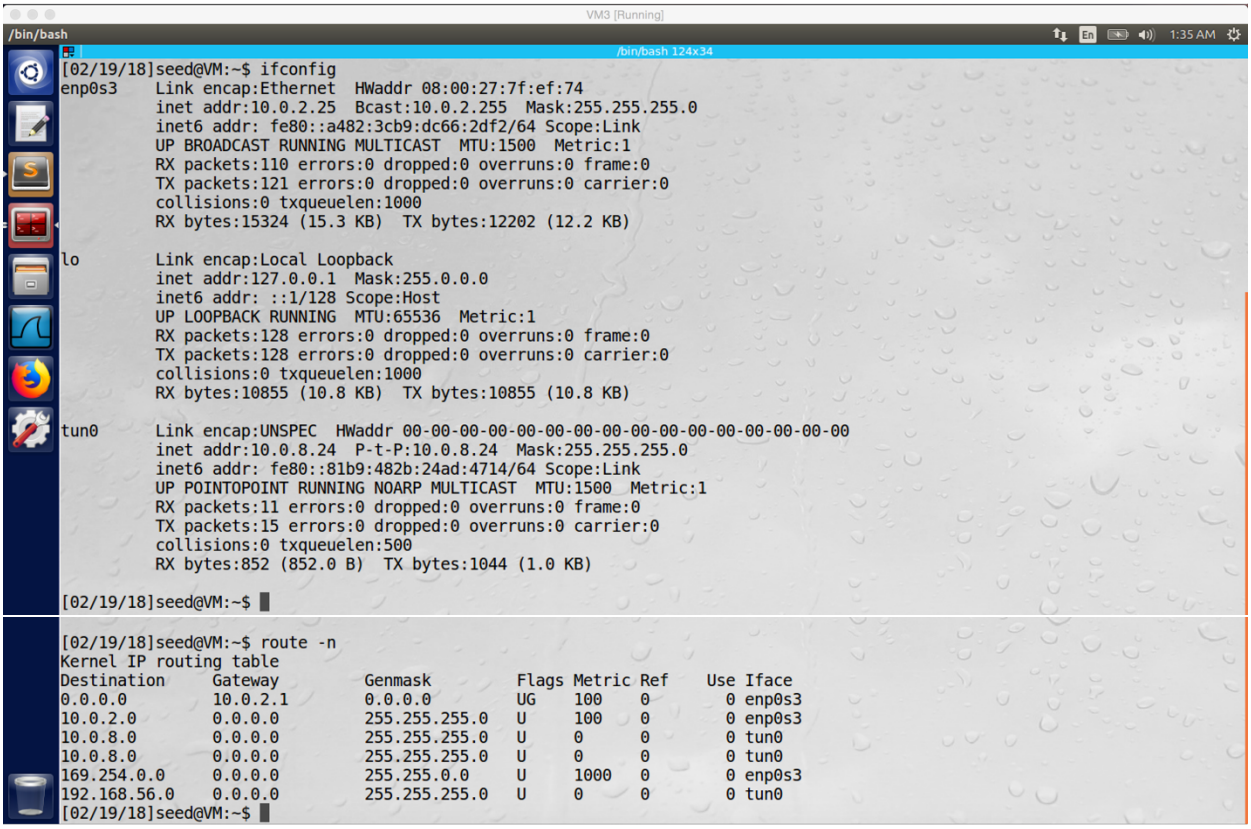


screenshot 3 on VM A. After configuration of VM A

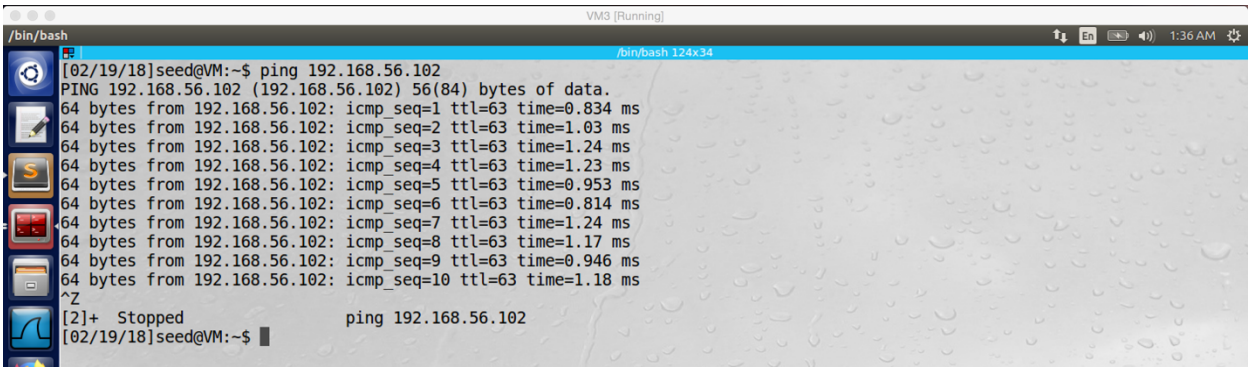




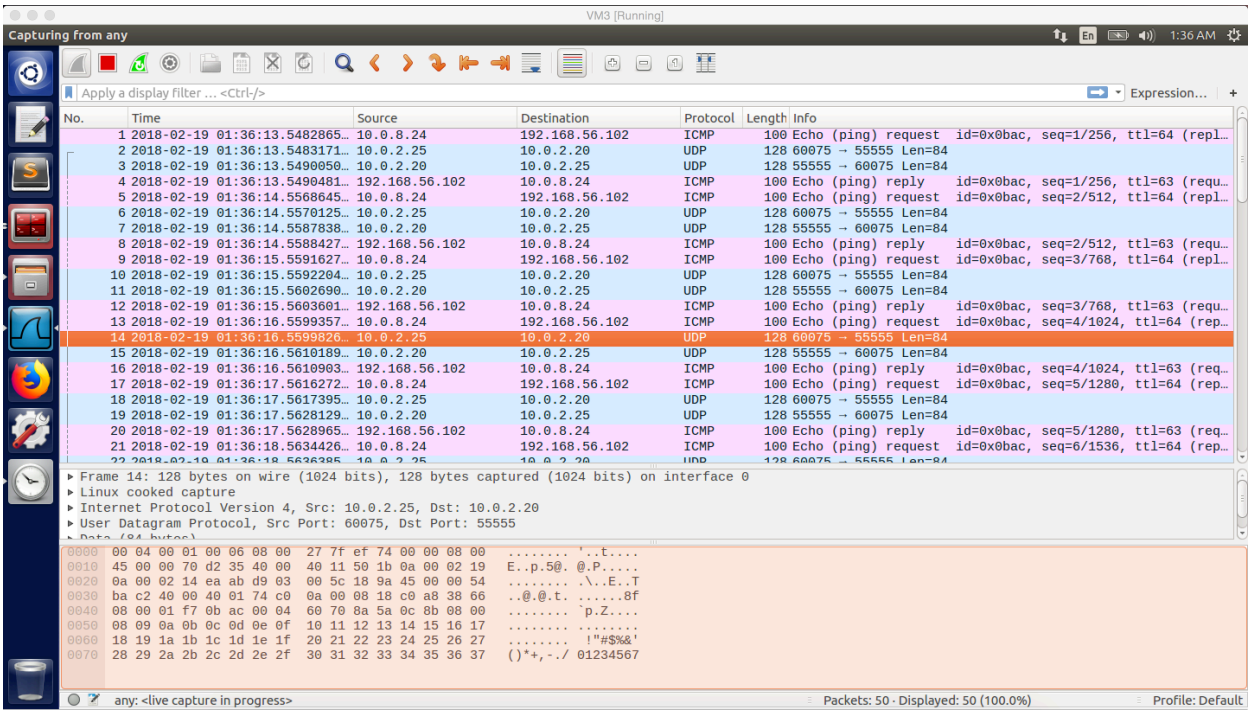
screenshot 4 on VM B. After configure VM B



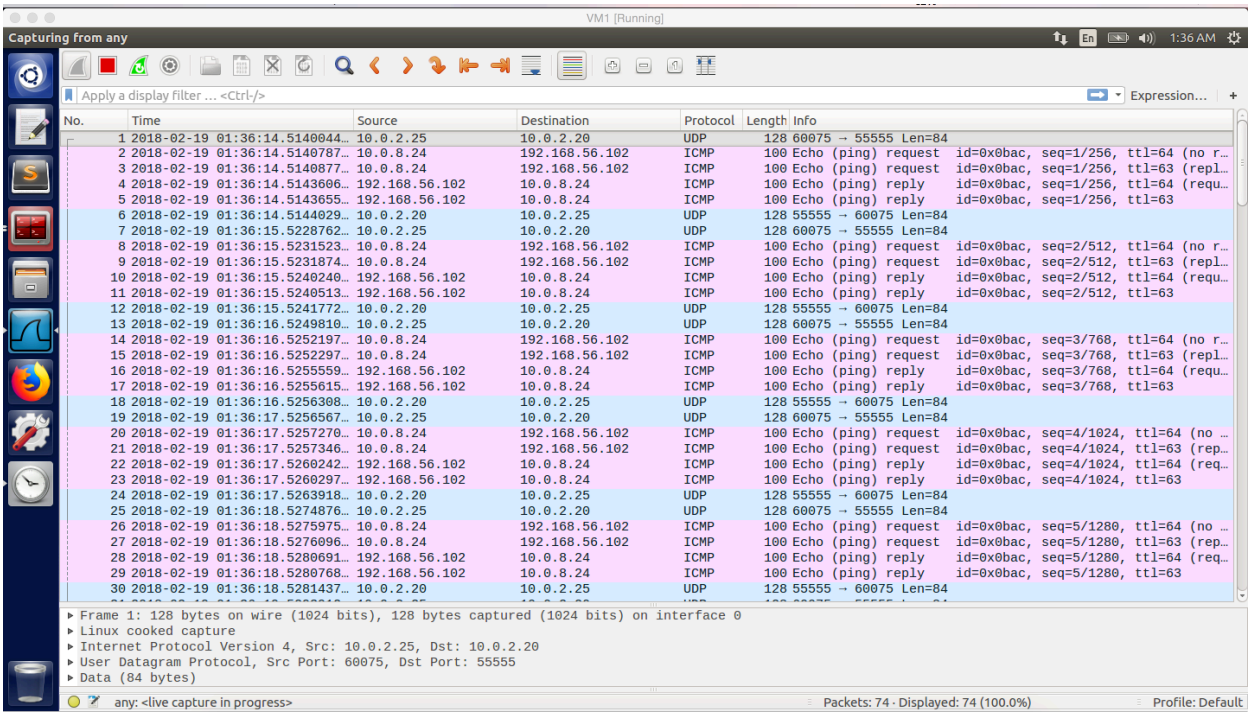
screenshot 5 on VM C. After configure VM C



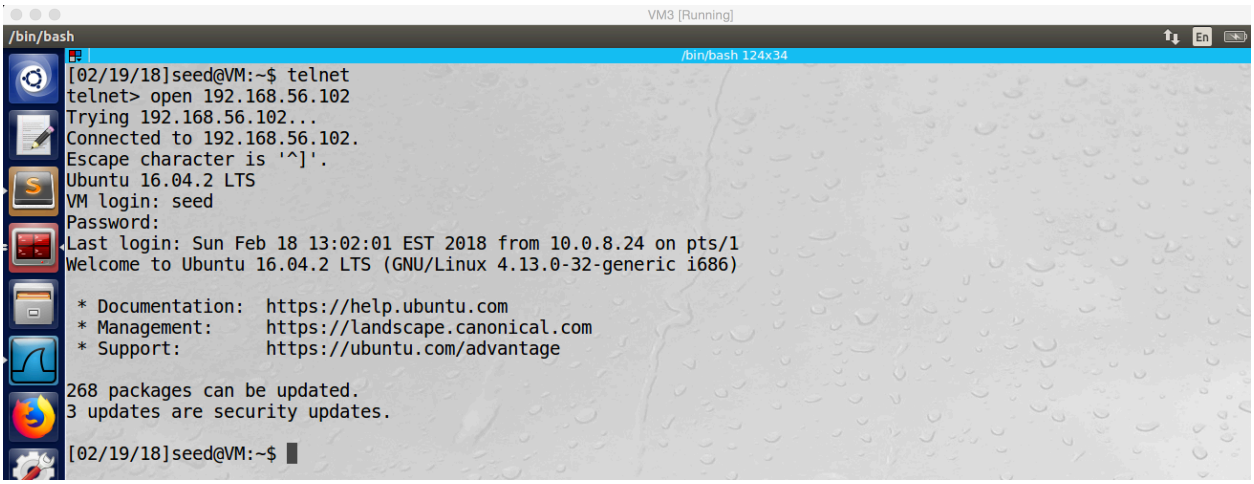
screenshot 6 on VM C. VM C successfully send ICMP request to VM B and received reply



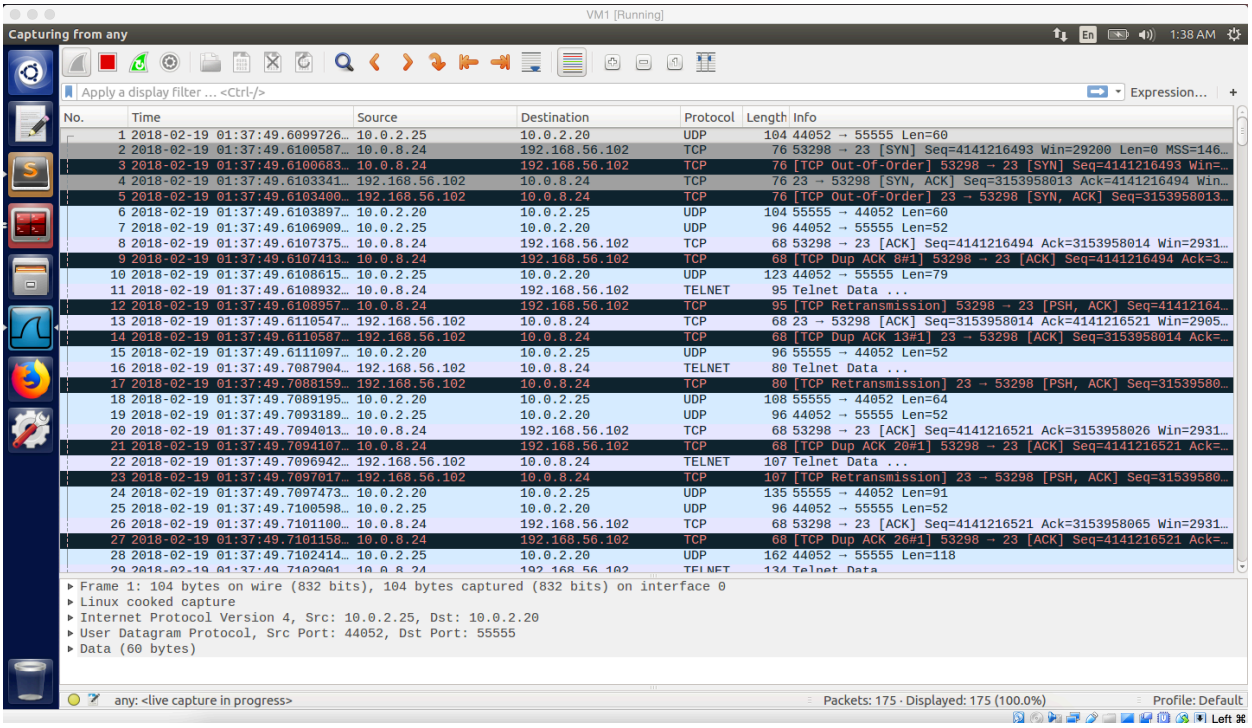
screenshot 6.1 on VM C. Wireshark captures the whole traffic



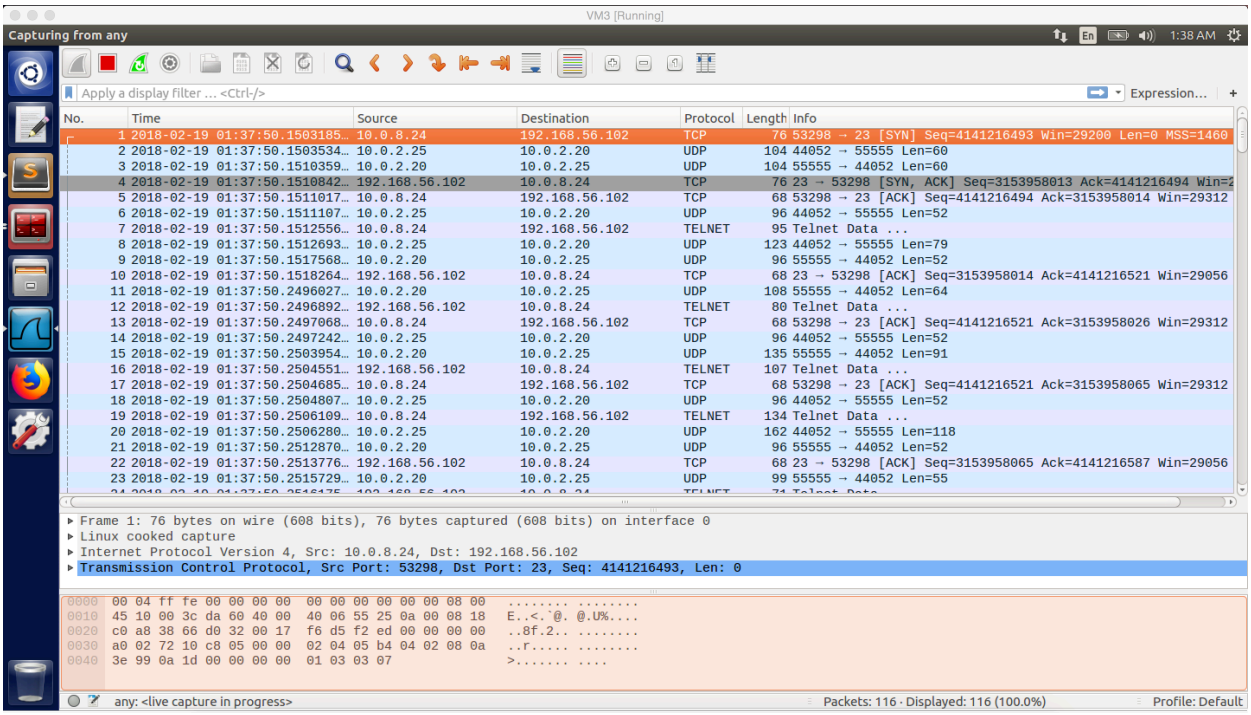
screenshot 6.2 on VM A. VM C successfully send ICMP packet to VM B and get reply immediately. Wireshark capture the whole traffic.



screenshot 7 on VM C. successfully login the telnet server on VM B from VM C via VPN.



screenshot 8.1 on VM A, Wireshark captured the whole traffic,



screenshot 8.2 on VM C. Wireshark captured TCP packets between VM C and VM B, and UDP packets between VM A and VM C

Observation and Explanation:

Before we start the task, I need to make sure that VM C is not able to telnet to VM B. Therefore, I use the iptables command in task3(screenshot 1), but I change the IP address to 192.168.56.0/24, this is the host-only network of VM B. This rule will prevent VM C to visit VM B, which will block telnet as well. After that, I did test. I try to telnet from VM C to VM B, there is nothing happen (screenshot 2).

Now, we can start this task. First, we run VPN server program on VM A, and VPN client program on VM C (with SERVER_IP = “10.0.2.20”). Second, we need to configure these three VMs. For VM A, I did same things as task1. (create, specify, assign IP address, and activate TUN interface. and add rule to routing table). For VM C, except these things, we also need to add one more rule which is “sudo route add –net 192.168.56.0/24 tun0”, this rule makes sure that all packets which go to network 192.168.56.0/24 are routed to tun0 of VM C. Finally, we need to configure VM B, we just need to add one rule to its routing table, “sudo route add –net 10.0.8.0/24 gw 192.168.56.101 enp0s8”. When the packet sends out from VM C, the source IP address is 10.0.8.24. Therefore, when VM B sends packet back, the destination IP address should be 10.0.8.24 as well. However, it does not know such IP address, so it cannot send the packet. Furthermore, we also need to make sure that the reply packet will go through the same path (VPN tunnel); otherwise, the packet cannot reach VM C. Therefore, we want to VM B to send the packet to VPN server which is VM A. And then the packet will go through the VPN tunnel and reach VM C. The above rule solved problem, it makes sure that packets go to 10.0.8.0/24 will be routed to 192.168.56.101 (VM A).

After VMs configuration, we try to send ICMP request from VM C to VM B, and we success. As screenshot 6.1 and 6.2 show (they are screenshot on different VMs, but describe

same traffic), the ICMP request (seq 1) is sent out from 10.0.8.24 which is tun0 of VM C, and received by VM B 192.168.56.102. And UDP packets are transferred between 10.0.2.20 and 10.0.2.25, which are VM A and VM C. This is same as task 4 part1. When we ping 192.168.56.102, the ICMP packet is routed to tun0 of VM C (No.2 in 6.2). And then the TUN interface put the ICMP packet into UDP packet and sent out from 10.0.2.25 (the real NIC). Because the VPN tunnel is established between VM C and VM A with IP address of 10.0.2.25 and 10.0.2.20, we can see the UDP packet (No.1 in 6.2) has source IP 10.0.2.25 and destination IP 10.0.2.20. After VM A received this packet, it takes ICMP packet out and send to VM B (No.3 in 6.2); however, this time the source IP address is 10.0.8.24 because we didn't run command which we used in task2. After VM B received the packet, it will send reply to 10.0.8.24 (No.4 in 6.2), but this packet is routed to VM A because the rule we added before. After VM A received this packet, it put it in UDP and send it out via VPN tunnel to VM C (No. 6 in 6.2). Finally, VM C received it (No.5 in 6.2).

Now, we can do telnet from VM C to VM B, and we success (screenshot 7). The traffic flow is same as the ICMP packet (8.1 and 8.2, they are screenshot on different VMs, but describe same traffic). Packet No.1 in 8.2 is routed to tun0 of VMB, and then VM B put it in UDP packet and send it out via VPN tunnel (No.2 in 8.2), after VM A received it, it takes out the TCP packet and forward to VM B (No.3 in 8.1). After VM B received packet, it will send reply to VM C with destination address 10.0.8.24, but the packet will be routed to VM A (No.4 in 8.1). After VM A received it, it will put it into UDP packet and send it out via VPN tunnel (No.3 in 8.2). Finally, VM B received TCP packet (No.4 in 8.2).

Appendix

Code fore VPN server

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <linux/if.h>
#include <linux/if_tun.h>
#include <sys/ioctl.h>

#define PORT_NUMBER 55555
#define BUFF_SIZE 2000

struct sockaddr_in peerAddr;

//when we run the VPN program, this function will be called firstly
//it will create TUN interface
int createTunDevice() {
   int tunfd;
   struct ifreq ifr;
   memset(&ifr, 0, sizeof(ifr));

   //specifies we are creating TUN interface
   ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
```

```
   //open the /dev/net/tun device and issue a corresponding ioctl() to
   //register a netword device with kernel
   tunfd = open("/dev/net/tun", O_RDWR);
   ioctl(tunfd, TUNSETIFF, &ifr);

   return tunfd;
}

//this function will establish a UDP packet based VPN tunnel when UDP request
//is received
int initUDPServer() {
    int sockfd;
    struct sockaddr_in server;
    char buff[100];

    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT_NUMBER);

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    //bind socket to port number
    bind(sockfd, (struct sockaddr*) &server, sizeof(server));

    // Wait for the VPN client to "connect".
    bzero(buff, 100);
    int peerAddrLen = sizeof(struct sockaddr_in);

    //read the UDP date arriving at port
    int len = recvfrom(sockfd, buff, 100, 0,
                (struct sockaddr *) &peerAddr, &peerAddrLen);

    printf("Connected with the client: %s\n", buff);
    return sockfd;
}

//when there is packet in TUN interface, this function will be called.
//and then put packet in a UDP packet and send it out via VPN tunnel
void tunSelected(int tunfd, int sockfd){
    int  len;
    char buff[BUFF_SIZE];

    printf("Got a packet from TUN\n");

    bzero(buff, BUFF_SIZE);
    //read original packet
    len = read(tunfd, buff, BUFF_SIZE);
    //put the packet into UDP packet and send the packet out
    sendto(sockfd, buff, len, 0, (struct sockaddr *) &peerAddr,
                    sizeof(peerAddr));
}

//if the socket interface has packet, this means there is a packet here
//through the VPN tunnel
//so we need to take the packet out from the UDP packet and send it to kernel
```

```
void socketSelected (int tunfd, int sockfd){
    int  len;
    char buff[BUFF_SIZE];

    printf("Got a packet from the tunnel\n");

    bzero(buff, BUFF_SIZE);
    //take packet out from UDP packet
    len = recvfrom(sockfd, buff, BUFF_SIZE, 0, NULL, NULL);
    //send the packet to kernel
    write(tunfd, buff, len);

}
int main (int argc, char * argv[]) {
   int tunfd, sockfd;

   tunfd  = createTunDevice();
   sockfd = initUDPServer();

   // Enter the main loop
   while (1) {
     fd_set readFDSet;

     //monitor transfer packets between two file descriptors (one for TUN,
     //one for socket) by using system call select()
     FD_ZERO(&readFDSet);
     //using FD_SET to store monitored file descriptors in a set
     FD_SET(sockfd, &readFDSet);
     FD_SET(tunfd, &readFDSet);
     //given the set to select()
     //this function will block the process until data are available on one
     //of the file descriptors
     select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

     //FD_ISSER is used to know which file descriptor has received data
     if (FD_ISSET(tunfd,  &readFDSet)) tunSelected(tunfd, sockfd);
     if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd);
  }
}
```

Code fore VPN client, the only difference is client need to connect server, not initialize server. So VPN client program have following function instead of initUDPServer() in VPN server program.

```
//we create UDP socket, and use sendto() to send packet to the server
int connectToUDPServer(){
    int sockfd;
    char *hello="Hello";

    memset(&peerAddr, 0, sizeof(peerAddr));
    peerAddr.sin_family = AF_INET;
    peerAddr.sin_port = htons(PORT_NUMBER);

    //SERVER_IP need to be defined
    peerAddr.sin_addr.s_addr = inet_addr(SERVER_IP);

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
// Send a hello message to "connect" with the VPN server
sendto(sockfd, hello, strlen(hello), 0,
            (struct sockaddr *) &peerAddr, sizeof(peerAddr));
return sockfd;}
```