`

TCS 2351 Network Security

Trimester 1, 2023/2024

Lab Test: Packet Decode

Lecture Tutor: Dr. Chan Wai Kok

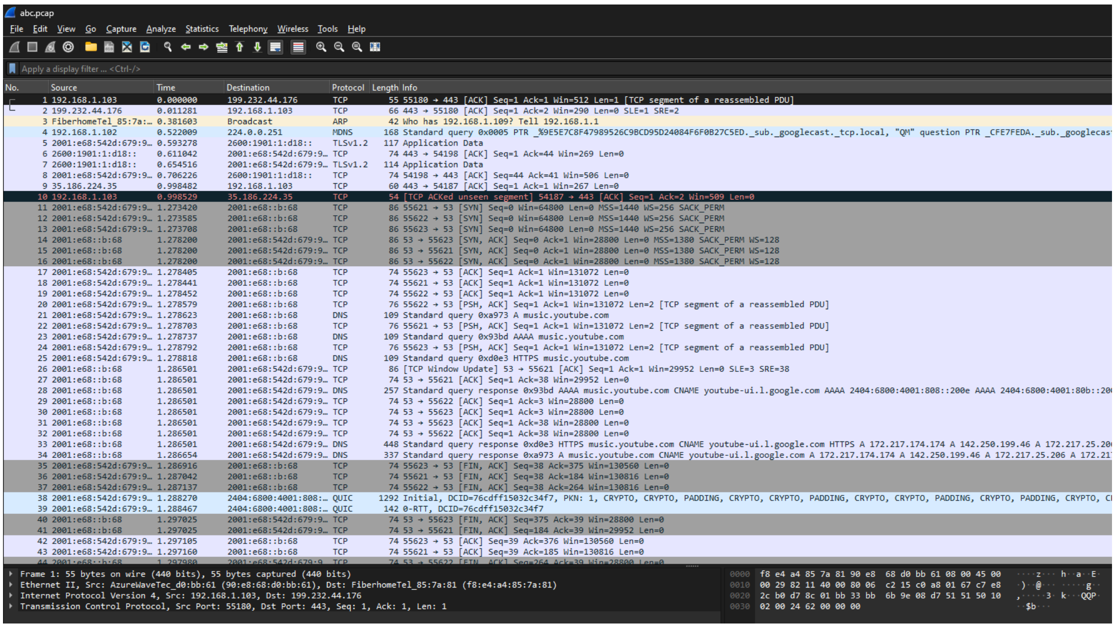
Group Members:

NAME	STUDENT ID
NUR AYU AMIRA BINTI IDRIS	1201200722

Question 2 : Packet Corruption

- Your task is to read the file “abc”. Locate all IP packets and corrupt the IP packet field such as TTL = 0, protocol = unknown, source add = destination add, source add = IP Multicast address, IP data length mismatch with UDP data length etc. The input to the corruption should be user-specified and not fixed inside the program. The packet generator will use the file to send the packet to various hosts and see how its OS reacts to the problem. Store the output into a file “xyz”.

1) Capture the packet in Wireshark and save as “abc.pcap”

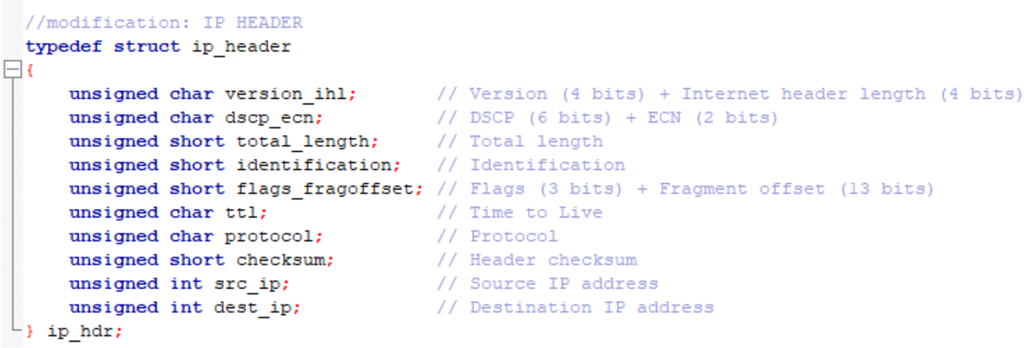

abc.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Source                 Time      Destination            Protocol Length Info
   1 192.168.1.103          0.000000  199.232.44.176         TCP        55 55180 → 443 [ACK] Seq=1 Ack=1 Win=512 Len=1 [TCP segment of a reassembled PDU]
   2 199.232.44.176         0.011281  192.168.1.103          TCP        66 443 → 55180 [ACK] Seq=1 Ack=2 Win=290 Len=0 SLE=1 SRE=2
   3 FiberhomeTel_85:7a:…   0.381603  Broadcast              ARP        42 Who has 192.168.1.109? Tell 192.168.1.1
   4 192.168.1.102          0.522009  224.0.0.251            MDNS      168 Standard query 0x0005 PTR _%9E5E7C8F47989526C9BCD95D24084F6F0B27C5ED._sub._googlecast._tcp.local, "QM" question PTR _CFE7FEDA._sub._googlecas
   5 2001:e68:542d:679:9…   0.593278  2600:1901:1:d18::      TLSv1.2   117 Application Data
   6 2600:1901:1:d18::      0.611042  2001:e68:542d:679:9…   TCP        74 443 → 54198 [ACK] Seq=1 Ack=44 Win=269 Len=0
   7 2600:1901:1:d18::      0.654516  2001:e68:542d:679:9…   TLSv1.2   114 Application Data
   8 2001:e68:542d:679:9…   0.706226  2600:1901:1:d18::      TCP        74 54198 → 443 [ACK] Seq=44 Ack=41 Win=506 Len=0
   9 35.186.224.35          0.998482  192.168.1.103          TCP        60 443 → 54187 [ACK] Seq=1 Ack=1 Win=267 Len=0
  10 192.168.1.103          0.998529  35.186.224.35          TCP        54 [TCP ACKed unseen segment] 54187 → 443 [ACK] Seq=1 Ack=2 Win=509 Len=0
  11 2001:e68:542d:679:9…   1.273420  2001:e68::b:68         TCP        86 55621 → 53 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
  12 2001:e68:542d:679:9…   1.273585  2001:e68::b:68         TCP        86 55622 → 53 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
  13 2001:e68:542d:679:9…   1.273708  2001:e68::b:68         TCP        86 55623 → 53 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
  14 2001:e68::b:68         1.278200  2001:e68:542d:679:9…   TCP        86 53 → 55623 [SYN, ACK] Seq=0 Ack=1 Win=28800 Len=0 MSS=1380 SACK_PERM WS=128
  15 2001:e68::b:68         1.278200  2001:e68:542d:679:9…   TCP        86 53 → 55621 [SYN, ACK] Seq=0 Ack=1 Win=28800 Len=0 MSS=1380 SACK_PERM WS=128
  16 2001:e68::b:68         1.278200  2001:e68:542d:679:9…   TCP        86 53 → 55622 [SYN, ACK] Seq=0 Ack=1 Win=28800 Len=0 MSS=1380 SACK_PERM WS=128
  17 2001:e68:542d:679:9…   1.278405  2001:e68::b:68         TCP        74 55623 → 53 [ACK] Seq=1 Ack=1 Win=131072 Len=0
  18 2001:e68:542d:679:9…   1.278441  2001:e68::b:68         TCP        74 55621 → 53 [ACK] Seq=1 Ack=1 Win=131072 Len=0
  19 2001:e68:542d:679:9…   1.278452  2001:e68::b:68         TCP        74 55622 → 53 [ACK] Seq=1 Ack=1 Win=131072 Len=0
  20 2001:e68:542d:679:9…   1.278579  2001:e68::b:68         TCP        76 55622 → 53 [PSH, ACK] Seq=1 Ack=1 Win=131072 Len=2 [TCP segment of a reassembled PDU]
  21 2001:e68:542d:679:9…   1.278623  2001:e68::b:68         DNS       109 Standard query 0xa973 A music.youtube.com
  22 2001:e68:542d:679:9…   1.278703  2001:e68::b:68         TCP        76 55621 → 53 [PSH, ACK] Seq=1 Ack=1 Win=131072 Len=2 [TCP segment of a reassembled PDU]
  23 2001:e68:542d:679:9…   1.278737  2001:e68::b:68         DNS       109 Standard query 0x93bd AAAA music.youtube.com
  24 2001:e68:542d:679:9…   1.278792  2001:e68::b:68         TCP        76 55623 → 53 [PSH, ACK] Seq=1 Ack=1 Win=131072 Len=2 [TCP segment of a reassembled PDU]
  25 2001:e68:542d:679:9…   1.278818  2001:e68::b:68         DNS       109 Standard query 0xd0e3 HTTPS music.youtube.com
  26 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   TCP        86 [TCP Window Update] 53 → 55621 [ACK] Seq=1 Ack=1 Win=29952 Len=0 SLE=3 SRE=38
  27 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   TCP        74 53 → 55621 [ACK] Seq=1 Ack=38 Win=29952 Len=0
  28 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   DNS       257 Standard query response 0x93bd AAAA music.youtube.com CNAME youtube-ui.l.google.com AAAA 2404:6800:4001:808::200e AAAA 2404:6800:4001:80b::20
  29 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   TCP        74 53 → 55622 [ACK] Seq=1 Ack=3 Win=28800 Len=0
  30 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   TCP        74 53 → 55623 [ACK] Seq=1 Ack=3 Win=28800 Len=0
  31 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   TCP        74 53 → 55623 [ACK] Seq=1 Ack=38 Win=28800 Len=0
  32 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   TCP        74 53 → 55622 [ACK] Seq=1 Ack=38 Win=28800 Len=0
  33 2001:e68::b:68         1.286501  2001:e68:542d:679:9…   DNS       448 Standard query response 0xd0e3 HTTPS music.youtube.com CNAME youtube-ui.l.google.com HTTPS A 172.217.174.174 A 142.250.199.46 A 172.217.25.20
  34 2001:e68::b:68         1.286654  2001:e68:542d:679:9…   DNS       337 Standard query response 0xa973 A music.youtube.com CNAME youtube-ui.l.google.com A 172.217.174.174 A 142.250.199.46 A 172.217.25.206 A 172.21
  35 2001:e68:542d:679:9…   1.286916  2001:e68::b:68         TCP        74 55623 → 53 [FIN, ACK] Seq=38 Ack=375 Win=130560 Len=0
  36 2001:e68:542d:679:9…   1.287042  2001:e68::b:68         TCP        74 55621 → 53 [FIN, ACK] Seq=38 Ack=184 Win=130816 Len=0
  37 2001:e68:542d:679:9…   1.287137  2001:e68::b:68         TCP        74 55622 → 53 [FIN, ACK] Seq=38 Ack=264 Win=130816 Len=0
  38 2001:e68:542d:679:9…   1.288270  2404:6800:4001:808:…   QUIC     1292 Initial, DCID=76cdff15032c34f7, PKN: 1, CRYPTO, CRYPTO, PADDING, CRYPTO, CRYPTO, PADDING, CRYPTO, CRYPTO, PADDING, CRYPTO, PADDING, CRYPTO, C
  39 2001:e68:542d:679:9…   1.288467  2404:6800:4001:808:…   QUIC      142 0-RTT, DCID=76cdff15032c34f7
  40 2001:e68::b:68         1.297025  2001:e68:542d:679:9…   TCP        74 53 → 55623 [FIN, ACK] Seq=375 Ack=39 Win=28800 Len=0
  41 2001:e68::b:68         1.297025  2001:e68:542d:679:9…   TCP        74 53 → 55621 [FIN, ACK] Seq=184 Ack=39 Win=29952 Len=0
  42 2001:e68:542d:679:9…   1.297105  2001:e68::b:68         TCP        74 55623 → 53 [ACK] Seq=39 Ack=376 Win=130560 Len=0
  43 2001:e68:542d:679:9…   1.297160  2001:e68::b:68         TCP        74 55621 → 53 [ACK] Seq=39 Ack=185 Win=130816 Len=0
  44 2001:e68::b:68         1.297980  2001:e68:542d:679:9…   TCP        74 53 → 55622 [FIN, ACK] Seq=264 Ack=39 Win=28800 Len=0
```

```
▸ Frame 1: 55 bytes on wire (440 bits), 55 bytes captured (440 bits)
▸ Ethernet II, Src: AzureWaveTec_d0:bb:61 (90:e8:68:d0:bb:61), Dst: FiberhomeTel_85:7a:81 (f8:e4:a4:85:7a:81)
▸ Internet Protocol Version 4, Src: 192.168.1.103, Dst: 199.232.44.176
▸ Transmission Control Protocol, Src Port: 55180, Dst Port: 443, Seq: 1, Ack: 1, Len: 1
```

```
0000  f8 e4 a4 85 7a 81 90 e8  68 d0 bb 61 08 00 45 00   ····z··· h··a··E·
0010  00 29 82 11 40 00 80 06  c2 15 c0 a8 01 67 c7 e8   ·)··@··· ·····g··
0020  2c b0 d7 8c 01 bb 33 bb  6b 9e 08 d7 51 51 50 10   ,·····3· k···QQP·
0030  02 00 24 62 00 00 00                               ··$b···
```

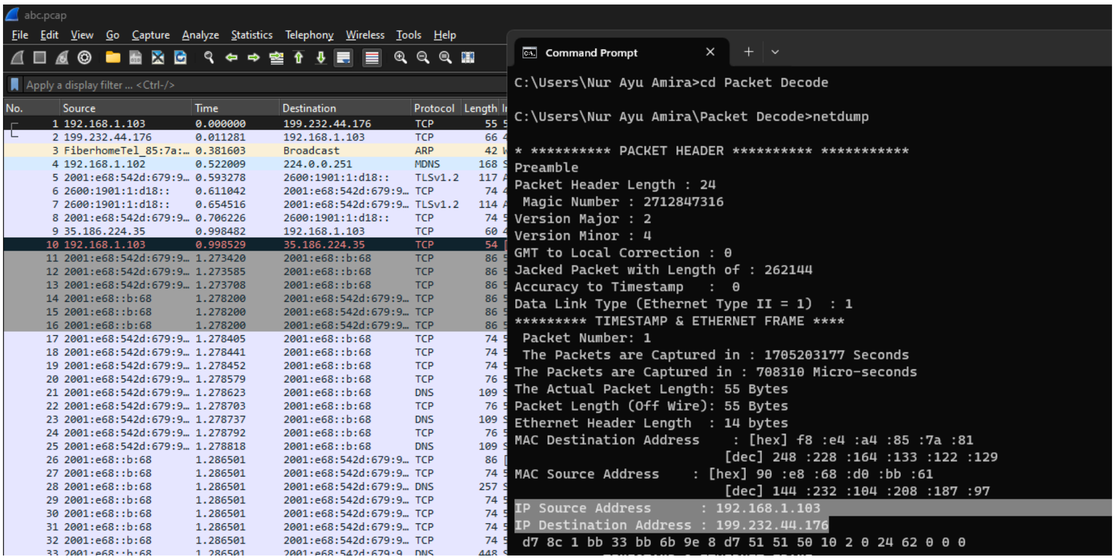
2) Add code for capture IP Source Address and Destination Address



```
//modification: IP HEADER
typedef struct ip_header
{
    unsigned char version_ihl;        // Version (4 bits) + Internet header length (4 bits)
    unsigned char dscp_ecn;           // DSCP (6 bits) + ECN (2 bits)
    unsigned short total_length;      // Total length
    unsigned short identification;    // Identification
    unsigned short flags_fragoffset;  // Flags (3 bits) + Fragment offset (13 bits)
    unsigned char ttl;                // Time to Live
    unsigned char protocol;           // Protocol
    unsigned short checksum;          // Header checksum
    unsigned int src_ip;              // Source IP address
    unsigned int dest_ip;             // Destination IP address
} ip_hdr;
```

```
    //modification: READ AND DISPLAY IP HEADER INFORMATION
fread((char *)&ip, sizeof(ip), 1, input);

cout << "IP Source Address      : " << ((ip.src_ip >> 0) & 0xFF) << "." << ((ip.src_ip >> 8) & 0xFF)
     << "." << ((ip.src_ip >> 16) & 0xFF) << "." << ((ip.src_ip >> 24) & 0xFF) << endl;

cout << "IP Destination Address : " << ((ip.dest_ip >> 0) & 0xFF) << "." << ((ip.dest_ip >> 8) & 0xFF)
     << "." << ((ip.dest_ip >> 16) & 0xFF) << "." << ((ip.dest_ip >> 24) & 0xFF) << endl;
```

3) Run netdump in Command Prompt to check IP Source Address and Destination Address same as in Wireshark or not?



abc.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Command Prompt

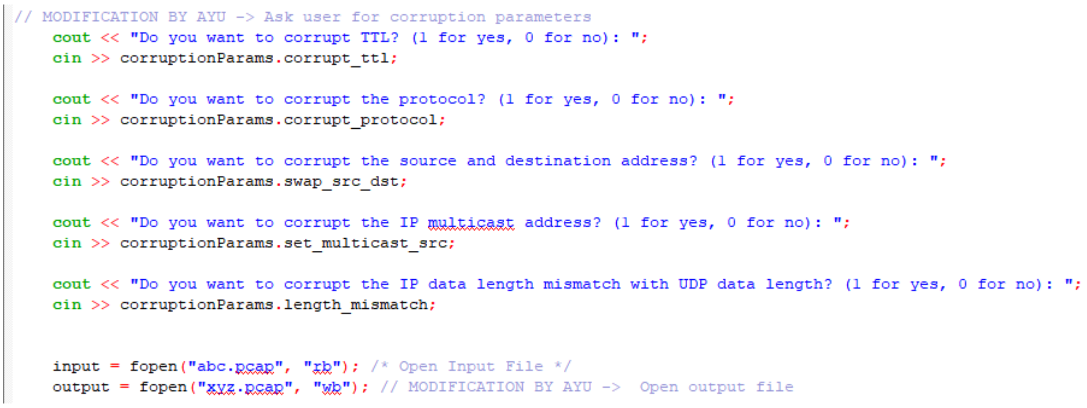
No.	Source	Time	Destination	Protocol	Length
1	192.168.1.103	0.000000	199.232.44.176	TCP	55
2	199.232.44.176	0.011281	192.168.1.103	TCP	66
3	FiberhomeTel_85:7a:…	0.381603	Broadcast	ARP	42
4	192.168.1.102	0.522009	224.0.0.251	MDNS	168
5	2001:e68:542d:679:9…	0.593278	2600:1901:1:d18::	TLSv1.2	117
6	2600:1901:1:d18::	0.611042	2001:e68:542d:679:9…	TCP	74
7	2600:1901:1:d18::	0.654516	2001:e68:542d:679:9…	TLSv1.2	114
8	2001:e68:542d:679:9…	0.706226	2600:1901:1:d18::	TCP	74
9	35.186.224.35	0.998482	192.168.1.103	TCP	60
10	192.168.1.103	0.998529	35.186.224.35	TCP	54
11	2001:e68:542d:679:9…	1.273420	2001:e68::b:68	TCP	86
12	2001:e68:542d:679:9…	1.273585	2001:e68::b:68	TCP	86
13	2001:e68:542d:679:9…	1.273708	2001:e68::b:68	TCP	86
14	2001:e68::b:68	1.278200	2001:e68:542d:679:9…	TCP	86
15	2001:e68::b:68	1.278200	2001:e68:542d:679:9…	TCP	86
16	2001:e68::b:68	1.278200	2001:e68:542d:679:9…	TCP	86
17	2001:e68:542d:679:9…	1.278405	2001:e68::b:68	TCP	74
18	2001:e68:542d:679:9…	1.278441	2001:e68::b:68	TCP	74
19	2001:e68:542d:679:9…	1.278452	2001:e68::b:68	TCP	74
20	2001:e68:542d:679:9…	1.278579	2001:e68::b:68	TCP	76
21	2001:e68:542d:679:9…	1.278623	2001:e68::b:68	DNS	109
22	2001:e68:542d:679:9…	1.278703	2001:e68::b:68	TCP	76
23	2001:e68:542d:679:9…	1.278737	2001:e68::b:68	DNS	109
24	2001:e68:542d:679:9…	1.278792	2001:e68::b:68	TCP	76
25	2001:e68:542d:679:9…	1.278818	2001:e68::b:68	DNS	109
26	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	TCP	86
27	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	TCP	74
28	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	DNS	257
29	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	TCP	74
30	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	TCP	74
31	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	TCP	74
32	2001:e68::b:68	1.286501	2001:e68:542d:679:9…	TCP	74
33	2001:e68::b:68	1.286501	2001:e68:542d:679:9	DNS	448

```
C:\Users\Nur Ayu Amira>cd Packet Decode

C:\Users\Nur Ayu Amira\Packet Decode>netdump

* ********** PACKET HEADER ********** ***********
Preamble
Packet Header Length : 24
 Magic Number : 2712847316
Version Major : 2
Version Minor : 4
GMT to Local Correction : 0
Jacked Packet with Length of : 262144
Accuracy to Timestamp   :  0
Data Link Type (Ethernet Type II = 1)  : 1
********* TIMESTAMP & ETHERNET FRAME ****
 Packet Number: 1
 The Packets are Captured in : 1705203177 Seconds
The Packets are Captured in : 708310 Micro-seconds
The Actual Packet Length: 55 Bytes
Packet Length (Off Wire): 55 Bytes
Ethernet Header Length  : 14 bytes
MAC Destination Address    : [hex] f8 :e4 :a4 :85 :7a :81
                          [dec] 248 :228 :164 :133 :122 :129
MAC Source Address    : [hex] 90 :e8 :68 :d0 :bb :61
                          [dec] 144 :232 :104 :208 :187 :97
IP Source Address      : 192.168.1.103
IP Destination Address : 199.232.44.176
 d7 8c 1 bb 33 bb 6b 9e 8 d7 51 51 50 10 2 0 24 62 0 0 0
```

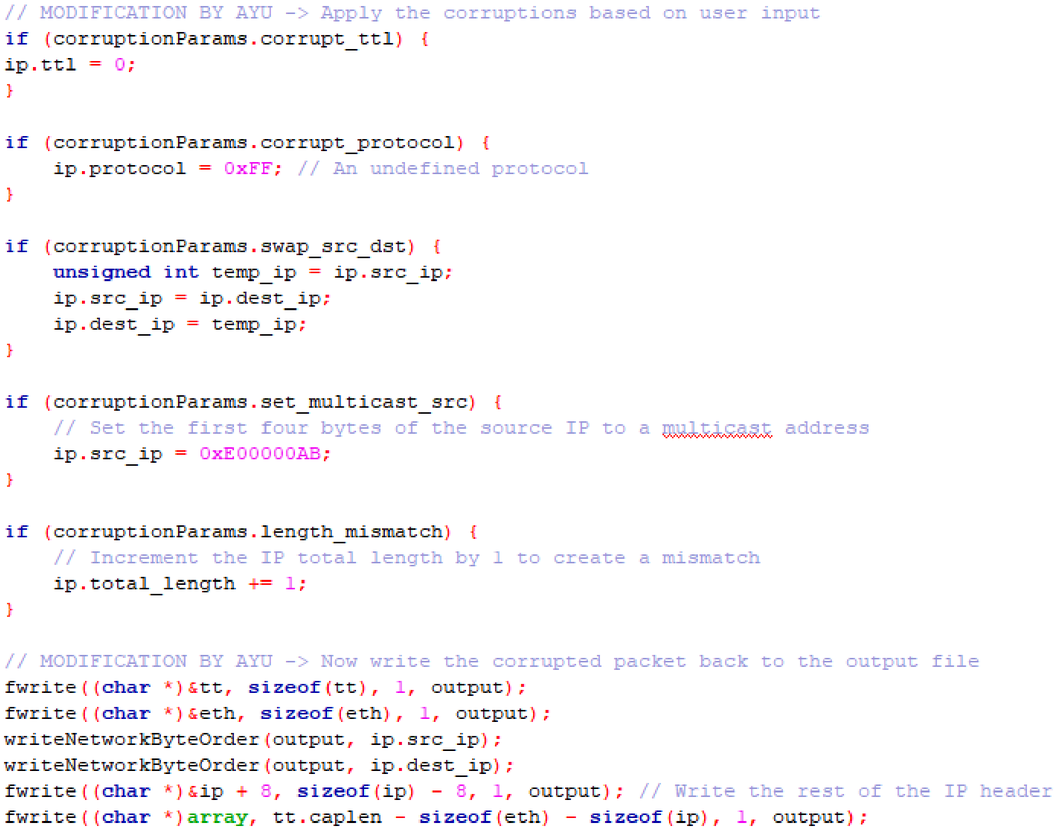
- We can see the IP Source Address and Destination Address that we run in the command prompt is the **same** as in Wireshark. So, **it's work to locate all the ip packets.**

4) Next, I need to corrupt the IP packet field such as TTL = 0, protocol = unknown, source add = destination add, source add = IP Multicast address, IP data length mismatch with UDP data length etc. The input to the corruption should be user-specified and not fixed inside the program.

```
// MODIFICATION BY AYU -> Define corruption parameters structure
struct CorruptionParams {
    bool corrupt_ttl;
    bool corrupt_protocol;
    bool swap_src_dst;
    bool set_multicast_src;
    bool length_mismatch;
} corruptionParams;

// MODIFICATION BY AYU -> Helper function to write an integer in network b
void writeNetworkByteOrder(FILE *file, unsigned int value) {
    unsigned char bytes[4];
    bytes[0] = (value >> 24) & 0xFF;
    bytes[1] = (value >> 16) & 0xFF;
    bytes[2] = (value >> 8) & 0xFF;
    bytes[3] = value & 0xFF;
    fwrite(bytes, sizeof(bytes), 1, file);
}
```

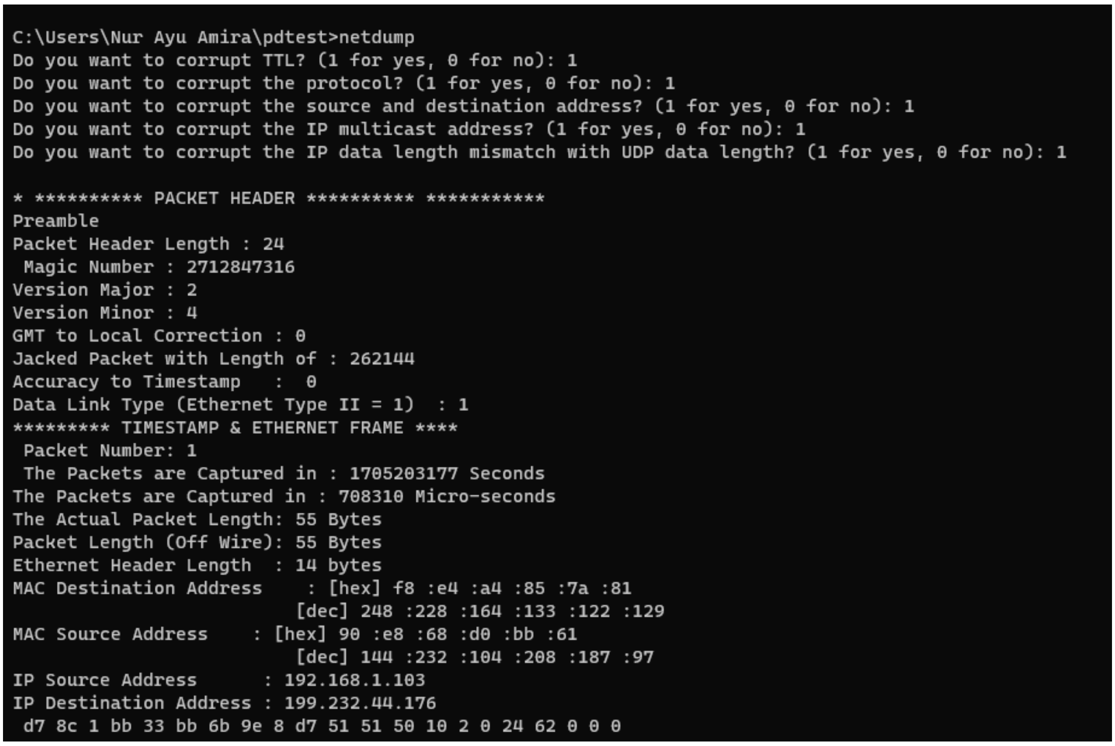


```
// MODIFICATION BY AYU -> Ask user for corruption parameters
    cout << "Do you want to corrupt TTL? (1 for yes, 0 for no): ";
    cin >> corruptionParams.corrupt_ttl;

    cout << "Do you want to corrupt the protocol? (1 for yes, 0 for no): ";
    cin >> corruptionParams.corrupt_protocol;

    cout << "Do you want to corrupt the source and destination address? (1 for yes, 0 for no): ";
    cin >> corruptionParams.swap_src_dst;

    cout << "Do you want to corrupt the IP multicast address? (1 for yes, 0 for no): ";
    cin >> corruptionParams.set_multicast_src;

    cout << "Do you want to corrupt the IP data length mismatch with UDP data length? (1 for yes, 0 for no): ";
    cin >> corruptionParams.length_mismatch;


    input = fopen("abc.pcap", "rb"); /* Open Input File */
    output = fopen("xyz.pcap", "wb"); // MODIFICATION BY AYU ->  Open output file
```

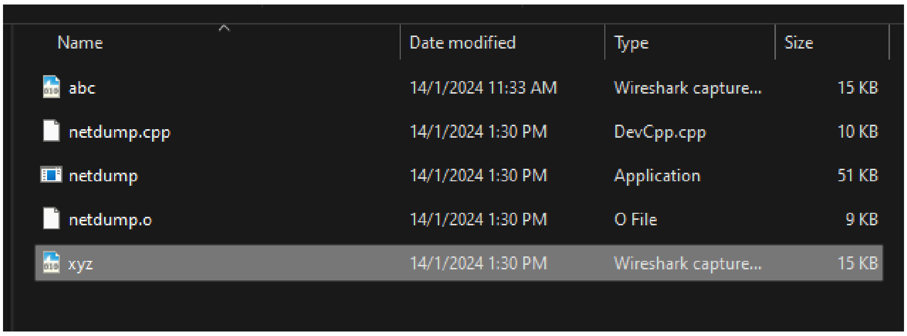


```
// MODIFICATION BY AYU -> Apply the corruptions based on user input
if (corruptionParams.corrupt_ttl) {
ip.ttl = 0;
}

if (corruptionParams.corrupt_protocol) {
    ip.protocol = 0xFF; // An undefined protocol
}

if (corruptionParams.swap_src_dst) {
    unsigned int temp_ip = ip.src_ip;
    ip.src_ip = ip.dest_ip;
    ip.dest_ip = temp_ip;
}

if (corruptionParams.set_multicast_src) {
    // Set the first four bytes of the source IP to a multicast address
    ip.src_ip = 0xE00000AB;
}

if (corruptionParams.length_mismatch) {
    // Increment the IP total length by 1 to create a mismatch
    ip.total_length += 1;
}

// MODIFICATION BY AYU -> Now write the corrupted packet back to the output file
fwrite((char *)&tt, sizeof(tt), 1, output);
fwrite((char *)&eth, sizeof(eth), 1, output);
writeNetworkByteOrder(output, ip.src_ip);
writeNetworkByteOrder(output, ip.dest_ip);
fwrite((char *)&ip + 8, sizeof(ip) - 8, 1, output); // Write the rest of the IP header
fwrite((char *)array, tt.caplen - sizeof(eth) - sizeof(ip), 1, output);
```

5) **Now, I need to run the code again in command prompt to make it the corrupt file.**



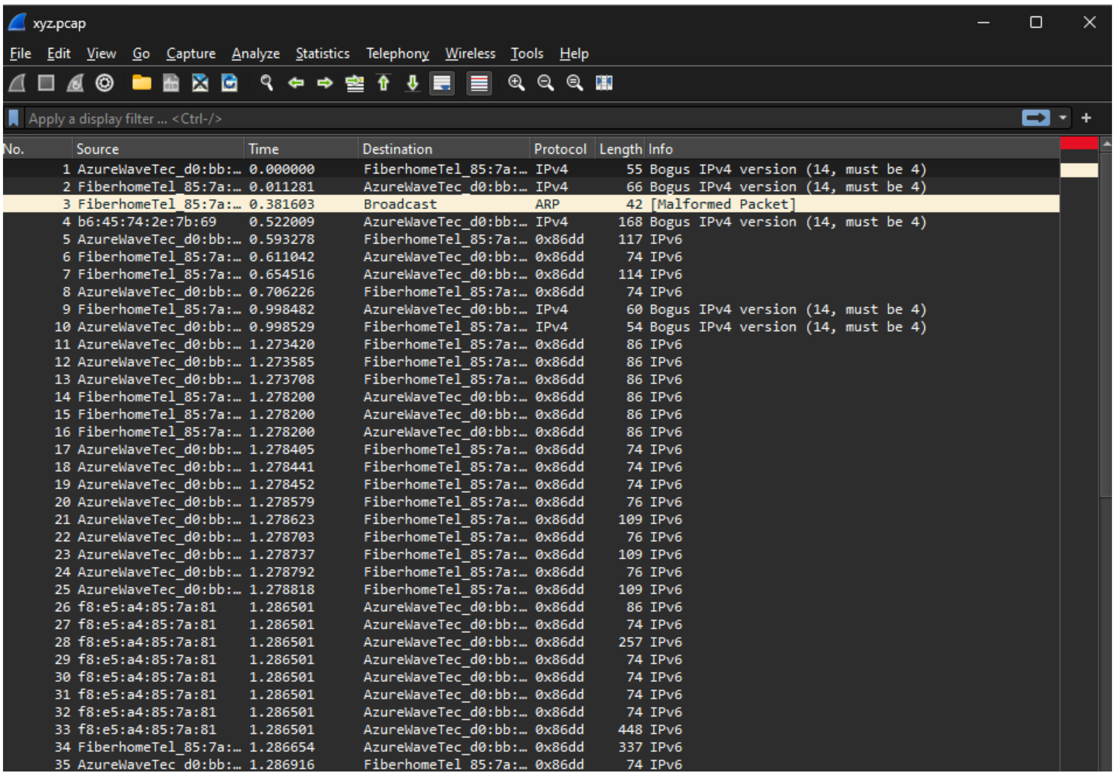
```
C:\Users\Nur Ayu Amira\pdtest>netdump
Do you want to corrupt TTL? (1 for yes, 0 for no): 1
Do you want to corrupt the protocol? (1 for yes, 0 for no): 1
Do you want to corrupt the source and destination address? (1 for yes, 0 for no): 1
Do you want to corrupt the IP multicast address? (1 for yes, 0 for no): 1
Do you want to corrupt the IP data length mismatch with UDP data length? (1 for yes, 0 for no): 1

* ********** PACKET HEADER ********** ***********
Preamble
Packet Header Length : 24
 Magic Number : 2712847316
Version Major : 2
Version Minor : 4
GMT to Local Correction : 0
Jacked Packet with Length of : 262144
Accuracy to Timestamp   :  0
Data Link Type (Ethernet Type II = 1)  : 1
********* TIMESTAMP & ETHERNET FRAME ****
 Packet Number: 1
 The Packets are Captured in : 1705203177 Seconds
The Packets are Captured in : 708310 Micro-seconds
The Actual Packet Length: 55 Bytes
Packet Length (Off Wire): 55 Bytes
Ethernet Header Length  : 14 bytes
MAC Destination Address    : [hex] f8 :e4 :a4 :85 :7a :81
                          [dec] 248 :228 :164 :133 :122 :129
MAC Source Address    : [hex] 90 :e8 :68 :d0 :bb :61
                          [dec] 144 :232 :104 :208 :187 :97
IP Source Address      : 192.168.1.103
IP Destination Address : 199.232.44.176
 d7 8c 1 bb 33 bb 6b 9e 8 d7 51 51 50 10 2 0 24 62 0 0 0
```



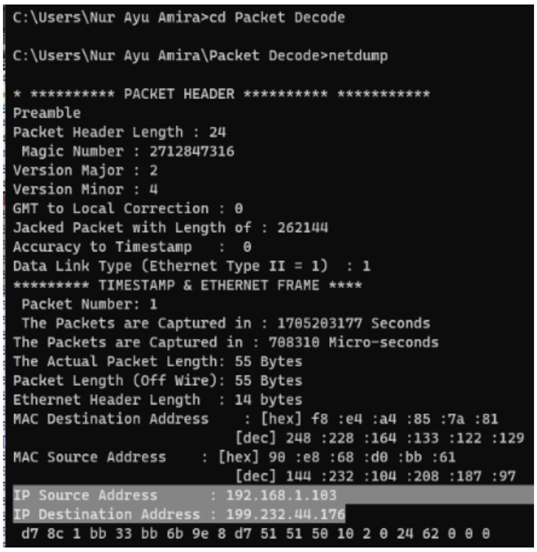
Name	Date modified	Type	Size
abc	14/1/2024 11:33 AM	Wireshark capture...	15 KB
netdump.cpp	14/1/2024 1:30 PM	DevCpp.cpp	10 KB
netdump	14/1/2024 1:30 PM	Application	51 KB
netdump.o	14/1/2024 1:30 PM	O File	9 KB
xyz	14/1/2024 1:30 PM	Wireshark capture...	15 KB

- It will **store** the output into a **file "xyz.pcap".**

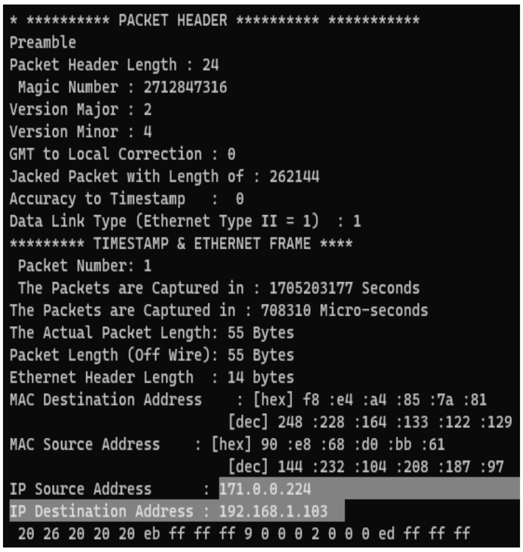
6) **Now we open the “xyz.pcap” and compare it with “abc.pcap”**


xyz.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Source	Time	Destination	Protocol	Length	Info
1	AzureWaveTec_d0:bb:…	0.000000	FiberhomeTel_85:7a:…	IPv4	55	Bogus IPv4 version (14, must be 4)
2	FiberhomeTel_85:7a:…	0.011281	AzureWaveTec_d0:bb:…	IPv4	66	Bogus IPv4 version (14, must be 4)
3	FiberhomeTel_85:7a:…	0.381603	Broadcast	ARP	42	[Malformed Packet]
4	b6:45:74:2e:7b:69	0.522009	AzureWaveTec_d0:bb:…	IPv4	168	Bogus IPv4 version (14, must be 4)
5	AzureWaveTec_d0:bb:…	0.593278	FiberhomeTel_85:7a:…	0x86dd	117	IPv6
6	FiberhomeTel_85:7a:…	0.611042	AzureWaveTec_d0:bb:…	0x86dd	74	IPv6
7	FiberhomeTel_85:7a:…	0.654516	AzureWaveTec_d0:bb:…	0x86dd	114	IPv6
8	AzureWaveTec_d0:bb:…	0.706226	FiberhomeTel_85:7a:…	0x86dd	74	IPv6
9	FiberhomeTel_85:7a:…	0.998482	AzureWaveTec_d0:bb:…	IPv4	60	Bogus IPv4 version (14, must be 4)
10	AzureWaveTec_d0:bb:…	0.998529	FiberhomeTel_85:7a:…	IPv4	54	Bogus IPv4 version (14, must be 4)
11	AzureWaveTec_d0:bb:…	1.273420	FiberhomeTel_85:7a:…	0x86dd	86	IPv6
12	AzureWaveTec_d0:bb:…	1.273585	FiberhomeTel_85:7a:…	0x86dd	86	IPv6
13	AzureWaveTec_d0:bb:…	1.273708	FiberhomeTel_85:7a:…	0x86dd	86	IPv6
14	FiberhomeTel_85:7a:…	1.278200	AzureWaveTec_d0:bb:…	0x86dd	86	IPv6
15	FiberhomeTel_85:7a:…	1.278200	AzureWaveTec_d0:bb:…	0x86dd	86	IPv6
16	FiberhomeTel_85:7a:…	1.278200	AzureWaveTec_d0:bb:…	0x86dd	86	IPv6
17	AzureWaveTec_d0:bb:…	1.278405	FiberhomeTel_85:7a:…	0x86dd	74	IPv6
18	AzureWaveTec_d0:bb:…	1.278441	FiberhomeTel_85:7a:…	0x86dd	74	IPv6
19	AzureWaveTec_d0:bb:…	1.278452	FiberhomeTel_85:7a:…	0x86dd	74	IPv6
20	AzureWaveTec_d0:bb:…	1.278579	FiberhomeTel_85:7a:…	0x86dd	76	IPv6
21	AzureWaveTec_d0:bb:…	1.278623	FiberhomeTel_85:7a:…	0x86dd	109	IPv6
22	AzureWaveTec_d0:bb:…	1.278703	FiberhomeTel_85:7a:…	0x86dd	76	IPv6
23	AzureWaveTec_d0:bb:…	1.278737	FiberhomeTel_85:7a:…	0x86dd	109	IPv6
24	AzureWaveTec_d0:bb:…	1.278792	FiberhomeTel_85:7a:…	0x86dd	76	IPv6
25	AzureWaveTec_d0:bb:…	1.278818	FiberhomeTel_85:7a:…	0x86dd	109	IPv6
26	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	86	IPv6
27	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	74	IPv6
28	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	257	IPv6
29	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	74	IPv6
30	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	74	IPv6
31	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	74	IPv6
32	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	74	IPv6
33	f8:e5:a4:85:7a:81	1.286501	AzureWaveTec_d0:bb:…	0x86dd	448	IPv6
34	FiberhomeTel_85:7a:…	1.286654	AzureWaveTec_d0:bb:…	0x86dd	337	IPv6
35	AzureWaveTec_d0:bb:…	1.286916	FiberhomeTel_85:7a:…	0x86dd	74	IPv6



```
C:\Users\Nur Ayu Amira>cd Packet Decode

C:\Users\Nur Ayu Amira\Packet Decode>netdump

* ********** PACKET HEADER ********** ***********
Preamble
Packet Header Length : 24
 Magic Number : 2712847316
Version Major : 2
Version Minor : 4
GMT to Local Correction : 0
Jacked Packet with Length of : 262144
Accuracy to Timestamp   :  0
Data Link Type (Ethernet Type II = 1)  : 1
********* TIMESTAMP & ETHERNET FRAME ****
 Packet Number: 1
 The Packets are Captured in : 1705203177 Seconds
The Packets are Captured in : 708310 Micro-seconds
The Actual Packet Length: 55 Bytes
Packet Length (Off Wire): 55 Bytes
Ethernet Header Length  : 14 bytes
MAC Destination Address    : [hex] f8 :e4 :a4 :85 :7a :81
                         [dec] 248 :228 :164 :133 :122 :129
MAC Source Address    : [hex] 90 :e8 :68 :d0 :bb :61
                         [dec] 144 :232 :104 :208 :187 :97
IP Source Address      : 192.168.1.103
IP Destination Address : 199.232.44.176
 d7 8c 1 bb 33 bb 6b 9e 8 d7 51 51 50 10 2 0 24 62 0 0 0
```



```
* ********** PACKET HEADER ********** ***********
Preamble
Packet Header Length : 24
 Magic Number : 2712847316
Version Major : 2
Version Minor : 4
GMT to Local Correction : 0
Jacked Packet with Length of : 262144
Accuracy to Timestamp   :  0
Data Link Type (Ethernet Type II = 1)  : 1
********* TIMESTAMP & ETHERNET FRAME ****
 Packet Number: 1
 The Packets are Captured in : 1705203177 Seconds
The Packets are Captured in : 708310 Micro-seconds
The Actual Packet Length: 55 Bytes
Packet Length (Off Wire): 55 Bytes
Ethernet Header Length  : 14 bytes
MAC Destination Address    : [hex] f8 :e4 :a4 :85 :7a :81
                         [dec] 248 :228 :164 :133 :122 :129
MAC Source Address    : [hex] 90 :e8 :68 :d0 :bb :61
                         [dec] 144 :232 :104 :208 :187 :97
IP Source Address      : 171.0.0.224
IP Destination Address : 192.168.1.103
 20 26 20 20 20 eb ff ff ff 9 0 0 0 2 0 0 0 ed ff ff ff
```

- We can see the packet is **corrupted.** Because we can see from the **“abc.pcap”** the source and destination ip is **different** with **“xyz.pcap”**

Full Source Code

red is modification by student

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

FILE *input;

#define NULL 0
#define TCPDUMP_MAGIC 0xa1b2c3d4      /* Tcpdump Magic Number (Preamble) */
#define PCAP_VERSION_MAJOR 2           /* Tcpdump Version Major (Preamble) */
#define PCAP_VERSION_MINOR 4           /* Tcpdump Version Minor (Preamble) */

#define DLT_NULL 0                    /* Data Link Type Null   */
#define DLT_EN10MB 1                   /* Data Link Type for Ethernet II 100 MB and
above */
#define DLT_EN3MB 2                    /* Data Link Type for 3 Mb Experimental Ethernet
*/

// Ethernet Header
#define ETHER_ADDR_LEN 6

typedef struct packet_header
{
    unsigned int magic;                /* Tcpdump Magic Number  */
    unsigned short version_major;      /* Tcpdump Version Major */
    unsigned short version_minor;      /* Tcpdump Version Minor */
    unsigned int thiszone;             /* GMT to Local Correction */
    unsigned int sigfigs;              /* Accuracy of timestamps */
    unsigned int snaplen;              /* Max Length of Portion of Saved Packet */
    unsigned int linktype;             /* Data Link Type */
} hdr;

typedef struct packet_timestamp
{
    unsigned int tv_sec;                /* Timestamp in Seconds */
    unsigned int tv_usec;               /* Timestamp in Micro Seconds */
    /* Total Length of Packet Portion (Ethernet Length until the End of Each Packet) */
    unsigned int caplen;
    unsigned int len;                   /* Length of the Packet (Off Wire) */
} tt;

typedef struct ether_header
{
    unsigned char edst[ETHER_ADDR_LEN]; /* Ethernet Destination Address */
    unsigned char esrc[ETHER_ADDR_LEN]; /* Ethernet Source Address */
    unsigned short etype;               /* Ethernet Protocol Type */
} eth;

//MOFIFY BY AYU: IP Header
typedef struct ip_header
{
    unsigned char version_ihl;       // Version (4 bits) + Internet header length (4
bits)
    unsigned char dscp_ecn;          // DSCP (6 bits) + ECN (2 bits)
    unsigned short total_length;     // Total length
    unsigned short identification;   // Identification
    unsigned short flags_fragoffset; // Flags (3 bits) + Fragment offset (13 bits)
    unsigned char ttl;               // Time to Live
    unsigned char protocol;          // Protocol
    unsigned short checksum;         // Header checksum
    unsigned int src_ip;             // Source IP address
    unsigned int dest_ip;            // Destination IP address
} ip_hdr;

//MODIFY BY AYU: Define corruption parameters structure
struct CorruptionParams {
    bool corrupt_ttl;
    bool corrupt_protocol;
    bool swap_src_dst;
    bool set_multicast_src;
    bool length_mismatch;
} corruptionParams;


//MODIFY BY AYU: Helper function to write an integer in network byte order
void writeNetworkByteOrder(FILE *file, unsigned int value) {
    unsigned char bytes[4];
    bytes[0] = (value >> 24) & 0xFF;
    bytes[1] = (value >> 16) & 0xFF;
    bytes[2] = (value >> 8) & 0xFF;
    bytes[3] = value & 0xFF;
    fwrite(bytes, sizeof(bytes), 1, file);
}


int main(int argc, char *argv[])
{
    unsigned int remain_len = 0;
    unsigned char temp = 0;
    int i, count = 0;

    struct packet_header hdr;           /* Initialize Packet Header Structure */
    struct packet_timestamp tt;         /* Initialize Timestamp Structure */
    struct ether_header eth;           /* Initialize Ethernet Structure */
    struct ip_header ip;               /* Initialize IP Header Structure */
    unsigned char buff, array[1500];

    //MODIFY BY AYU: Ask user for corruption parameters
    cout << "Do you want to corrupt TTL? (1 for yes, 0 for no): ";
    cin >> corruptionParams.corrupt_ttl;

    cout << "Do you want to corrupt the protocol? (1 for yes, 0 for no): ";
    cin >> corruptionParams.corrupt_protocol;

    cout << "Do you want to corrupt the source and destination address? (1 for yes, 0
for no): ";
    cin >> corruptionParams.swap_src_dst;

    cout << "Do you want to corrupt the IP multicast address? (1 for yes, 0 for no): ";
    cin >> corruptionParams.set_multicast_src;

    cout << "Do you want to corrupt the IP data length mismatch with UDP data length? (1
for yes, 0 for no): ";
    cin >> corruptionParams.length_mismatch;
    FILE *output;
    input = fopen("abc.pcap", "rb"); /* Open Input File */
    output = fopen("xyz.pcap", "wb"); // Open output file

    if (input == NULL)
    {
        cout << "Cannot open saved windump file" << endl;
        return 1;
    }

    fread((char *)&hdr, sizeof(hdr), 1, input); /* Read & Display Packet Header
Information */

    cout << "\n* ********** PACKET HEADER ********** ***********" << endl;
    cout << "Preamble " << endl;
    cout << "Packet Header Length : " << sizeof(hdr) << endl;
    cout << " Magic Number : " << hdr.magic << endl;
    cout << "Version Major : " << hdr.version_major << endl;
    cout << "Version Minor : " << hdr.version_minor << endl;
    cout << "GMT to Local Correction : " << hdr.thiszone << endl;
    cout << "Jacked Packet with Length of : " << hdr.snaplen << endl;
    cout << "Accuracy to Timestamp   :  " << hdr.sigfigs << endl;
    cout << "Data Link Type (Ethernet Type II = 1)   : " << hdr.linktype << endl;


    // Write the pcap header to the output file before starting the packet processing
    fwrite((char *)&hdr, sizeof(hdr), 1, output);

    /* Use While Loop to Set the Packet Boundary */
    while (fread((char *)&tt, sizeof(tt), 1, input) && fread((char *)&eth, sizeof(eth),
1, input)) /* Read & Display Timestamp and Ethernet Header Information */
    {
        ++count;

        cout << "********* TIMESTAMP & ETHERNET FRAME ****" << endl;
        cout << " Packet Number: " << count << endl; /* Display Packet Number */
        cout << " The Packets are Captured in : " << tt.tv_sec << " Seconds" << endl;
        cout << "The Packets are Captured in : " << tt.tv_usec << " Micro-seconds" <<
endl;

        /* Use caplen to Find the Remaining Data Segment */
        cout << "The Actual Packet Length: " << tt.caplen << " Bytes" << endl;
        cout << "Packet Length (Off Wire): " << tt.len << " Bytes" << endl;

        // Read and display Ethernet header information
        cout << "Ethernet Header Length  : " << sizeof(eth) << " bytes" << endl;
        printf("MAC Destination Address    : [hex] %x :%x :%x :%x :%x :%x \n\t\t\t
[dec] %d :%d :%d :%d :%d :%d\n",
               eth.edst[0], eth.edst[1],
               eth.edst[2], eth.edst[3], eth.edst[4], eth.edst[5], eth.edst[0],
eth.edst[1],
               eth.edst[2], eth.edst[3], eth.edst[4], eth.edst[5], eth.edst[6]);

        printf("MAC Source Address    : [hex] %x :%x :%x :%x :%x :%x \n\t\t\t  [dec] %d
:%d :%d :%d :%d :%d\n",
               eth.esrc[0], eth.esrc[1], eth.esrc[2],
               eth.esrc[3], eth.esrc[4], eth.esrc[5], eth.esrc[0], eth.esrc[1],
               eth.esrc[2], eth.esrc[3], eth.esrc[4], eth.esrc[5]);

        // *********************** FOR ASSIGNMENT NOT INVOLVING WRITING BACK TO A FILE
```

```
******
        // ************************BEGIN MODIFICATION
HERE.********************************************
        //  *********************** It is recommended to add Your Code here **********
        // ****Nevertheless, in some of the questions you may need to add some code
        //  ** elsewhere in the program. ********************
        //  ......  Your Code

        //MODIFY BY AYU: Read and display IP header information
        fread((char *)&ip, sizeof(ip), 1, input);
        cout << "IP Source Address      : " << ((ip.src_ip >> 0) & 0xFF) << "." <<
((ip.src_ip >> 8) & 0xFF)
             << "." << ((ip.src_ip >> 16) & 0xFF) << "." << ((ip.src_ip >> 24) & 0xFF)
<< endl;

        cout << "IP Destination Address : " << ((ip.dest_ip >> 0) & 0xFF) << "." <<
((ip.dest_ip >> 8) & 0xFF)
             << "." << ((ip.dest_ip >> 16) & 0xFF) << "." << ((ip.dest_ip >> 24) & 0xFF)
<< endl;

        //MODIFY BY AYU: Apply the corruptions based on user input
        if (corruptionParams.corrupt_ttl) {
            ip.ttl = 0;
        }

        if (corruptionParams.corrupt_protocol) {
            ip.protocol = 0xFF; // An undefined protocol
        }

        if (corruptionParams.swap_src_dst) {
            unsigned int temp_ip = ip.src_ip;
            ip.src_ip = ip.dest_ip;
            ip.dest_ip = temp_ip;
        }

        if (corruptionParams.set_multicast_src) {
            // Set the first four bytes of the source IP to a multicast address
            ip.src_ip = 0xE00000AB;
        }

        if (corruptionParams.length_mismatch) {
            // Increment the IP total length by 1 to create a mismatch
            ip.total_length += 1;
        }

        //MODIFY BY AYU: Now write the corrupted packet back to the output file
        fwrite((char *)&tt, sizeof(tt), 1, output);
        fwrite((char *)&eth, sizeof(eth), 1, output);
        writeNetworkByteOrder(output, ip.src_ip);
        writeNetworkByteOrder(output, ip.dest_ip);
        fwrite((char *)&ip + 8, sizeof(ip) - 8, 1, output); // Write the rest of the IP
header
        fwrite((char *)array, tt.caplen - sizeof(eth) - sizeof(ip), 1, output);

        //  ****** END OF MODIFICATION HERE **********************
        //  WARNING: Try not to modify the while loop, the fread statement as you may
affect
        //  the packet boundary and the whole program may not work after that.

        for (i = 0; i < tt.caplen - sizeof(eth) - sizeof(ip); i++)
        {
            fread((char *)&buff, sizeof(buff), 1, input);
            printf(" %x", buff); // you may remove the printf line if necessary
            array[i] = buff;
        }

        printf("\n ");
    } // end while

    fclose(input); // Close input file
    fclose(output);

    return (0);
}
```