### CSN-503 | ACN Project

### **PACKET HEADER ANALYSIS**

**Group 3**

**Members:**

* Dayanand Raut (20535010)
* Swapnil Naik (20535029)
* Sagar Gaur (20535023)
* Atul Sharma (20535007)
* Mithlesh Kumar Yadav (20535016)
* Sajal Chourasiya (20535024)
* Gaurish Mishra (20535013)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SN | Name | Enrollment No | Brief Contribution | Page |
| 1 | Dayanand Raut | 20535010 | Analysis of ipv4, design of data structure to store fields of packet, extracting mac addresses of source and destination, code cleanup | 5,6,10, 24, 25 |
| 2 | Swapnil Naik | 20535029 | Carray file parsing, protocol parsing and analysis of protocols used in each layer. | 11,13,14,15,20,22,23 |
| 3 | Sagar Gaur | 20535023 | Extracting source and destination ports and IP addresses, Analysis of ipv6 and ARP packet. Contribution in report. | 4,6,7,17,18,19 |
| 4 | Atul Sharma | 20535007 | Extracting the Ethertype field , discarding the packets other than ipv4, ipv6 and ARP packets , debugging of code | 10,16,21 |
| 5 | Mithlesh Kumar Yadav | 20535016 | Extracting port number ,checking of packet whether it is ARP,IPV4,IPV6 or not ,code optimization and debugging ,contribution in ppt and report | 16,19,23 |
| 6 | Sajal Chourasiya | 20535024 | Extracting the application layer protocol using port number, worked on application layer part of print summary. Worked on presentation. | 13,18,23 |
| 7 | Gaurish Mishra | 20535013 | Printing the summary , Testing the code , Refining report | 11 |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SN** | **Topic** | **Page** |
| 1 | Problem Statement | 4 |
| 2 | Description | 4 |
| 3 | Analysis and Discussion | 5 |
| 4 | Conclusion | 8 |
|  | *References* | 9 |
|  | *Source code* | 10 |
|  | *Output* | 24 |

**Problem Statement**

Given a file containing some packets captured (in C array format). Analyze the packets and determine all the fields in the packet header like source/destination MAC, IP addresses, port numbers, and different protocols used in different layers, etc. The application should take the c array format as input and display all the parameters as an output. It should also display which are the most used protocols in different layers.

**Description**

**Using Wireshark for generating the packets and exporting captured packets in C array format.**

**Wireshark:** Wireshark is a network packet analyzer which presents captured packet data in as much detail as possible. It is an open-source software for analyzing the packets.

It can capture traffic from many different network media types, including Ethernet, Wireless LAN, Bluetooth, USB, and more. It can save captured packets in many formats including pcapng, libpcap and many more.

Wireshark provides a variety of options for exporting packet data. The formats available for exporting are plain text, C arrays, CSV, PSML, PDML, JSON.

**Analysis and Discussion**

**Analysis of C array packet**

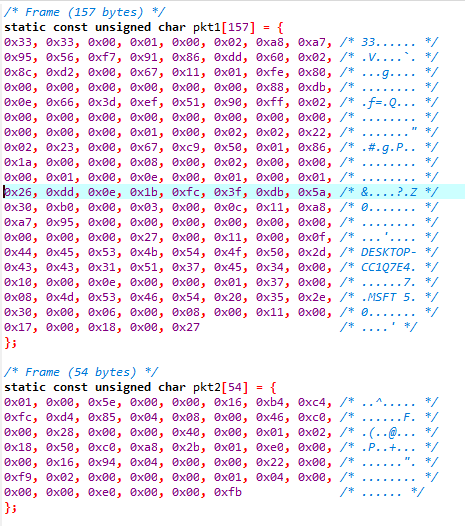


Fig. 1

C array file contains the packet in different arrays. Like as shown in figure 1 packets are captures and stored in character array.

**Analysis of the fields of ipv4 based on the hexacode from captured frame.**

|  |  |
| --- | --- |
| Field Name | Byte position |
| **Ethernet** |  |
| Destination mac | 0-5 |
| Source mac | 6-11 |
| Type | 12-13 |
|  |  |
| **IP** |  |
| Version | 14th first nibble |
| Header length | (14th second nibble) \* 4 |
| Source address | 26-29 |
| Destination address | 30-33 |
| Flags | 20 |
| Fragment Offset | 20-21 |
| TTL | 22 |
| Protocol | 23 |
| Total Length | 16-17 |
|  |  |
| **UDP / TCP** |  |
| Source port | 34-35 |
| Destination port | 36-37 |
| Length | 38-39 |

**In case of IPv6 in IP byte positions described below:**

Version: 14th byte first nibble

Priority: 14th byte second nibble and 15th byte first nibble

Flow label: 15th byte second nibble and 16-17

Payload length: 18-19

Next header: 20

Hop limit: 21

Source address: 22-37

Destination address: 38-53

TCP/UDP

Source port: 54-55

Destination port: 56-57

Length: 58-59

**ARP Packet**

ARP packet is of 42 bytes total. ARP protocol is identified by **type field** (12-13th byte position) of packet. Type field value for ARP protocol is 0806.

Analyzing the fields based on the hexacode from captured packet with byte position

**Field: byte position**

Destination MAC: 0-5

Source MAC: 6-11

Type: 12-13

Hardware type: 14-15

Protocol type: 16-17

Hardware size: 18

Protocol size: 19

Opcode: 20-21

Sender IP address: 28-31

Destination IP address: 38-41

ARP packet does not contain source and destination ports. So, in the code source port and destination port fields are stored –1

**Conclusion**

As a part of this project, packets were captured using Wireshark and exported to c array file. The file was parsed in c++ and various fields were analyzed from each packet. The results of analysis were displayed. Finally, the overall summary was prepared to show the different protocols used in different layers and most commonly used protocols were identified.

After analyzing various packets, we found that at various layers following protocols were most frequently used:

Application Layer : https

Transport Layer : TCP

Network Layer : IPv4

Data Link Layer : Ethernet

**References**

*https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml*   
*https://en.wikipedia.org/wiki/Transport\_layer*  
*https://en.wikipedia.org/wiki/Network\_layer*  
*https://blog.packet-foo.com/2015/01/determining-network-protocols/ https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/*

**Source Code**

***packet\_analyser.cpp***

#include<bits/stdc++.h>

using namespace std;

map<string,string> aplProtocolType; // to store application protocols

map<string,string> protocolType; // stores TYPE OF PROTOCOL - (hex, name\_of\_protocol)

/\*\*

Data structure to hold the values of different fields of packet.

\*/

class Packet{

public:

string srcIPAddress;

string destIPAddress;

string srcMacAddress;

string destMacAddress;

int srcPort;

int destPort;

string protocol;

string ethType;

string aplProtocol;

void printPacket(int counter){

cout<<counter<<"\t"<<protocol<<"\t"<<srcIPAddress<<"\t"<<destIPAddress<<"\t"<<srcMacAddress<<"\t"<<destMacAddress<<"\t"<<srcPort<<"\t"<<destPort<<"\t"<<ethType<<"\t"<<aplProtocol<<endl;

}

};

/\*\*

function to print the packets from the list of packet object.

\*/

void printPackets(vector<Packet> packets){

cout<<"Printing packets"<<endl;

cout<<"Packet"<<"\t"<<"Protocol"<<"\t"<<"srcIPAddress"<<"\t"<<"destIPAddress"<<"\t"<<"srcMacAddress"<<"\t"<<"destMacAddress"<<"\t"<<"srcPort"<<"\t"<<"destPort"<<"\t"<<"ethType"<<"\t"<<"aplProtocol"<<endl;

cout<<"-----------------------------------------------------------------------------------------------------------------------------------"<<endl;

for(int i=0; i<packets.size(); i++){

packets[i].printPacket(i+1);

}

}

/\*\*

function to generate the summary of protocols used in different layer.

\*/

void printSummary(vector<Packet> packets) {

cout<<"--------------------SUMMARY----------------------"<<endl;

map<string, int> nwproto;

map<string, int> transproto;

map<string, int> aplproto;

vector<string> nwprotolist = {"ICMP","IGMP","DDP","IPv6-Frag","IPv6-Route","IPv6-ICMP","IPv6-NoNxt","EGP","EIGRP","IPX-in-IP","OSPF","PIM"};

for(int i=0; i<packets.size(); i++){

if(find(nwprotolist.begin(), nwprotolist.end(), packets[i].protocol) != nwprotolist.end()) {

if(nwproto.count(packets[i].protocol))

nwproto[packets[i].protocol]++;

else

nwproto[packets[i].protocol] = 1;

}

else {

if(nwproto.count(packets[i].ethType))

nwproto[packets[i].ethType]++;

else

nwproto[packets[i].ethType] = 1;

}

}

map<string, int>::iterator itr;

string mostUsednwProtocol = "";

int nwmax = 0;

cout<<"Protocols used in network layer: "<<endl;

for(itr= nwproto.begin(); itr!=nwproto.end(); itr++) {

cout<<itr->first<<" "<<itr->second<<endl;

if(nwmax < itr->second) {

mostUsednwProtocol = itr->first;

nwmax = itr->second;

}

}

cout<<"Most used protocol on network layer: "<<mostUsednwProtocol<<endl;

cout<<"----------------------------------------------------"<<endl;

//-------- transport layer part

vector<string> transprotocolList = {"TCP","UDP","DCCP","FC","IL","RDP","SCTP","TCP","UDP","UDPLite"};

for(int i=0; i<packets.size(); i++){

if(find(transprotocolList.begin(), transprotocolList.end(), packets[i].protocol) != transprotocolList.end()) {

// it means it is some transport layer protocol

if(transproto.count(packets[i].protocol))

transproto[packets[i].protocol]++;

else

transproto[packets[i].protocol] = 1;

}

else {

if(transproto.count("OTHER")) {

transproto["OTHER"]++;

}

else

transproto["OTHER"] = 1;

}

}

string mostUsedtransProtocol = "";

int transmax = 0;

cout<<"Protocols used in Transport layer: "<<endl;

for(itr= transproto.begin(); itr!=transproto.end(); itr++) {

cout<<itr->first<<" "<<itr->second<<endl;

if(transmax < itr->second && itr->first.compare("OTHER") != 0) {

mostUsedtransProtocol = itr->first;

transmax = itr->second;

}

}

cout<<"Most used protocol on Transport layer: "<<mostUsedtransProtocol<<endl;

cout<<"--------------------------------------------------"<<endl;

//--------- application layer part

for(int i=0; i<packets.size(); i++){

if(aplproto.count(packets[i].aplProtocol)) {

aplproto[packets[i].aplProtocol]++;

}

else {

aplproto[packets[i].aplProtocol] = 1;

}

}

string mostUsedaplProtocol = "";

int aplmax = 0;

cout<<"Protocols used in Application layer: "<<endl;

for(itr= aplproto.begin(); itr!=aplproto.end(); itr++) {

cout<<itr->first<<" "<<itr->second<<endl;

if(aplmax < itr->second && itr->first.compare("OTHER") != 0 && itr->first.compare("NONE") != 0) {

mostUsedaplProtocol = itr->first;

aplmax = itr->second;

}

}

cout<<"Most used protocol on Application layer: "<<mostUsedaplProtocol<<endl;

cout<<"---------------------------------------------------"<<endl;

}

/\*\*

function to read a c array file and returns it in string format

\*/

string readcFile(string name) {

string line;

string result = "";

ifstream myfile (name);

if (myfile.is\_open()) {

while (getline(myfile,line))

result += line;

myfile.close();

}

else

cout<<"Unable to open file";

return result;

}

/\*\*

function to read protocols from given file with list of all protocols

format is of the form - HEX NUM NAME

\*/

void readprotocolFile() {

string line;

string result = "";

ifstream myfile ("listofprotocol.txt");

if (myfile.is\_open()) {

while (getline(myfile,line)) {

string temp;

int i=0;

string hex = "";

while(line[i] != ' ' && i<line.length()) {

hex += string(1,line[i]);

i++;

}

i++;

string num;

while(line[i] != ' ' && i<line.length()) {

num += string(1,line[i]);

i++;

}

i++;

string protocol;

while(line[i] != ' ' && i<line.length()) {

protocol += string(1,line[i]);

i++;

}

transform(hex.begin(), hex.end(), hex.begin(), ::tolower);

protocolType[hex] = protocol;

}

myfile.close();

}

else

cout<<"Unable to open file";

}

/\*\*

function to split packet based on ,

\*/

vector<string> splitData(string packet) {

int n = packet.length();

vector<string> result;

string temp;

for(int i=0; i<n; i++) {

if(packet[i]!=',')

temp += string(1,packet[i]);

else {

result.push\_back(temp);

temp = "";

}

}

if(temp!="" || temp!= " ")

result.push\_back(temp);

return result;

}

/\*\*

function to get mac addresses

\*/

void getSourceDestMac(vector<string> fields, Packet \*pkt){

int MACsourceAddressStartIndex = 6, MACdestAddressStartIndex = 0;

string sourceMAC = fields[MACsourceAddressStartIndex]+fields[MACsourceAddressStartIndex+1]+fields[MACsourceAddressStartIndex+2]+fields[MACsourceAddressStartIndex+3]+fields[MACsourceAddressStartIndex+4]+fields[MACsourceAddressStartIndex+5];

string destMAC = fields[MACdestAddressStartIndex]+fields[MACdestAddressStartIndex+1]+fields[MACdestAddressStartIndex+2]+fields[MACdestAddressStartIndex+3]+fields[MACdestAddressStartIndex+4]+fields[MACdestAddressStartIndex+5];

string sourcefinal = sourceMAC.substr(2, 2)+":"+sourceMAC.substr(6, 2)+":"+sourceMAC.substr(10, 2)+":"+sourceMAC.substr(14, 2)+":"+sourceMAC.substr(18, 2)+":"+sourceMAC.substr(22, 2);

string destfinal = destMAC.substr(2, 2)+":"+destMAC.substr(6, 2)+":"+destMAC.substr(10, 2)+":"+destMAC.substr(14, 2)+":"+destMAC.substr(18, 2)+":"+destMAC.substr(22, 2);;

pkt->srcMacAddress = sourcefinal;

pkt->destMacAddress = destfinal;

}

/\*\*

function to check whether the packet is ARP

\*/

int checkARP(vector<string> fields, Packet \*pkt){

string str1 = fields[12]+fields[13]; // checking ARP on basis of 13th and 14th byte.

string check = str1.substr(2,2) + str1.substr(6, 2);

string compare1 = "0806";

string compare2 = "0800";

string compare3 = "86dd";

if((check.compare(compare1)) == 0)

return (1);

else if((check.compare(compare2)) == 0)

return (2);

else if((check.compare(compare3)) == 0)

return (3);

else

return (4);

}

/\*\*

function to get various fields from ARP packet

\*/

void ARPdetails(vector<string> fields, Packet \*pkt){

{

int sourceAddressStartIndex = 28, destAddressStartIndex = 38;

string src = to\_string(stoi(fields[sourceAddressStartIndex], 0, 16))+"."+to\_string(stoi(fields[sourceAddressStartIndex+1], 0, 16))+"."+to\_string(stoi(fields[sourceAddressStartIndex+2], 0, 16))+"."+to\_string(stoi(fields[sourceAddressStartIndex+3], 0, 16));

string des = to\_string(stoi(fields[destAddressStartIndex], 0, 16))+"."+to\_string(stoi(fields[destAddressStartIndex+1], 0, 16))+"."+to\_string(stoi(fields[destAddressStartIndex+2], 0, 16))+"."+to\_string(stoi(fields[destAddressStartIndex+3], 0, 16));

pkt->srcIPAddress = src;

pkt->destIPAddress = des;

pkt->srcPort = -1;

pkt->destPort = -1;

pkt->protocol = "ARP";

}

}

/\*\*

function to get SOURCE and DESTINATION ip addresses based on HEX value

\*/

void getSourceDest(vector<string> fields, Packet \*pkt) {

int n = 14,version;

string str = fields[n];

string str2= str.substr(2,1); // for extracting IP version

version = stoi(str2, 0, 10);

if(version == 4){

int sourceAddressStartIndex = 26, destAddressStartIndex = 30;

string src = to\_string(stoi(fields[sourceAddressStartIndex], 0, 16))+"."+to\_string(stoi(fields[sourceAddressStartIndex+1], 0, 16))+"."+to\_string(stoi(fields[sourceAddressStartIndex+2], 0, 16))+"."+to\_string(stoi(fields[sourceAddressStartIndex+3], 0, 16));

string des = to\_string(stoi(fields[destAddressStartIndex], 0, 16))+"."+to\_string(stoi(fields[destAddressStartIndex+1], 0, 16))+"."+to\_string(stoi(fields[destAddressStartIndex+2], 0, 16))+"."+to\_string(stoi(fields[destAddressStartIndex+3], 0, 16));

pkt->srcIPAddress = src;

pkt->destIPAddress = des;

}

else{

// for IPv6 address.

int sourceAddressStartIndex = 22,i, destAddressStartIndex = 38;

string strsource1,strsource2,strsource11,strdest1,strdest2,strdest11,sourcefinal="",destfinal="",comparestring="ff";

strsource1 = fields[sourceAddressStartIndex];

strsource2 = fields[sourceAddressStartIndex+1];

strsource11 = strsource1.substr(2, 2)+strsource2.substr(2, 2);

sourcefinal = sourcefinal+strsource11;

strdest1 = fields[destAddressStartIndex];

strdest2 = fields[destAddressStartIndex+1];

strdest11 = strdest1.substr(2, 2)+strdest2.substr(2, 2);

destfinal = destfinal+strdest11;

for(i=2;i<=14;i=i+2)

{

strsource1 = fields[sourceAddressStartIndex+i];

strsource2 = fields[sourceAddressStartIndex+(i+1)];

strsource11 = strsource1.substr(2, 2)+strsource2.substr(2, 2);

sourcefinal=sourcefinal+":"+strsource11;

strdest1 = fields[destAddressStartIndex+i];

strdest2 = fields[destAddressStartIndex+(i+1)];

strdest11 = strdest1.substr(2, 2)+strdest2.substr(2, 2);

destfinal=destfinal+":"+strdest11;

strsource1="", strsource11 ="";

strdest1="" ,strdest11="";

}

pkt->srcIPAddress = sourcefinal;

pkt->destIPAddress = destfinal;

}

}

/\*\*

function to identify application layer protocol based on port number

\*/

void getApplicationLayerProtocol(int sourceportnum, int destportnum, Packet \*pkt) {

string sport = to\_string(sourceportnum);

string dport = to\_string(destportnum);

if(aplProtocolType.count(sport) || aplProtocolType.count(dport)) {

if(aplProtocolType.count(sport)) {

pkt->aplProtocol = aplProtocolType[sport];

}

else {

pkt->aplProtocol = aplProtocolType[dport];

}

}

else{

pkt->aplProtocol = "OTHER";

}

}

/\*\*

function to get source and destination ports

\*/

void getPorts(vector<string> fields, Packet \*pkt){

int n = 14,m; // used for storing location of header version and header length

int version, headerlength,sourceportnumber,destportnumber;

string sourceport="",destport="";

string str = fields[n];

string str2= str.substr(2,1); // for extracting IP version

string str3= str.substr(3,1);

version = stoi(str2, 0, 10);

headerlength = stoi(str3, 0, 10);

if(version == 4)

{

headerlength\*=4;

m= n+headerlength; // for storing source port index.

sourceport+= fields[m]+ fields[m+1];

sourceport = sourceport.substr(2,2) + sourceport.substr(6,2);

sourceportnumber = stoi(sourceport, 0, 16);

destport+=fields[m+2] + fields[m+3];

destport = destport.substr(2,2) + destport.substr(6,2);

destportnumber = stoi(destport, 0, 16);

pkt->srcPort = sourceportnumber;

pkt->destPort = destportnumber;

getApplicationLayerProtocol(sourceportnumber,destportnumber,pkt);

sourceport=""; destport="";

}

else

{

m = n+ 40;

sourceport+= fields[m]+ fields[m+1];

sourceport = sourceport.substr(2,2) + sourceport.substr(6,2);

sourceportnumber = stoi(sourceport, 0, 16);

destport+=fields[m+2] + fields[m+3];

destport = destport.substr(2,2) + destport.substr(6,2);

destportnumber = stoi(destport, 0, 16);

pkt->srcPort = sourceportnumber;

pkt->destPort = destportnumber;

getApplicationLayerProtocol(sourceportnumber,destportnumber,pkt);

sourceport=""; destport="";

}

}

/\*\*

function to get NAME OF THE PROTOCOL based on HEX value

\*/

void getProtocol(vector<string> fields, Packet\* pkt) {

int location = 23;

string protocol = "";

if(protocolType.count(fields[location])){

protocol = protocolType[fields[location]];

}

else{

protocol = "Unassigned/Experimental";

}

pkt->protocol = protocol;

}

/\* get SIZE of the packet in bytes \*/

void getPacketSize(vector<string> fields) {

cout<<"Packet Size: "<<fields.size()<<" bytes"<<endl;

}

/\*\*

Main function to analyse fields values

\*/

void analysefields(vector<string> fields, int packetCount, Packet \*pkt, int \*print) {

int check; // for checking ARP packet.

/\* Implement functions here \*/

check = checkARP(fields, pkt);

\*print=1;

if(check == 1)

{

pkt->ethType = "ARP";

pkt->aplProtocol = "NONE";

getSourceDestMac(fields, pkt);

ARPdetails(fields, pkt);

}

else if(check == 2){

pkt->ethType = "IPV4";

getSourceDestMac(fields, pkt);

getSourceDest(fields, pkt);

getProtocol(fields, pkt);

getPorts(fields, pkt);

}

else if(check == 3){

pkt->ethType = "IPV6";

getSourceDestMac(fields, pkt);

getSourceDest(fields, pkt);

// ipv6 has different offset

int location = 20;

string protocol = "";

if(protocolType.count(fields[location])){

protocol = protocolType[fields[location]];

}

else{

protocol = "Unassigned/Experimental";

}

pkt->protocol = protocol;

getPorts(fields, pkt);

}

else

{

pkt->ethType = "OTHER";

getPacketSize(fields);

\*print=0;

}

}

/\*\*

function to divide c array data into packets

\*/

void getPackets(string data) {

vector<Packet> packets;

int packetCount = 0;

string packet = "";

bool ignorePacket = false;

for(int i=0; i<data.length(); i++) {

char curr = data[i];

if(i+11<data.length() && data.substr(i,11).compare("Reassembled")==0) {

ignorePacket = true;

}

if(curr=='{') {

if(!ignorePacket)

packetCount++;

char next = '{';

while(next != '}') {

i++;

next = data[i];

if(next != ' ') {

// remove comment from c array file

if(data[i] == '/' && data[i+1]=='\*') {

i+=2;

int valid = 1;

while(valid) {

i++;

if(data[i] == '\*' && data[i+1] == '/')

valid = 0;

}

i++;

}

else if(next != '}' && next != ' ' && data[i] != '\n' && data[i] != '\r')

packet += next;

}

}

vector<string> fields = splitData(packet); // split data based on ,

packet = "";

if(!ignorePacket) {

// analyse each packet

Packet pkt;

int print; // print is 1 when ethertype is ipv4,ipv6 or arp otherwise 0

analysefields(fields,packetCount,&pkt, &print);

if(print==1)

packets.push\_back(pkt);

else

packetCount--;

}

else

ignorePacket = false;

}

}

printPackets(packets);

printSummary(packets);

}

void readApplicationLayerProtocolList() {

string line;

ifstream myfile("service-names-port-numbers.txt");

if(myfile.is\_open()) {

while(getline(myfile, line)) {

vector<string> result = splitData(line);

aplProtocolType[result[0]] = result[1];

}

myfile.close();

}

else

cout<<"unable to open file"<<endl;

}

int main () {

cout<<"Enter the name of the input file: "<<endl;

string temp;

cin>>temp;

readprotocolFile(); // read a list of protocol hex and their name

readApplicationLayerProtocolList();

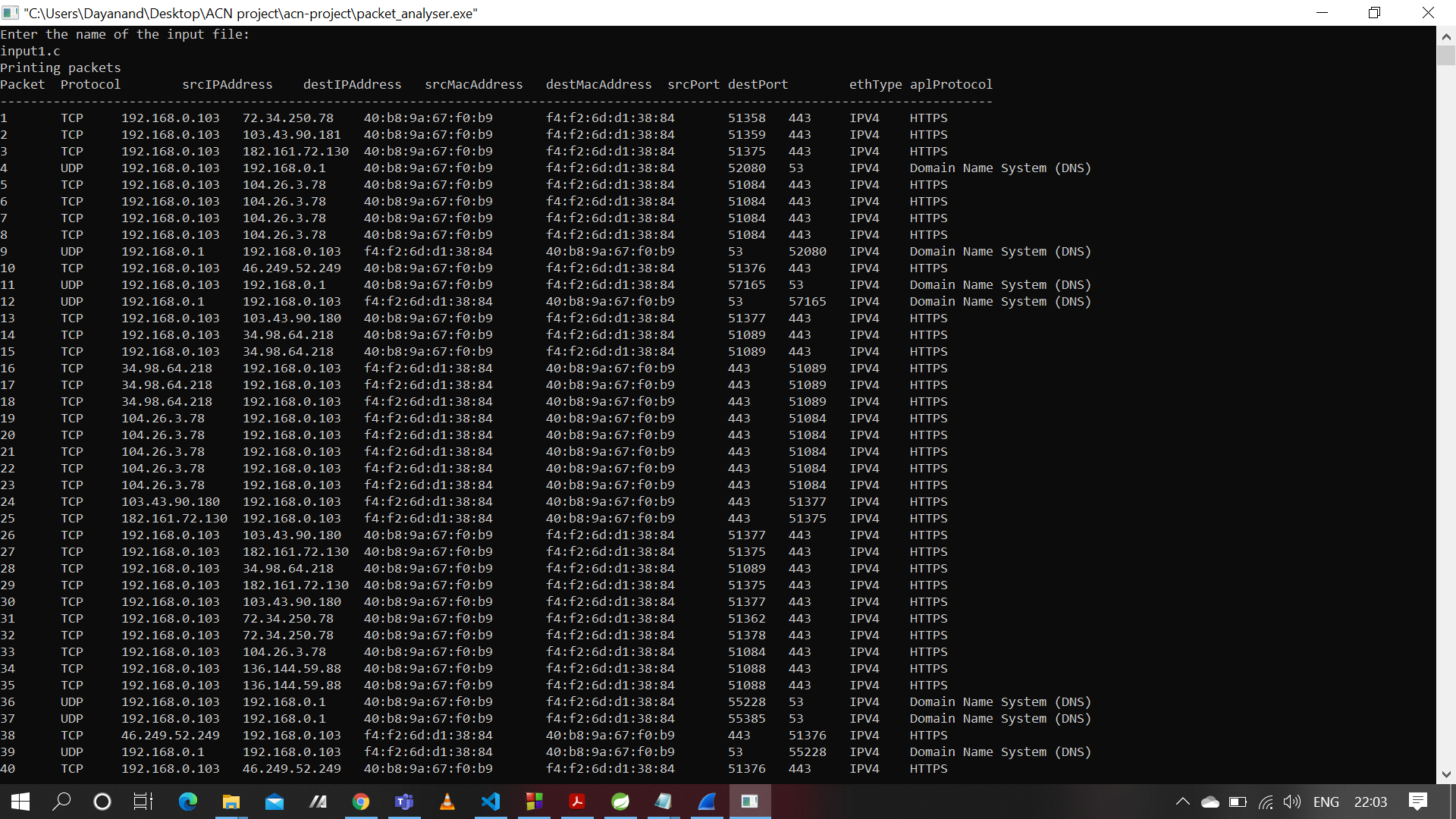
string data = readcFile(temp); // read c array file

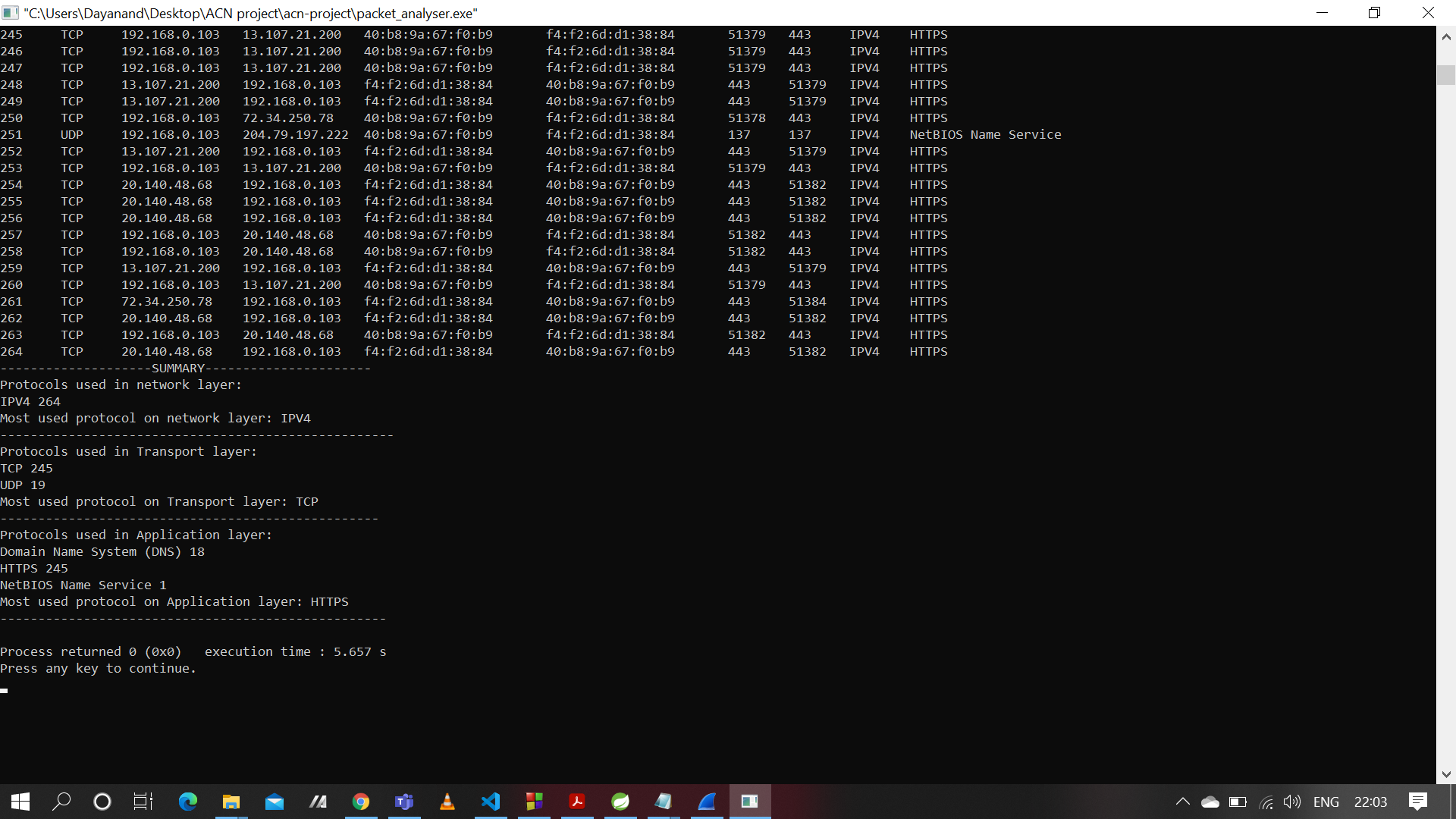
getPackets(data); // divide data into packets

return 0;

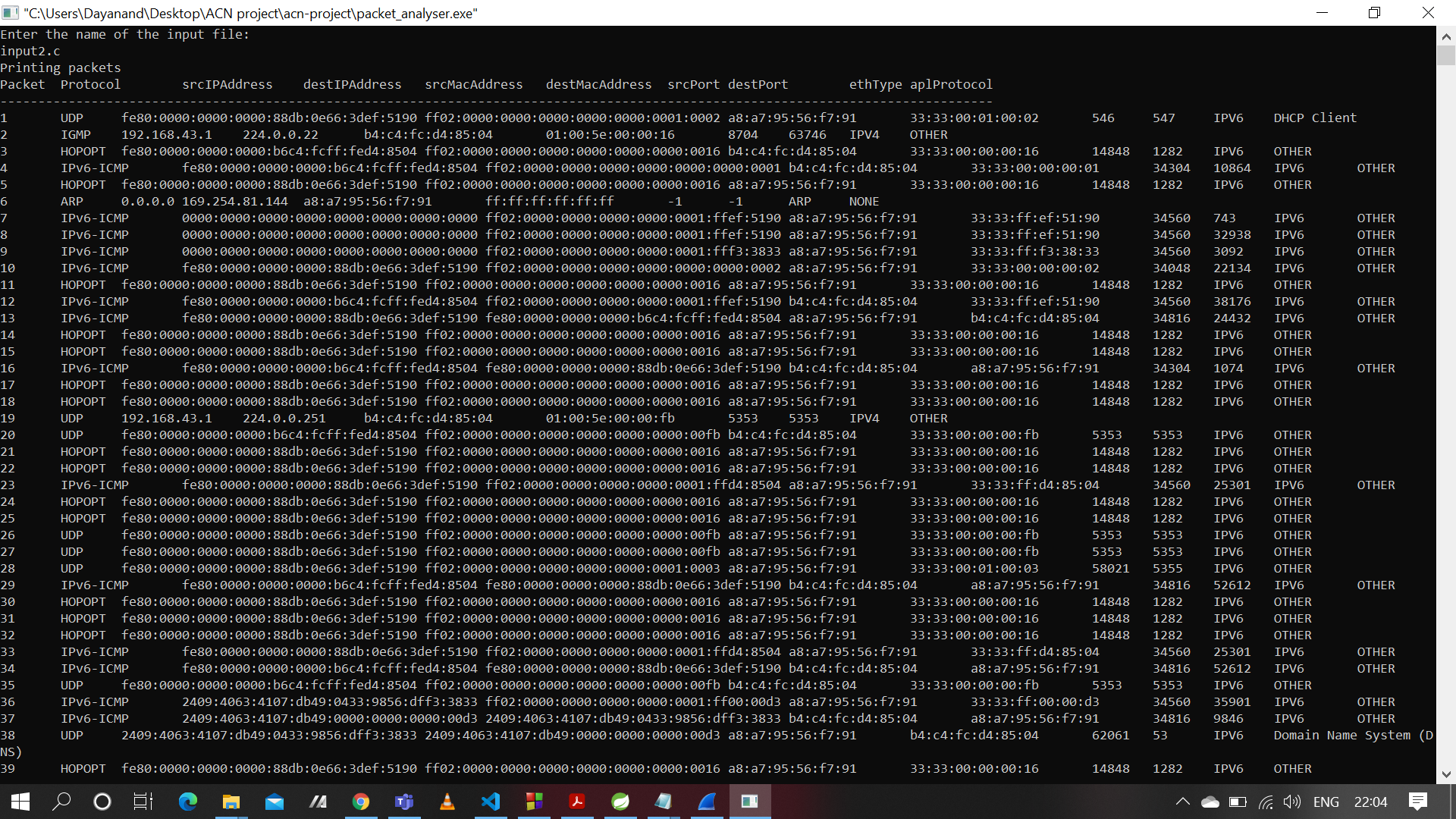
}

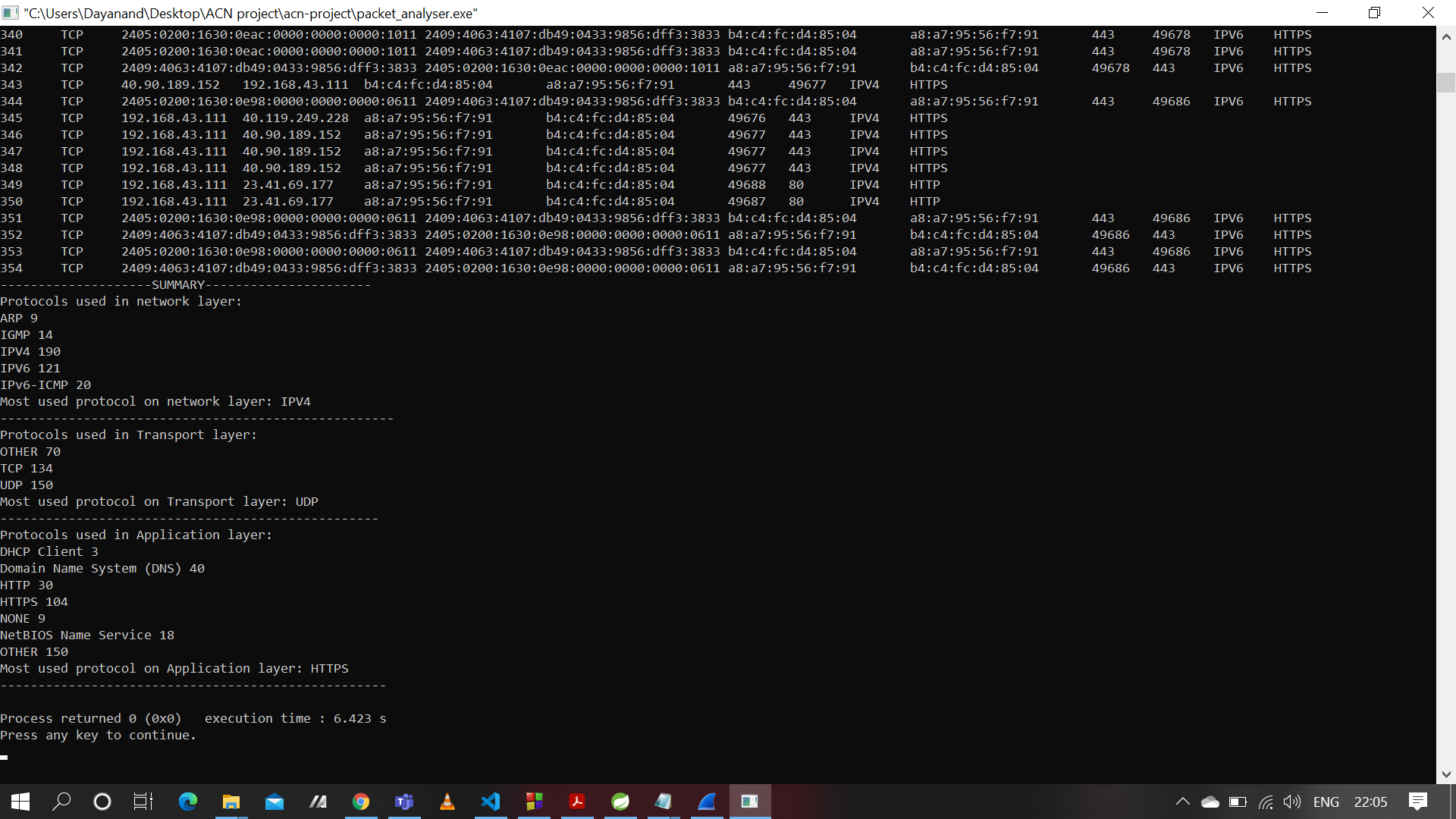
**Output Screenshots**





Output for input1.c





Output for input2.c