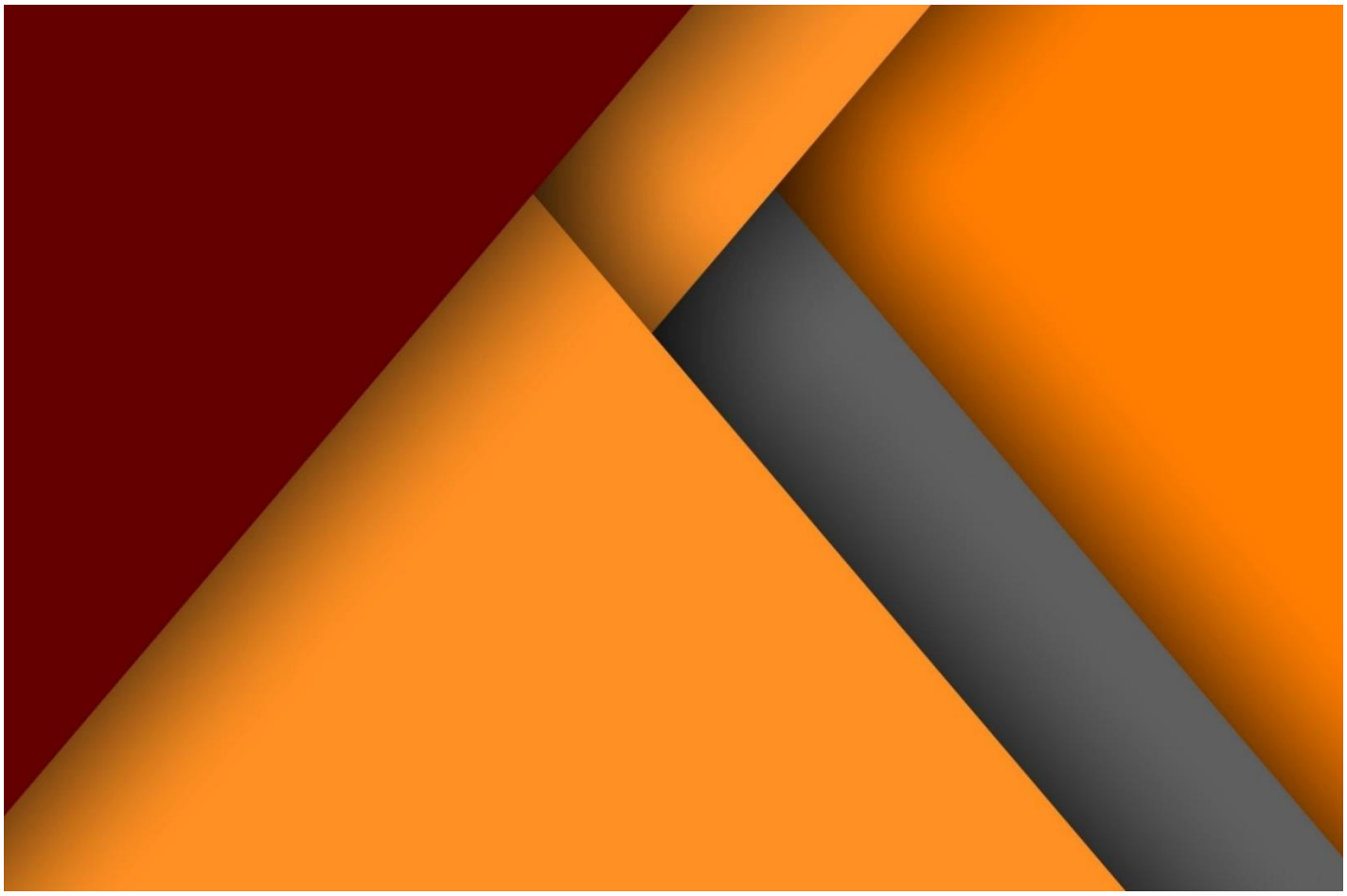

Computer Networks

Assignment-03

Section-S

Due: May 12, 2023

—

Mehmood ul Hassan

Course Instructor

Computer Networks

CS-3001

Group Members:

Zeeshan Ali

20i-2465

Ans Zeeshan

20i-0543

Assignment-03

In this assignment, we have to capture the live packets from our laptop or PC.
To capture the packets, we are independent to choose any of the programming language to implement this sceniro.

I use python language for this assignment and installed these dependencies to run my program.

```python
#!/usr/bin/env python
# coding: utf-8

import argparse
import os
import sys
import time
from scapy.utils import RawPcapReader
from scapy.layers.l2 import Ether
from scapy.layers.inet import IP, TCP

```

Now, explaining all the dependencies, to use argparse, I am able to run my program by command line and give some input values to my program at run time and using os and sys tells us about the operating system and system specification. To use the time library, it was for time stamp for the packets to calculate it and using the rest of library such as scapy, it is one of the best libraries to capture and sniffing the packets.

```python
#---
if __name__ == '__main__':
    print("Capturing packets.... & savingg into the file.")
    parser = argparse.ArgumentParser(description='PCAP reader')
    parser.add_argument('--pcap', metavar='packet.pcap',
                        help='pcap file to parse', required=True)
    args = parser.parse_args()

    file_name = 'packet.pcap'
    if not os.path.isfile(file_name):
        print('"{}" does not exist'.format(file_name), file=sys.stderr)
        sys.exit(-1)

    process_pcap(file_name)

    sys.exit(0)
```

This is main function of my code, where I capture the packets from my Laptop IP as a client and also capture the packets from my group partner Laptop Ip as server and save all the receiving packets into the file named as packet.pcap

```
11
12 def printable_timestamp(ts, resol):
13     ts_sec = ts // resol
14     ts_subsec = ts % resol
15     ts_sec_str = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(ts_sec))
16     return '{}.{}'.format(ts_sec_str, ts_subsec)
17
18 class PktDirection():
19     not_defined = 0
20     client_to_server = 1
21     server_to_client = 2
22
```

In this code, as the packets are being sent and receiving to each other, there will be a time stamp between the packets and packet direction defines that, either the packet is from the client side or server side.

```
22
23 def process_pcap(file_name):
24     print('Opening {}...'.format(file_name))
25
26     client = '192.168.1.137:8080'
27     server = '152.19.134.43:80'
28
29     (client_ip, client_port) = client.split(':')
30     (server_ip, server_port) = server.split(':')
31
32     count = 0
33     interesting_packet_count = 0
34
35     server_sequence_offset = None
36     client_sequence_offset = None
37
38     for (pkt_data, pkt_metadata,) in RawPcapReader(file_name):
39         count += 1
40
41         ether_pkt = Ether(pkt_data)
42         if 'type' not in ether_pkt.fields:
43             # LLC frames will have 'len' instead of 'type'.
44             # We disregard those
45             continue
46
47         if ether_pkt.type != 0x0800:
48             # disregard non-IPv4 packets
49             continue
50
51         ip_pkt = ether_pkt[IP]
52
53         if ip_pkt.proto != 6:
54             # Ignore non-TCP packet
55             continue
56
57         tcp_pkt = ip_pkt[TCP]
58
```

In this code snippet, the process-pcap function is used to iterate the packets to get useful information about packets and drop all the irrelevant packets. I have added the ip's of both pc's and marked their offset none because it should start from zero. In this code, it only separates the ethernet packets because I am implementing the TCP communication of IPV4 addresses and marked them as the Ip address.

```
58
59          direction = PktDirection.not_defined
60
61          if ip_pkt.src == client_ip:
62              if tcp_pkt.sport != int(client_port):
63                  continue
64              if ip_pkt.dst != server_ip:
65                  continue
66              if tcp_pkt.dport != int(server_port):
67                  continue
68              direction = PktDirection.client_to_server
69          elif ip_pkt.src == server_ip:
70              if tcp_pkt.sport != int(server_port):
71                  continue
72              if ip_pkt.dst != client_ip:
73                  continue
74              if tcp_pkt.dport != int(client_port):
75                  continue
76              direction = PktDirection.server_to_client
77          else:
78              continue
79
```

```
91
92          if direction == PktDirection.client_to_server:
93              if client_sequence_offset is None:
94                  client_sequence_offset = tcp_pkt.seq
95              relative_offset_seq = tcp_pkt.seq - client_sequence_offset
96          else:
97              assert direction == PktDirection.server_to_client
98              if server_sequence_offset is None:
99                  server_sequence_offset = tcp_pkt.seq
100             relative_offset_seq = tcp_pkt.seq - server_sequence_offset
101
```

In this code snippet, it just defines the packets source and destination ports of the client and server so that it communicates with each other, when the desired values match according to the packets it set the values and proceed it. It also updates the sequence number of the packets according to the arrival basis.

```
        interesting_packet_count += 1
        if interesting_packet_count == 1:
            first_pkt_timestamp = (pkt_metadata.tshigh << 32) | pkt_metadata.tslow
            first_pkt_timestamp_resolution = pkt_metadata.tsresol
            first_pkt_ordinal = count
```

In this code, it only tells the number of counts that how many packets are valid and that are still alive till the end of the communication.

```

        last_pkt_timestamp = (pkt_metadata.tshigh << 32) | pkt_metadata.tslow
        last_pkt_timestamp_resolution = pkt_metadata.tsresol
        last_pkt_ordinal = count

        this_pkt_relative_timestamp = last_pkt_timestamp - first_pkt_timestamp

```

This code will tell us about the details of the last packet, because while capturing it, the packets are too many packets to stop somewhere. I had use this to terminate the packet capturing.

```
        # If this TCP packet has the Ack bit set, then it must carry an ack number.
        if 'A' not in str(tcp_pkt.flags):
            relative_offset_ack = 0
        else:
            if direction == PktDirection.client_to_server:
                relative_offset_ack = tcp_pkt.ack - server_sequence_offset
            else:
                relative_offset_ack = tcp_pkt.ack - client_sequence_offset

        if (ip_pkt.flags == 'MF') or (ip_pkt.frag != 0):
            print('No support for fragmented IP packets')
            break

        tcp_payload_len = ip_pkt.len - (ip_pkt.ihl * 4) - (tcp_pkt.dataofs * 4)

        # Print
        fmt = '[{ordnl:>5}]{ts:>10.6f}s flag={flag:<3s} seq={seq:<9d} \
        ack={ack:<9d} len={len:<6d}'
        if direction == PktDirection.client_to_server:
            fmt = '{arrow}' + fmt
            arr = '-->'
        else:
            fmt = '{arrow:>69}' + fmt
            arr = '<--'
```

In this code snippet, the flags are assigned to the packets on the basis of their acknowledgement and if the packets would not be having the acknowledgement bit, it calculates, than assign a flag to it and at the end of this code, it just showing that how they have to print their information in the specific format.

```
126
127         print(fmt.format(arrow = arr,
128                          ordnl = last_pkt_ordinal,
129                          ts = this_pkt_relative_timestamp / pkt_metadata.tsresol,
130                          flag = str(tcp_pkt.flags),
131                          seq = relative_offset_seq,
132                          ack = relative_offset_ack,
133                          len = tcp_payload_len))
134     #---
135
136     print('{} contains {} packets ({} interesting)'.
137           format(file_name, count, interesting_packet_count))
138
139     print('First packet in connection: Packet #{} {}'.
140           format(first_pkt_ordinal,
141                  printable_timestamp(first_pkt_timestamp,
142                                      first_pkt_timestamp_resolution)))
143     print(' Last packet in connection: Packet #{} {}'.
144           format(last_pkt_ordinal,
145                  printable_timestamp(last_pkt_timestamp,
146                                      last_pkt_timestamp_resolution)))
```

In the last, it's just displayed the order in which they print it on console, arrow defines that from where it is coming. And the rest of variables are self-defined. In the last, it just shows the interesting packets count and first and last packet connection information.

```
                                                  <--[22551] 61.136741s flag=A   seq=52182721         ack=175      len=2880
-->[22552] 61.136758s flag=A   seq=175             ack=52185601  len=0
                                                  <--[22553] 61.137252s flag=A   seq=52185601         ack=175      len=7200
-->[22554] 61.137269s flag=A   seq=175             ack=52192801  len=0
                                                  <--[22555] 61.137349s flag=A   seq=52192801         ack=175      len=17280
-->[22556] 61.137363s flag=A   seq=175             ack=52210081  len=0
                                                  <--[22557] 61.137390s flag=A   seq=52210081         ack=175      len=5760
-->[22558] 61.137400s flag=A   seq=175             ack=52215841  len=0
                                                  <--[22559] 61.139254s flag=A   seq=52215841         ack=175      len=17280
-->[22560] 61.139286s flag=A   seq=175             ack=52233121  len=0
                                                  <--[22561] 61.139323s flag=A   seq=52233121         ack=175      len=5760
-->[22562] 61.139332s flag=A   seq=175             ack=52238881  len=0
                                                  <--[22563] 61.139493s flag=A   seq=52238881         ack=175      len=10080
-->[22564] 61.139509s flag=A   seq=175             ack=52248961  len=0
                                                  <--[22565] 61.142389s flag=A   seq=52248961         ack=175      len=11520
-->[22566] 61.142419s flag=A   seq=175             ack=52260481  len=0
                                                  <--[22567] 61.142485s flag=A   seq=52260481         ack=175      len=11520
-->[22568] 61.142497s flag=A   seq=175             ack=52272001  len=0
                                                  <--[22569] 61.142651s flag=A   seq=52272001         ack=175      len=10080
-->[22570] 61.142668s flag=A   seq=175             ack=52282081  len=0
                                                  <--[22571] 61.145454s flag=A   seq=52282081         ack=175      len=8640
-->[22572] 61.145484s flag=A   seq=175             ack=52290721  len=0
                                                  <--[22573] 61.145539s flag=A   seq=52290721         ack=175      len=11520
-->[22574] 61.145550s flag=A   seq=175             ack=52302241  len=0
                                                  <--[22575] 61.145725s flag=A   seq=52302241         ack=175      len=11520
-->[22576] 61.145739s flag=A   seq=175             ack=52313761  len=0
                                                  <--[22577] 61.145751s flag=A   seq=52313761         ack=175      len=1440
                                                  <--[22578] 61.147645s flag=PA  seq=52315201         ack=175      len=13483
-->[22579] 61.147676s flag=A   seq=175             ack=52328684  len=0
-->[22580] 61.148632s flag=FA  seq=175             ack=52328684  len=0
                                                  <--[22581] 61.440260s flag=FA  seq=52328684         ack=176      len=0
-->[22582] 61.440295s flag=A   seq=176             ack=52328685  len=0
packet.pcap contains 22639 packets (14975 interesting)
First packet in connection: Packet #2585 20:51:02.883718124
 Last packet in connection: Packet #22582 20:52:04.324012912
```

It is the output of my code, about the packet's header information as require.

References:

https://github.com/vnetman/scapy-pcap
https://github.com/vnetman/pcap-files

I take help from these GitHub repository and do many amendments according to my requirements and also learn the lectures to do this.