**Computer Networks Lab-18XD57**

**PACKAGE SUBMISSION**

**[Manipulating the network with Scapy]**

**AMCS**

**31.10.2021**

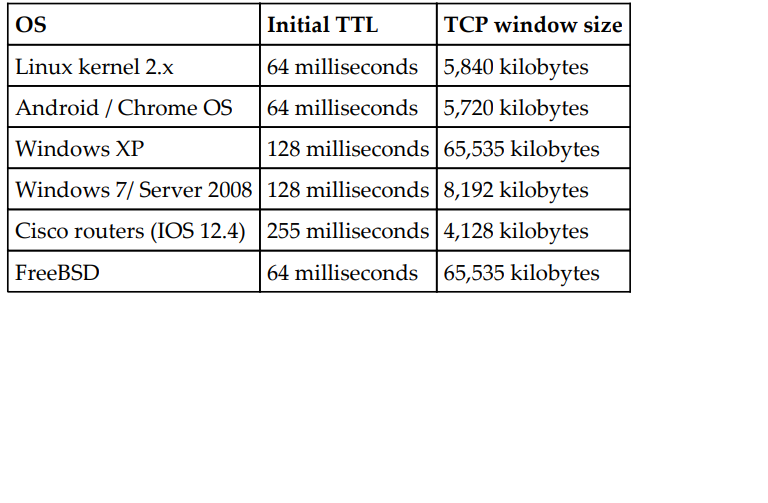
**CONTRIBUTION:**

**19PD04** ~ AKSHAYA L - OS Fingerprinting,Wireshark parser

**19PD05** ~ ALAGU PRAKALYA P- Mail Catcher (SMTP),IP Broadcast scanning

**Abstract:**

Using the packet manipulation library Scapy and network analyser Wireshark, we’ll demonstrate sniffing email credentials using telnet and the SMTP, POP3, and IMAP credentials. Having performed another application of this :ARP cache poisoning (MITM), further it is important to start gathering information about live systems/port and the services. Hence by deploying TTL analysis (below) OS fingerprinting is performed. And show how a mini-wireshark parser can be constructed . Finally , replay -traffic , a offline .pcap sniffer is deployed.



**1. Mail Catcher**

**Code:**

**Cmd:**

**Telnet smtp.gmail.com 25**

**HELO XYZ**

**AUTH LOGIN**

**STARTTLS**

**FROM (stuck\*\*\*\*)**

"""

# a sniffer that sniffs a packet in the network and prints it.

from scapy.all import \*

def pkt\_c(p):

print (p.show())

sniff(prn=pkt\_c,count=1)

"""

from scapy.all import \*

def pkt\_c(pkt):

if pkt[TCP].payload: #checking if we have any data payload

m\_pkt = bytes(pkt[TCP].payload) #mail packet received in bytes

m\_pkt= m\_pkt.decode("utf-8") #conversion into strings

print(m\_pkt)

if "user" in m\_pkt.lower() or "pass" in m\_pkt.lower():

print ("HOST -1 : %s" % pkt[IP].dst)

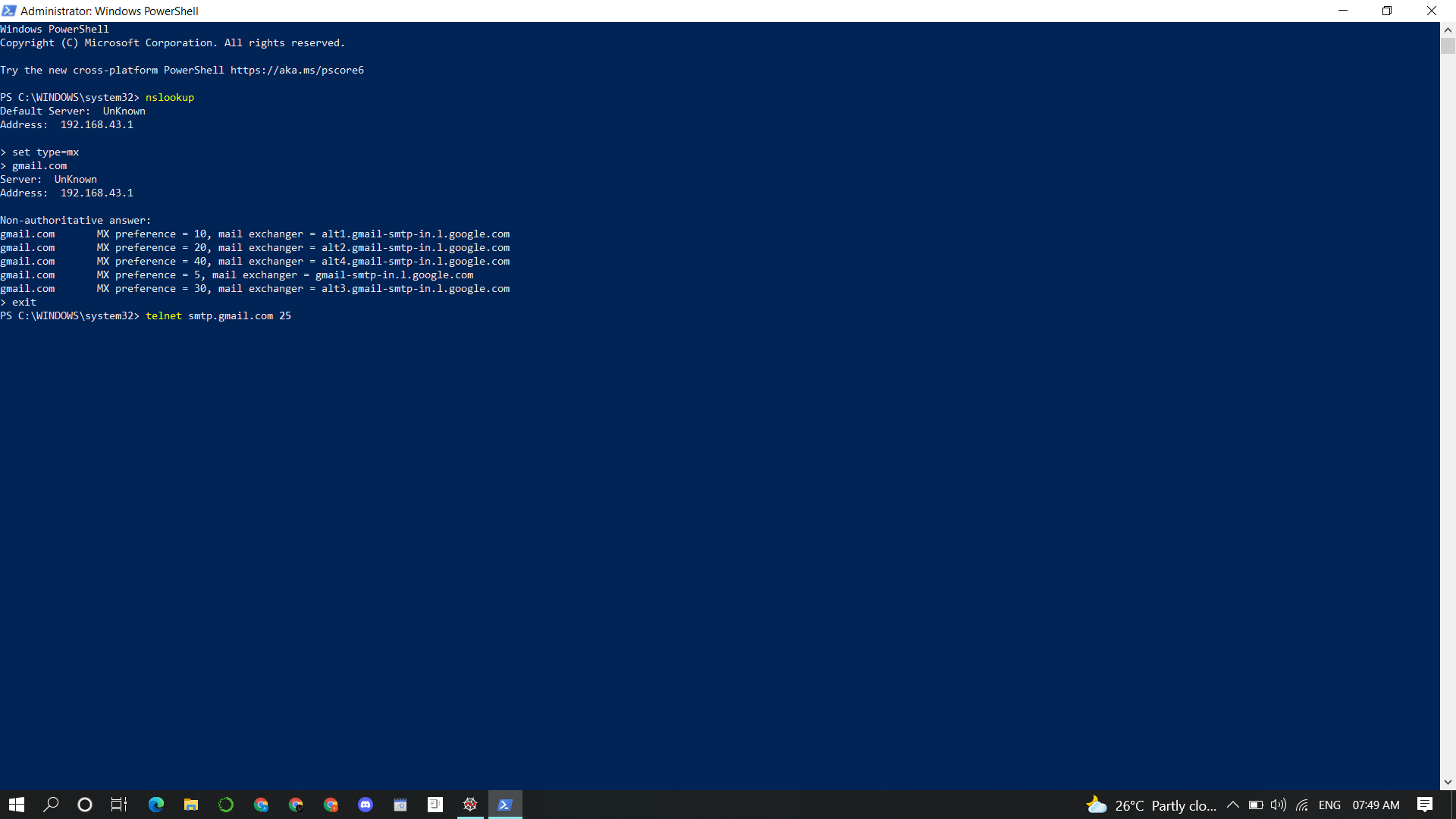
print (": %s" % pkt[TCP].payload)

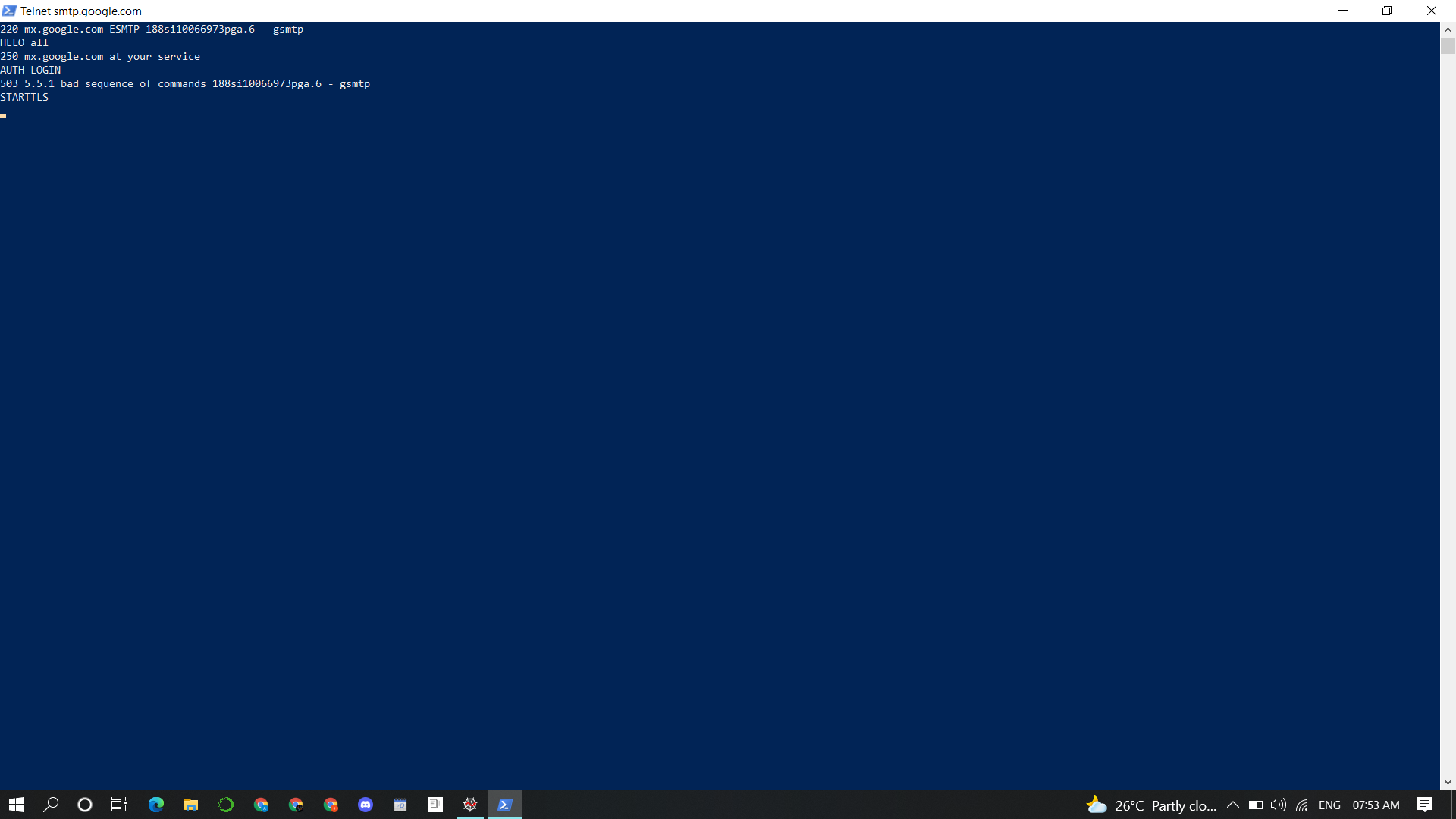
sniff(filter="tcp port 110 or tcp port 25 or tcp port 143",prn=pkt\_c,store=0)

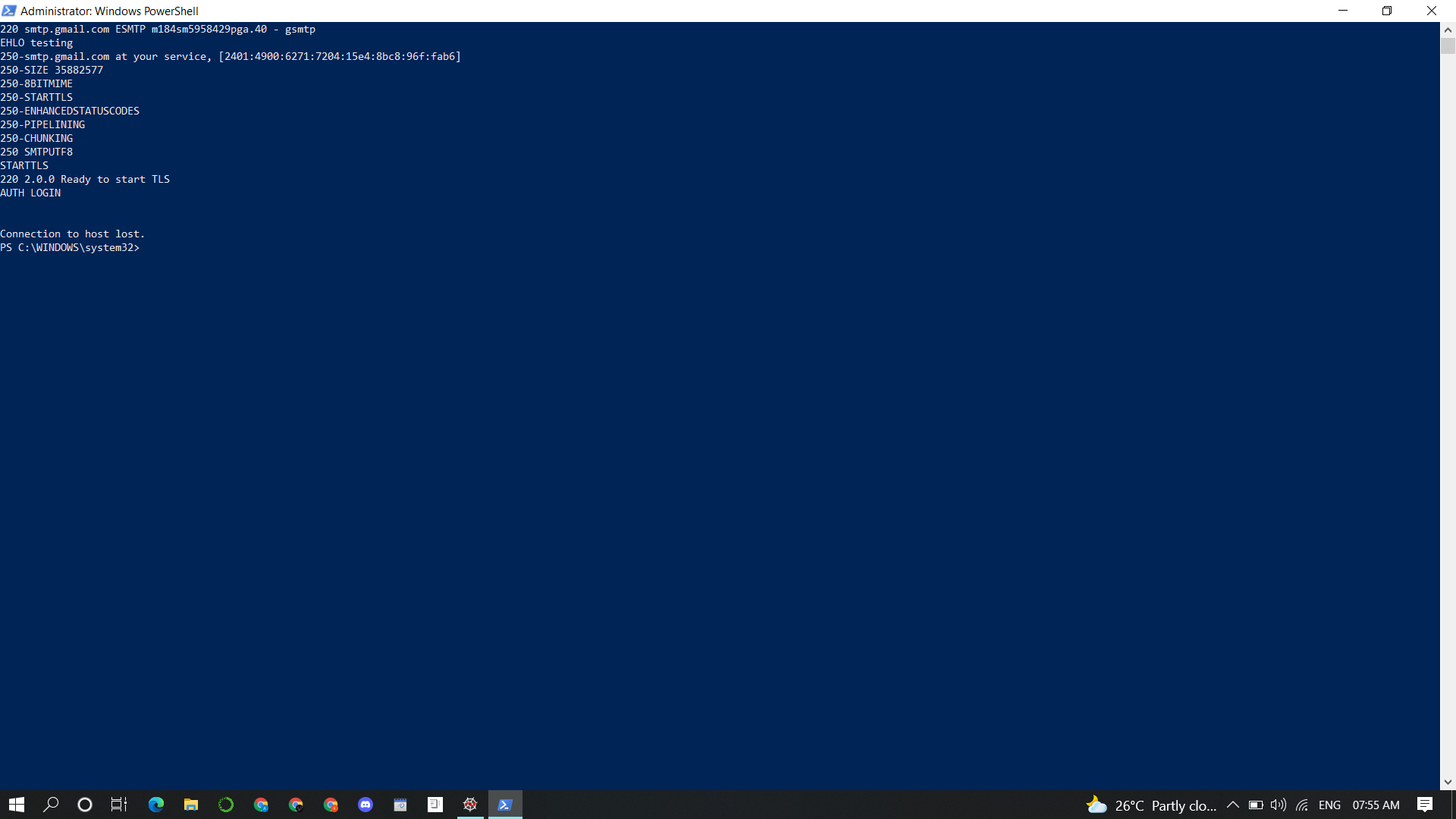
**#110 (POP3), 143 (IMAP), and SMTP (25) and store: for not spring pkt in memory**

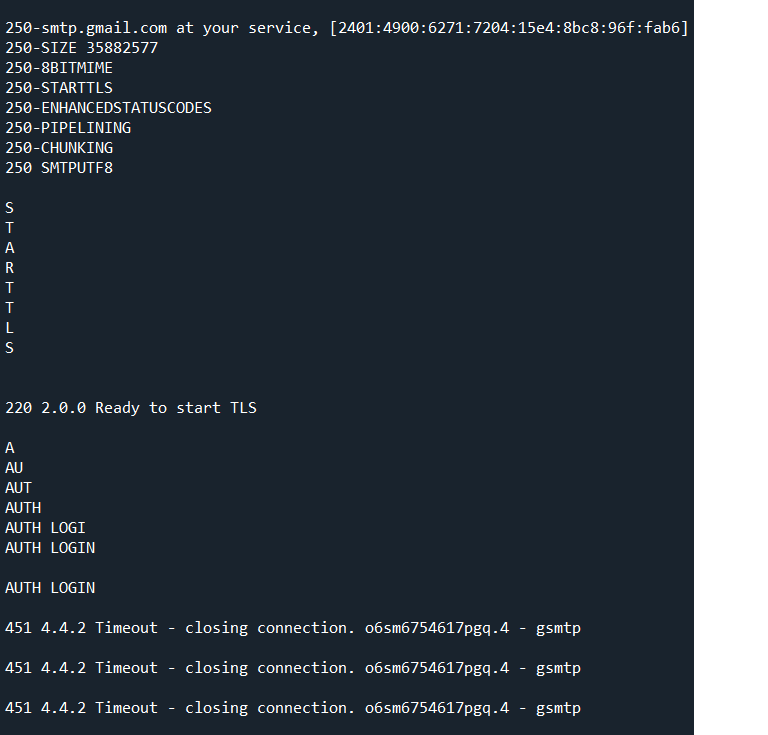
**OUTPUT:**

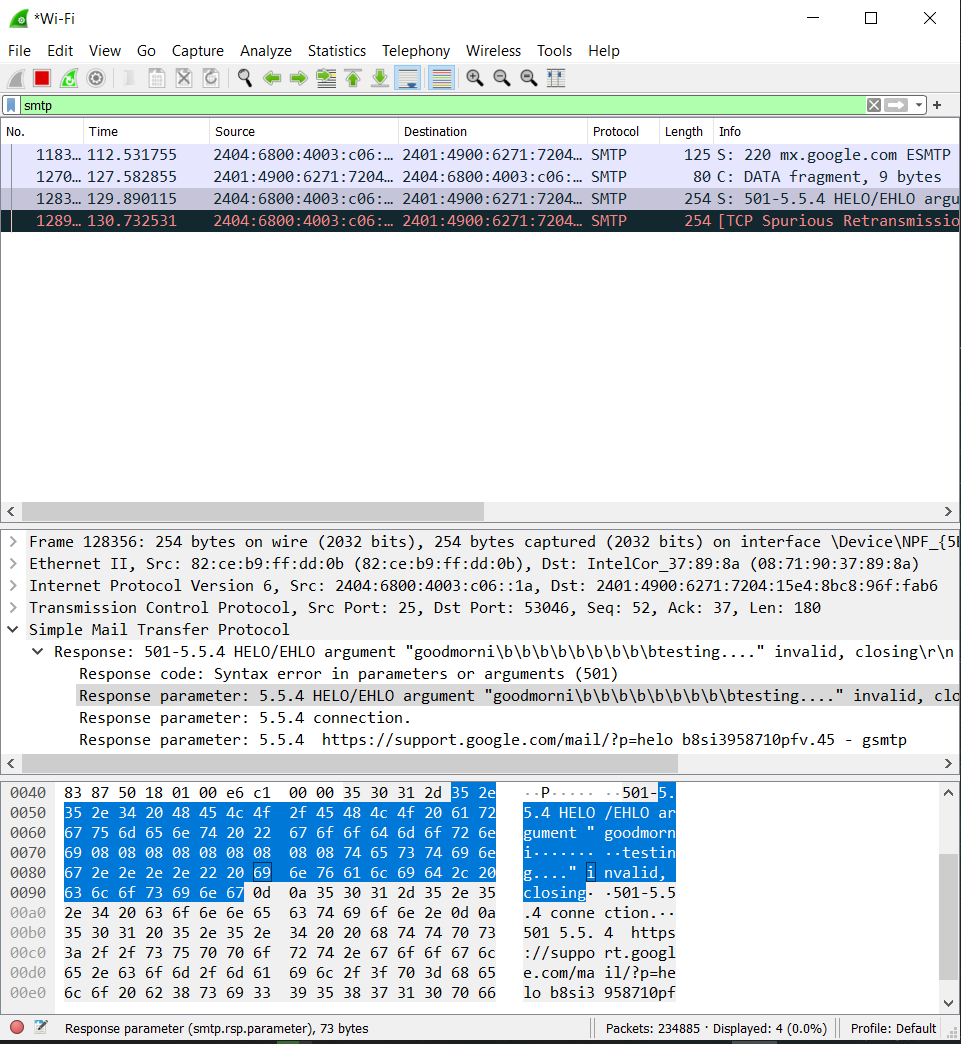
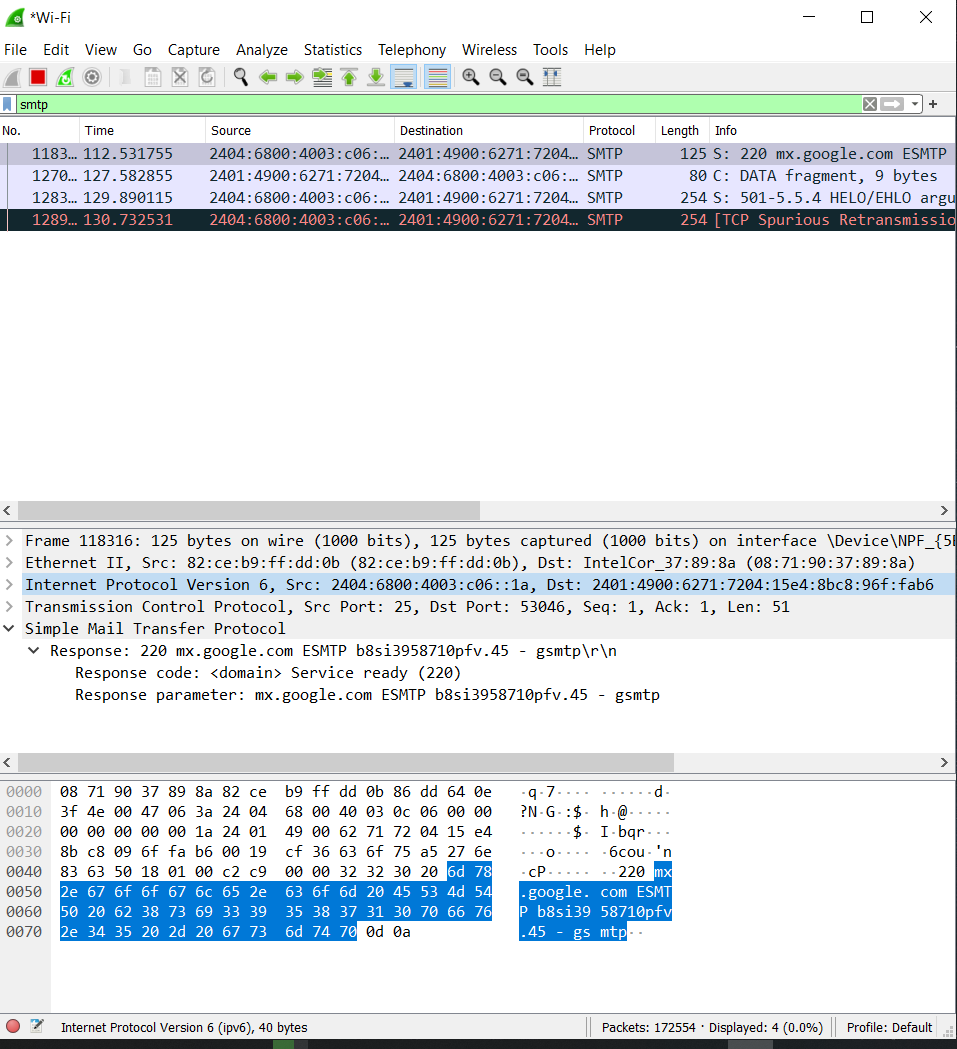
SETTING UP SMTP CONNECTION USING TELNET

****



SENDING AND RECEIVING MESSAGES USING SMTP - CLI:****

MESSAGES CAPTURED WHEN RUNNING THE SCAPY FILE:**NOTE:** CONNECTION TERMINATED AS TLS INITIATED, AND SINCE OUTLOOK WASN’T AVAILABLE IN THE PC.POP 3 WASN’T CAPTURED .IMAP RESULTS SAME AS SMTP.

WIRESHARK CAPTURE OF THE SMTP MESSAGES AND THE DATA BEING SNIFFED:****

**2.OS Fingerprinting:**

**Code:**

#!/usr/bin/python

from scapy.all import \*

import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

import sys

if len(sys.argv) != 2:

print ("The program will be performing OS fingerpriting via TTL capture analyisis")

'''sys.exit()'''

ip = sys.srgv[1]

ans = sr1(IP(dst=str(ip))/ICMP(),timeout=1,verbose=0)

if ans == None:

print ("Detection terminated!!")

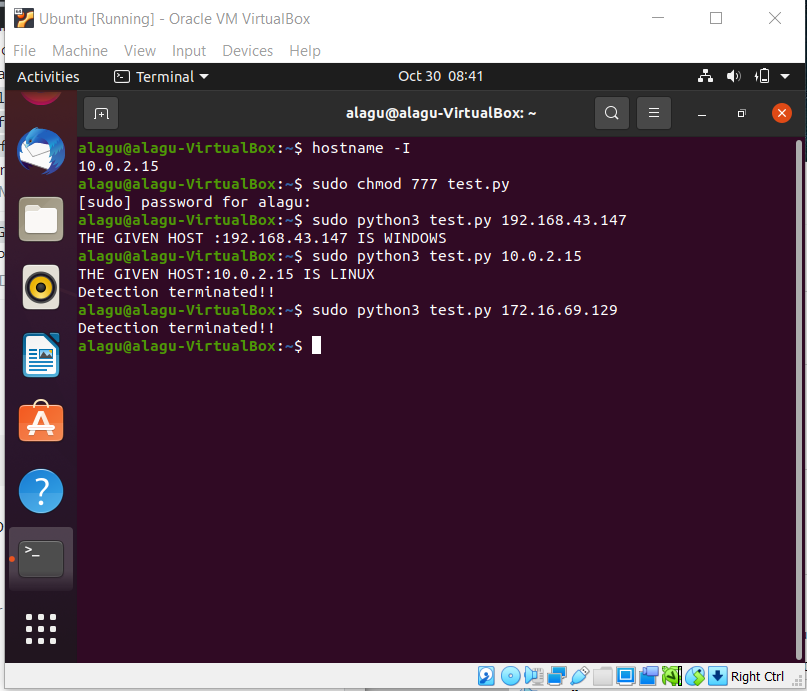
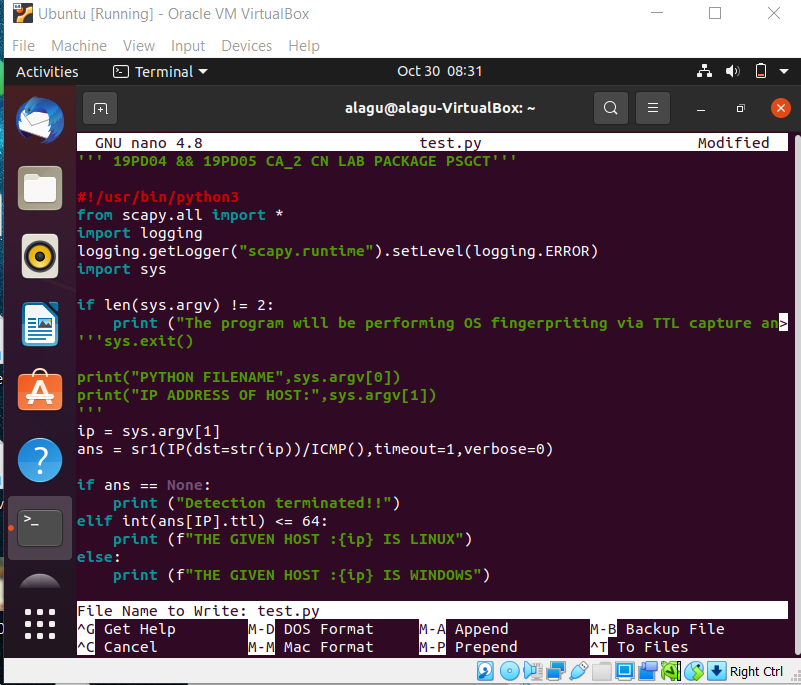
elif int(ans[IP].ttl) <= 64:

print ("THE GIVEN HOST [IP ADDRESS] IS LINUX")

else:

print ("THE GIVEN HOST [IP ADDRESS] IS WINDOWS")

**Output:**

****

**3. Broadcast scanning:**

It has a collect\_packet() callback function that sorts the packets in a dictionary called sorted\_data, depending on the protocol. If an IP is not already present in the dictionary, it creates a new entry. Otherwise, it updates the dictionary with the port number for that specific IP. Finally, it prints the IP addresses and ports in each line.

**CODE:**

from scapy.all import \*

import os

sorted\_data = dict()

END\_PORT = 443

def collect\_packet(packet):

if IP in packet:

#collect only if the packet is ip

if packet[IP].src not in sorted\_data:#check if already present in the dictionary

sorted\_data[packet[IP].src] = []

if TCP in packet:

#collect only if the packet is tcp

if packet[TCP].sport <= END\_PORT:

if not str(packet[TCP].sport) in sorted\_data[packet[IP].src]:

sorted\_data[packet[IP].src].append(str(packet[TCP].sport))

os.system('cls')

ip\_list = sorted(sorted\_data.keys())

for key in ip\_list:

ports=', '.join(sorted\_data[key])

if len (sorted\_data[key]) == 0:

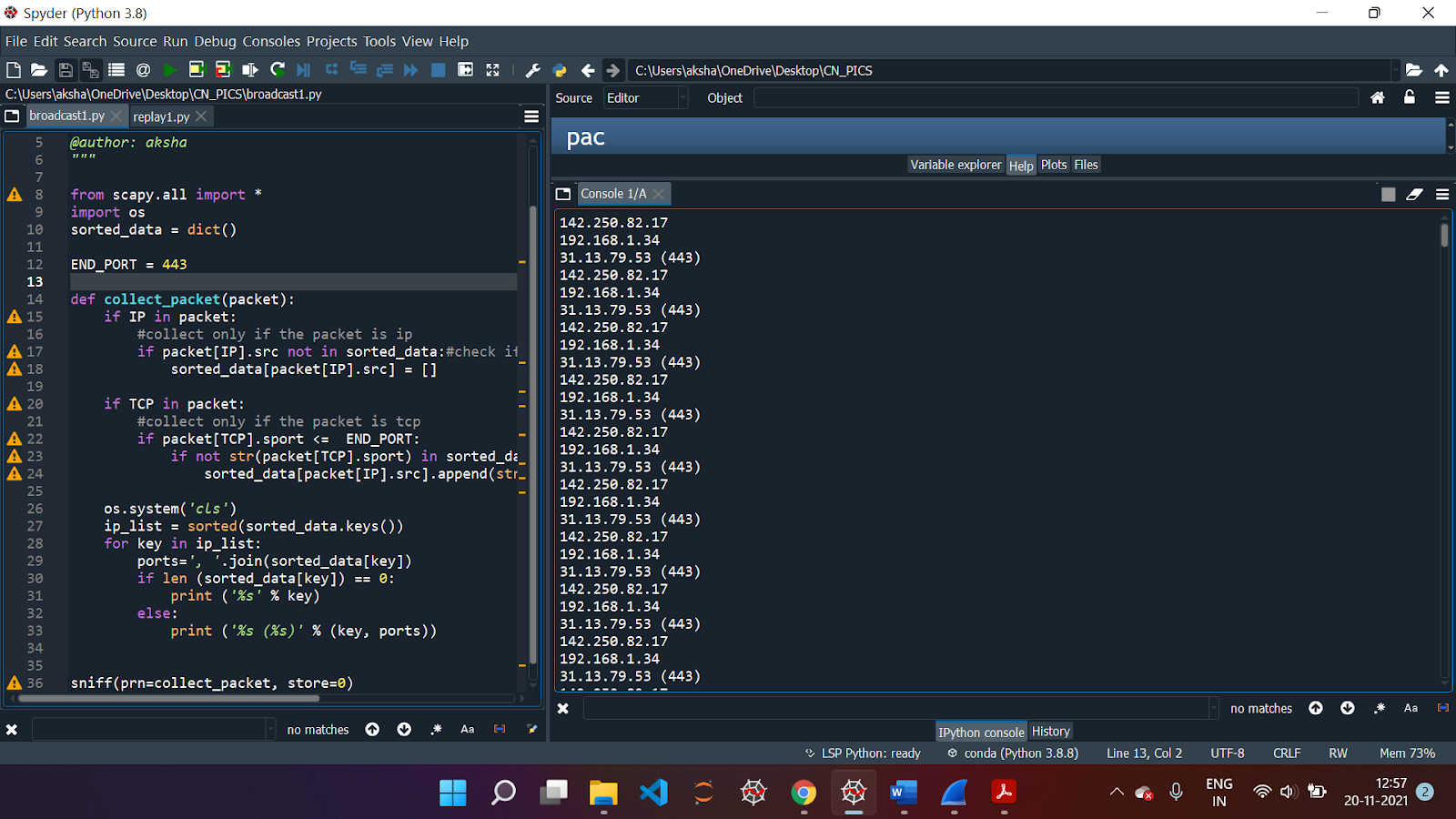
print ('%s' % key)

else:

print ('%s (%s)' % (key, ports))

if \_name\_ == '\_main\_':

sniff(prn=collect\_packet, store=0)



**4. Storing packets in pcap file and replaying traffic:**

We capture all the network packets and write them onto a file. The pkts list is written onto a file named pcapx. Now, sniff() is used with an offline parameter, which captures packets from the file instead of network. In each packet, the payload is modified, thus changing the source and destination IPs. In addition to this, the checksum packet is deleted as it was based on the original IP address.

**CODE:**

import argparse

from scapy.all import \*

packet = []

count = 0

pacnum = 0

def write\_cap(x):

global packet

global count

global pacnum

packet.append(x)

count += 1 #increment count when packet arrives

if count == 5:

pacnum += 1 #increment packet number

pname = "pcap%d.pcap" % pacnum

wrpcap(pname, packet) #write in pcap file

packet = []

count = 0

def output\_file():

print ("Checking presence of the pcap file...")

packet\_file = "./pcap1.pcap"

if os.path.exists(packet\_file):

print ("pcap fie %s found." %packet\_file)

packet = sniff(offline=packet\_file) #local scan

count = 0

while (count <=4):

print ("----Output of pcap pkt no:%s----" %count)

print (hexdump(packet[count]))

count += 1

else:

print ("pcap fie %s not found." %packet\_file)

def send\_packet(recvd\_pkt, src\_ip, dst\_ip, count):

pkt\_cnt = 0

p\_out = []

for p in recvd\_pkt:

pkt\_cnt += 1

new\_pkt = p.payload

new\_pkt[IP].dst = dst\_ip

new\_pkt[IP].src = src\_ip

del new\_pkt[IP].chksum

p\_out.append(new\_pkt)

if pkt\_cnt % count == 0:

send(PacketList(p\_out))

p\_out = []

# Send rest of packet

send(PacketList(p\_out))

print ("Total packets sent: %d" %pkt\_cnt)

if \_name\_ == '\_main\_':

print ("Started packet capturing and displaying... Press CTRL+C to stop")

sniff(prn=write\_cap)

output\_file()

# storing source ip, destination ip and count using commandline arguments

parser = argparse.ArgumentParser(description='Packet Sniffer')

parser.add\_argument('--infile', action="store", dest="infile", default='pcap1.pcap')

parser.add\_argument('--src-ip', action="store", dest="src\_ip", default='1.1.1.1')

parser.add\_argument('--dst-ip', action="store", dest="dst\_ip",default='2.2.2.2')

parser.add\_argument('--count', action="store", dest="count", default=100, type=int)

given\_args = ga = parser.parse\_args()

global src\_ip, dst\_ip

infile, src\_ip, dst\_ip, count = ga.infile, ga.src\_ip,ga.dst\_ip, ga.count

#check if pcap file present

try:

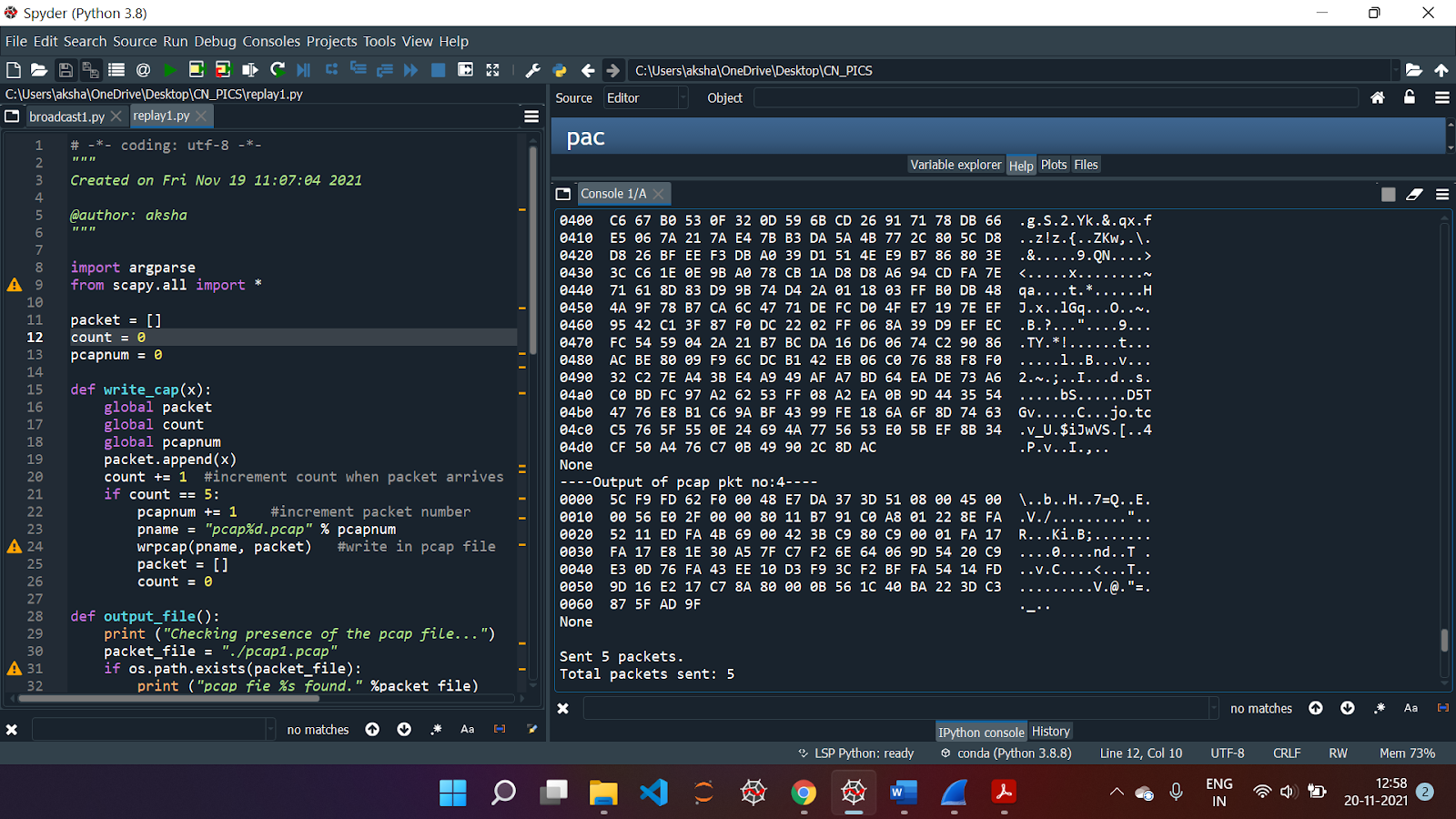
pkt\_reader = PcapReader(infile)

send\_packet(pkt\_reader, src\_ip, dst\_ip, count)

except IOError:

print ("Failed reading file %s contents" % infile)

sys.exit(1)

****