CPS 633 - Lab 1 Report

**Task 2.1a:**

Without running with root privileges, the shell throws a “operation is not permitted” error which means that the operation is being blocked for normal users who do not have root access.

**Task 2.1b:**

To capture only ICMP packets we set the filter to “icmp”. Must include the quotation marks or else it will not run. The code is the exact same as the sample code provided.

#!/usr/bin/python

from scapy.all import \*

def print\_pkt(pkt):

pkt.show()

pkt = sniff(filter=”icmp”,prn=print\_pkt)

Of course this must run with root privileges or else it will not work.

To capture only TCP packets with the destination port of 23, all we do is change the filter to “tcp port 23”. Then to test to see if this actually sniffs packs only on port 23 we first run a simple ping to that VM. The program shows nothing because ping packets are ICMP. So we then run the command echo text | netcat 10.0.2.12 23 in order to first send a TCP packet and then send it to the correct port.

#!/usr/bin/python

from scapy.all import \*

def print\_pkt(pkt):

pkt.show()

pkt = sniff(filter=”tcp port 23”,prn=print\_pkt)

Finally, to capture packets that come from or go to a particular subnet we change the filter to "net subnet" where subnet is the address of the subnet you are trying to capture from. In this particular case we used subnet 128.230.0.0 because it was a subnet our VM was not attached to. To test our sniffer, we simply pinged the subnet from the other VM and were able to capture packets from it.

#!/usr/bin/python

from scapy.all import \*

def print\_pkt(pkt):

pkt.show()

pkt = sniff(filter=”net 128.230.0.0”,prn=print\_pkt)

**Task 2.2:**

To show that we can spoof ICMP echo request packets with an arbitrary source IP address, we made a simple but very effective change to the code. We changed the IP source address of our spoofed packet to an arbitrary address in order to trick the other VM into thinking the packet came from a different IP address. We then ran our new program with the destination address being our other VM and monitored it in Wireshark. It confirmed out packet had been registered as being received from our arbitrary address.

!/usr/bin/python

from scapy.all import \*

a=IP()

**a.src = "10.12.10.2"**

a.dst = "10.0.2.8"

b = ICMP()

p = a/b

send(p)

**Task 3:**

Using the traceroute program provided and Wireshark, we continuously increased the TTL metric by one until we reached the router address. We knew we got to the router when the address ended with 1.1 which is the typical ending of a router address in basic networking. Also we found that if we increased the TTL by 1 more after reaching the router address, then Wireshark would show an error for that packet which again confirms that we had indeed reached the router. It took us a total for 12 hops to get to the router. Below is the final source code we used and all the addresses we hit in order before we finally got to the router address.

#!/usr/bin/python

from scapy.all import \*

a = IP()

a.dst = "1.2.3.4"

a.ttl = 12

b = ICMP()

send(a/b)

10.0.2.1

192.168.251.1

141.117.231.1

172.16.200.9

172.30.0.10

172.29.6.50

172.29.3.142

10.10.199.91

172.29.1.141

172.29.1.49

172.29.1.35

**172.29.1.1**

**Task 4:**

First using the sniff() function we filter for ICMP echo requests by capturing ICMP packets that are ICMP type 8 and send them to the function print\_pkt(). In the function, we create an IP object then set its source IP address to the destination IP address of the echo request and its destination IP address to the source IP address of the echo request. We create an ICMP object then set its type to echo reply (because the default is echo request) and the code to that of an ICMP echo reply, which is 0. Then we set its identifier and sequence to those of the echo request. Finally, we stack a and b to create a new object representing an ICMP packet that we now send out.

#!/usr/bin/python

from scapy.all import \*

def print\_pkt(pkt):

a = IP()

a.src = pkt[IP].dst

a.dst = pkt[IP].src

b = ICMP()

b.type = “echo-reply”

b.code = 0

b.id = pkt[ICMP].id

b.seq = pkt[ICMP].seq

send(a/b)

pkt = sniff(filter=”icmp[icmptype] == 8”,prn=print\_pkt)

To test the program, we ran it on one vm while we ping 1.2.3.4 (an arbitrary IP address) on the other. Our technique worked because every packet that was sent when running the code showed up on Wireshark immediately followed by an echo reply.

