Scenario Independent Feature Extraction for Detecting Intrusions over TCP/IP connections

Report submitted to the SASTRA Deemed to be University

As the requirement for the course

CSE 302: COMPUTER NETWORKS

Submitted by

Sankara Narayanan S

(Reg. No.: 124156079, B. Tech, CSE spl. Artificial intelligence and data science)

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR - 613401

Bonafide Certificate

This is to certify that the report titled “Scenario Independent Feature Extraction for Detecting Intrusions over TCP/IP connections”submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech is a bonafide record of the work done by **Mr. S Sankaranarayanan**

(Reg. No.124156079, B.Tech Computer Science and Engineering with specialization in ai and Data Science) during the academic year 2020-21, in the School of Electrical and Electronics Engineering.

Project Based Work *Viva voc*e held on ___________

Examiner 1 **Examiner**

List of Figures

ABBREVIATIONS

AI-Artificial intelligence

CSV-Comma Separated Value

IDS-Intrusion Detection System

IPS-Intrusion Protection system

ML-Machine Learning

TCP-Transmission Control Protocol

SVC-Support Vector Classifier

Abstract:

Work inspired by E. Viegas, A. Santin and V. Abreu's paper:

"Enabling Anomaly-based Intrusion Detection Through Model Generalization".

The goal is to recreate an Intrusion Detection System (IDS) by training a **Machine Learning** model based on the traffic recreated within a virtual environment. The traffic generated is difficult to use to train machine learning models as it is scenario dependent, so it would lead to models trained for that specific scenario. To solve this problem, it is necessary to treat the generated traffic to be independent of the simulated session (virtual or real environment).

The traffic generated (HTTP, SMTP, SMNP, SSH) is listened to using **tcpdump**; The generated .dump file is converted into a file called totaltraffic.c containing array C using **wireshark**; Featuresextractor.py containing **python** code is launched;

In the end, 50 features independent of the scenario are obtained and can be used for model training.

KEY WORDS: Machine Learning,tcpdump,wireshark,python

Table

1.INTRODUCTION:

Intrusion detection systems are systems which automate the process of monitoring and analyzing the events that occur in computer network,to detect malicious activity. The ultimate objective of any IDS is to get culprits into demonstration before they do real damage to resources. As the seriousness of attacks occurring in the network has been increased rapidly,Intrusion detection system have become a necessary addition to security infrastructure of most organizations.

IDS are often confused with firewalls. But there is difference between them. A firewall must be regarded as a hedge that protects the information flow and prevent intrusions whereas IDS detects if the network is under attack or if the security imposed by the firewall has been penetrated. Together firewall and IDS improves and protects the security of network.

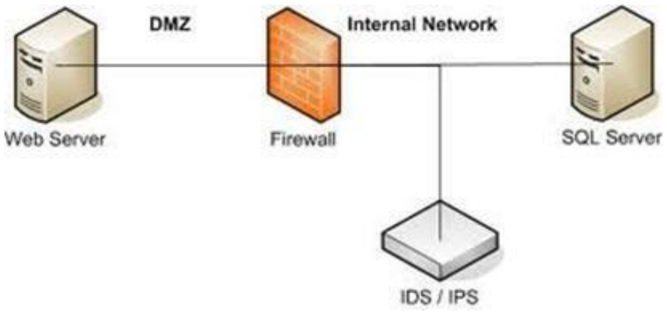


Fig. 1.1 Network Design: Firewall and IDS

TYPES OF IDS:

Intrusion detection systems can use different kind of methods to detect suspiciousactivities.It is broadly classified into

Signature based intrusion detection:

It uses database of well-known attack pattern and any incoming packet matches one of thepatterns are considered as malicious.This type of IDS cannot detect new attacks and its database should be updated continuously.

Anomaly based intrusion detection:

It creates profile that represents normal behaviour and any deviation from this behaviour isconsidered as attack packet.

IDS can detect intrusions in different places.Based on where they discover,they can be classified into

Network intrusion detection systems:

When IDS look for patterns in network traffic,then it is network based

Host intrusion detection systems:

when IDS look for attack signatures in log files,then it is host based.

LIMITATIONS OF EXISTING IDS:

- Some real attacks are far less than the number of false alarms raised. This causes some risky threats to often go unnoticed.
- Another problem with existing IDSs is they fail to detect unknown attacks. Becausenetwork environments change frequently, attack variants and new attacks emerge constantly
- Constant database updates are required for signature-based IDS to keep up with thenew threats.

Machine learning based IDS can achieve satisfactory detection levels when sufficient training data is available, and machine learning models have sufficient generalizability to detect attack variants and novel attacks. In addition, machine learning based IDS's do not rely heavily on domain knowledge; therefore, they are easy to design and construct.

2. Procedure:

2.1 Creation of virtual environment, execution of attacks and capture of packets:

Virtual environment like the one shown in the Fig 1 is created.

We can use any virtualizer, the important thing is that the client machines can only communicate with the server. The server is the only access point to the internet and takes care of providing connectivity to clients. The goal is to create an environment that is as isolated as possible.Client and server implement different types of services as shown in Fig 2.

To implement the described scenario, Debian-based distributions (ParrotOS and Kali Linux) were used. The following configuration in \ etc \ network \ interfaces can be used for each client as shown in Fig 3.

```
#Client1 configuration (dhcp or static)

auto eth0

iface eth0 inet dhcp

#Default gateway

post-up route add default gw 10.0.1.2
```

To automate the traffic acquisition process, the clients have been synchronized with the server following the scheme shown in Fig 4. RUN.py is a script that runs one of the 4 scripts shown in Fig 4) and after a certain period starts both the LOIC(Low Orbit Ion Cannon) and SYNflood attack.LOIC is required to generate the HTTPflood attack.

To use it we can use mono:

For that we have to run some commands in linux:

```
sudo apt install apt-transport-https dirmngr gnupg ca-certificates

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys

 3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF

 echo "deb https://download.mono-project.com/repo/debian stable-buster main"

 | sudo tee /etc/apt/sources.list.d/mono-official-stable.list

 sudo apt update

 sudo apt install mono-devel
```

After completing the setup, first ActivationServer.py is executed then ActivationClient.py is ran successfully as it connects the three clients to the Server. Run.py is executed where it uses LOIC and SYNFLOOD attacks on HTTP,SNMP,SMTP,SSH protocols , where it takes 13 min to start an attack and runs for 22 minutes for a total amount of 47 minutes , meanwhile we can monitor the attacks using Wireshark.

2.2 Featureextraction:

To use featuresExtractor we need to convert packages to C array. we can use Wireshark for this purpose as shown in Fig 5)

To run the featuresExtractor.py it is necessary to pass first the list of IPs of the Server interfaces and then that of the Client. Order is important.The file to be obtained must have the structure , featuresextractor will take care of extracting the information and creating the dataset.

Here is an example to understand how to use featuresextractor.

Files used must be stored in same folder:

1.connection.py

2.counterHistory.py

3.extractor.py

4.mergefile.py

5.featuresExtractor

6.empty folder tmp to store tempory files

7.C array script

We run the script: python featuresExtractor.py [10.0.1.2,10.0.2.2,10.0.3.2] [10.0.1.3,10.0.2.3,10.0.3.3]

Running it finally we obtain output.csv as in Fig 6) containing all the required features which can be used for further modelling.

2.3 Machine Learning to predict Intrusions:

Output.csv is imported using pandas, Null values are checked . Correlation and data preprocessing is done by viewing the heat map as shown in Fig 7) . Correlated values over 0.95 is removed using a simple for loop. Finally we obtain “best1.csv” which contains the required data for machine learning.

The data is used to Train in following models to predict whether the column [‘Type’] is either Attack or Normal i.e to predict the intrusions:

1. Naïve Bayes
2. Decision Tree
3. Random Tree Classifier
4. SVC
5. Logistic Regression

Fig 9,10,11,12 describes the heatmap of the confusion matrix of the above ml models

3. Source Code

Activation Client.py

```
import socket
import sys
import os
import time

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((sys.argv[1], int(sys.argv[2])))
    signal = s.recv(1024)

    if signal.decode('UTF-8') == 'OK':
        n = s.recv(1024).decode('UTF-8')
        time.sleep(2)
        os.system('python RUN.py ' + n)
```

ActivationServer.py:

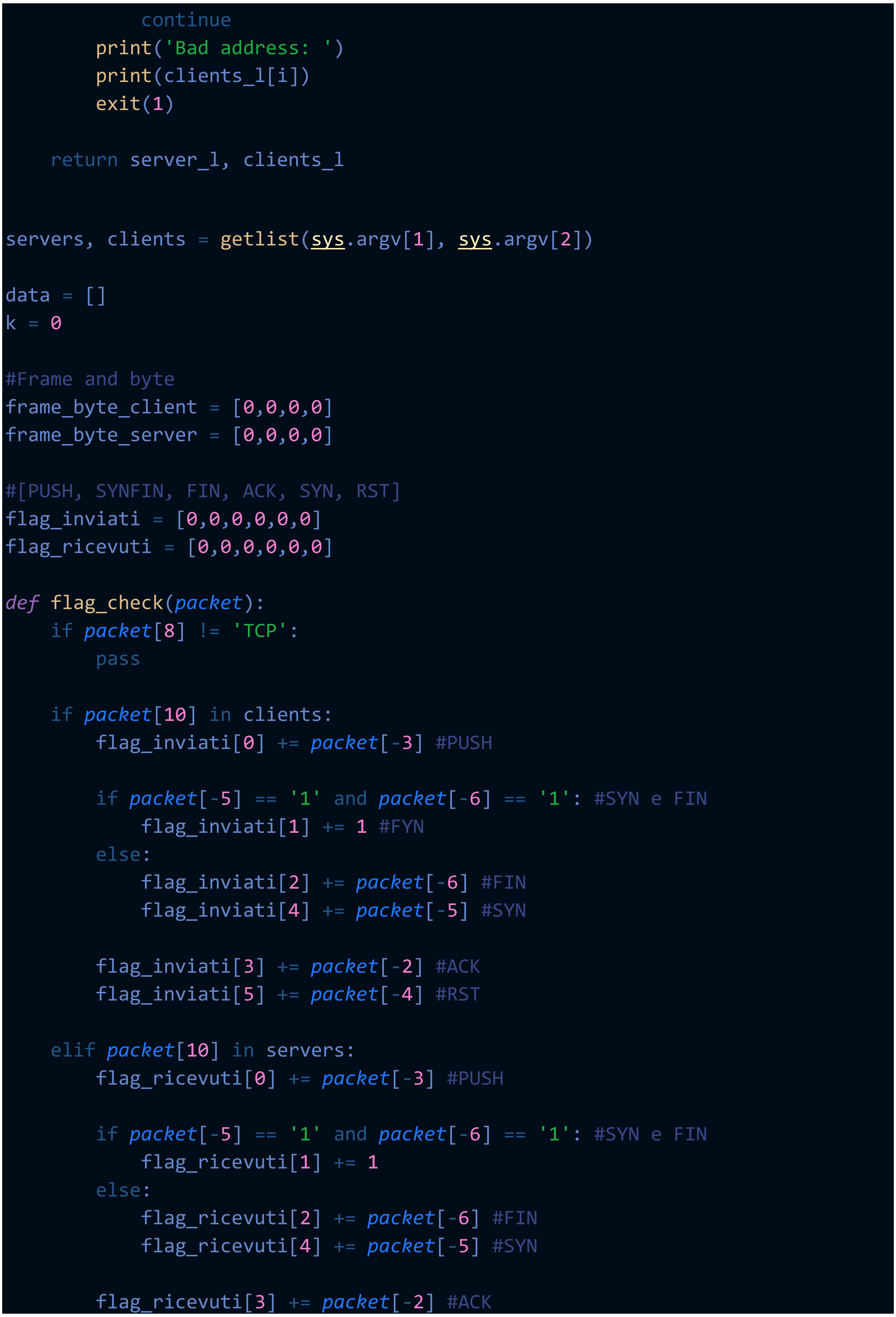
```
import os
import socket
import time
import sys


HOSTS= ['10.0.1.2', '10.0.2.2', '10.0.3.2']
PORT=[88,89,90]

pid_figlio = [0,0,0]
s = []

for i in range(0,3):
    s.append(socket.socket(socket.AF_INET, socket.SOCK_STREAM))
    s[i].bind((HOSTS[i], PORT[i]))
    s[i].listen()

#È una soluzione tappabuchi ma in linux non ho modo di condividere le
variabili
time.sleep(10)

for i in range(0, 3):
    pid_figlio[i] = os.fork()
    if pid_figlio[i] == 0:
        connection, client_addr = s[i].accept()
        connection.sendall('OK'.encode('UTF-8'))
        connection.sendall(str(i).encode('UTF-8'))
    else:
        os.system('python HTTP/httpServer.py ' + str(i))

sys.exit()
```

RUN.py:

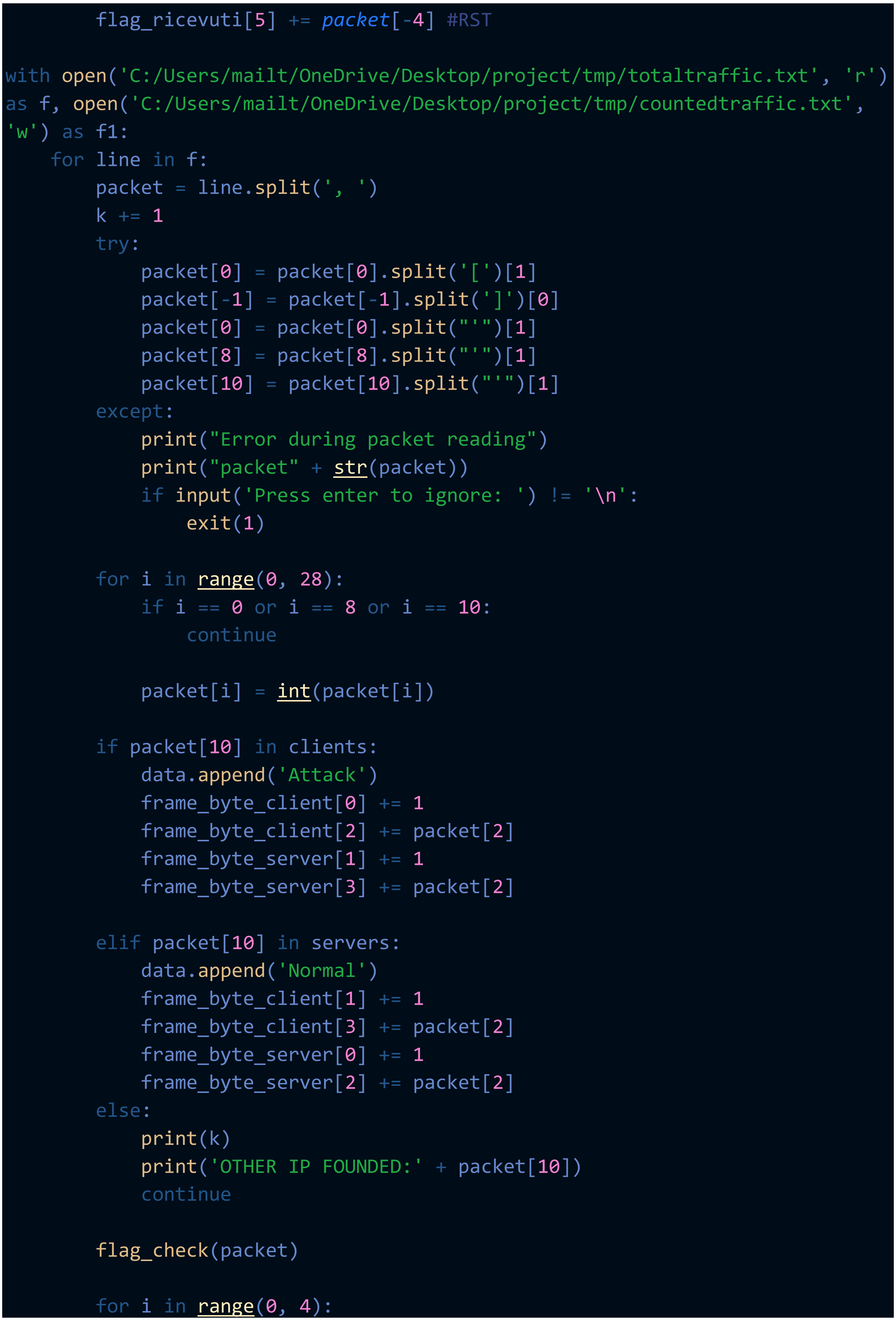
```
from random import uniform, randint
import time
from datetime import datetime, timedelta
import subprocess
import sys

path = ['HTTP/httpClient.py', 'SSH/ssh.py', 'SMTP/SMTP.py', 'SNMP/SNMP.py']
clients = ['client1', 'client2', 'client3']

ips = ['10.0.1.2', '10.0.2.2', '10.0.3.2']
ports = [85, 86, 87]

attack_done = False
attack_close = False

i = int(sys.argv[1])


def curr_time():
    return datetime.strptime(str(datetime.now()), '%Y-%m-%d %H:%M:%S.%f')


end_time = curr_time() + timedelta(minutes=40)
hack_time = curr_time() + timedelta(minutes=13)
end_hack_time = hack_time + timedelta(minutes=22)


while curr_time() <= end_time:

    service = randint(0, 3)

    # Bloccante, creare un nuovo processo indipendente
    if curr_time() >= hack_time and attack_done == False:
        attack_done = True
        subprocess.Popen(
            ["mono", "/home/sankar/Desktop/project/Script/LOIC.exe"])

        subprocess.Popen(
            ["msfconsole", "-r", "attack_config" + sys.argv[1] + ".rc", "-x", 
"run"])

    if curr_time() >= end_hack_time and attack_close == False:
        attack_close = True
        subprocess.Popen(["killall", "-e", "mono"])
        subprocess.Popen(["killall", "-e", "ruby"])

    # BUILD COMMAND
    if service == 0:
        subprocess.Popen(['python' , path[service] , ips[i] , str(ports[i])])
    if service == 1:
        subprocess.Popen(['python' , path[service] , ips[i]])
    if service == 2:
        subprocess.Popen(['python', path[service] , clients[i] ,ips[i]])
    if service == 3:
        subprocess.Popen(['python' , path[service] , ips[i]])

    time.sleep(uniform(0, 4))
```

Connection.py:

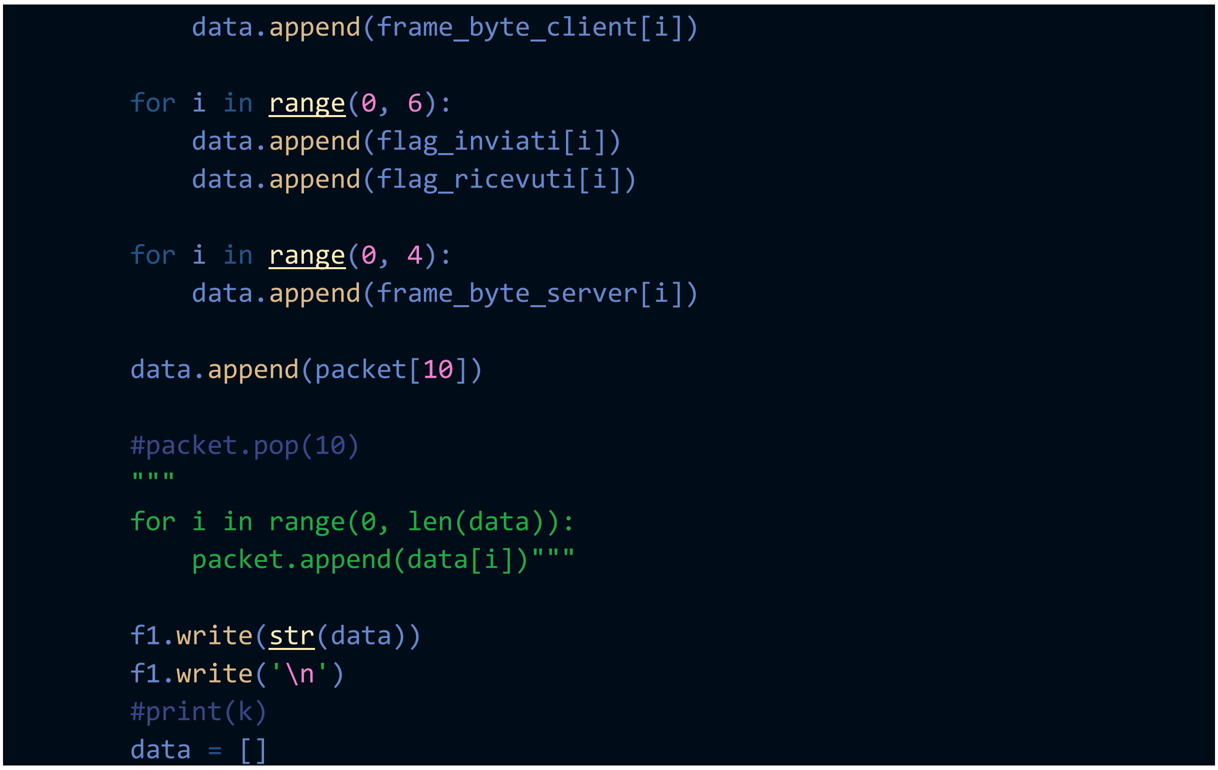
```
import sys
import re

def getlist(server_list, client_list):
    try:
        server_l = server_list.split(',')
        clients_l = client_list.split(',')
        server_l[0] = server_l[0].split('[')[1]
        clients_l[0] = clients_l[0].split('[')[1]
        server_l[-1] = server_l[-1].split(']')[0]
        clients_l[-1] = clients_l[-1].split(']')[0]
    except:
        print('Input bad formatted')
        exit(1)

    for i in range(0, len(server_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$", server_l[i]):
            continue
        print('Bad address: ')
        print(server_l[i])
        exit(1)

    for i in range(0, len(clients_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$", 
clients_l[i]):
```



```
            continue
        print('Bad address: ')
        print(clients_l[i])
        exit(1)

    return server_l, clients_l


servers, clients = getlist(sys.argv[1], sys.argv[2])

data = []
k = 0

#Frame and byte
frame_byte_client = [0,0,0,0]
frame_byte_server = [0,0,0,0]

#[PUSH, SYNFIN, FIN, ACK, SYN, RST]
flag_inviati = [0,0,0,0,0,0]
flag_ricevuti = [0,0,0,0,0,0]

def flag_check(packet):
    if packet[8] != 'TCP':
        pass

    if packet[10] in clients:
        flag_inviati[0] += packet[-3] #PUSH

        if packet[-5] == '1' and packet[-6] == '1': #SYN e FIN
            flag_inviati[1] += 1 #FYN
        else:
            flag_inviati[2] += packet[-6] #FIN
            flag_inviati[4] += packet[-5] #SYN

        flag_inviati[3] += packet[-2] #ACK
        flag_inviati[5] += packet[-4] #RST

    elif packet[10] in servers:
        flag_ricevuti[0] += packet[-3] #PUSH

        if packet[-5] == '1' and packet[-6] == '1': #SYN e FIN
            flag_ricevuti[1] += 1
        else:
            flag_ricevuti[2] += packet[-6] #FIN
            flag_ricevuti[4] += packet[-5] #SYN

        flag_ricevuti[3] += packet[-2] #ACK
```

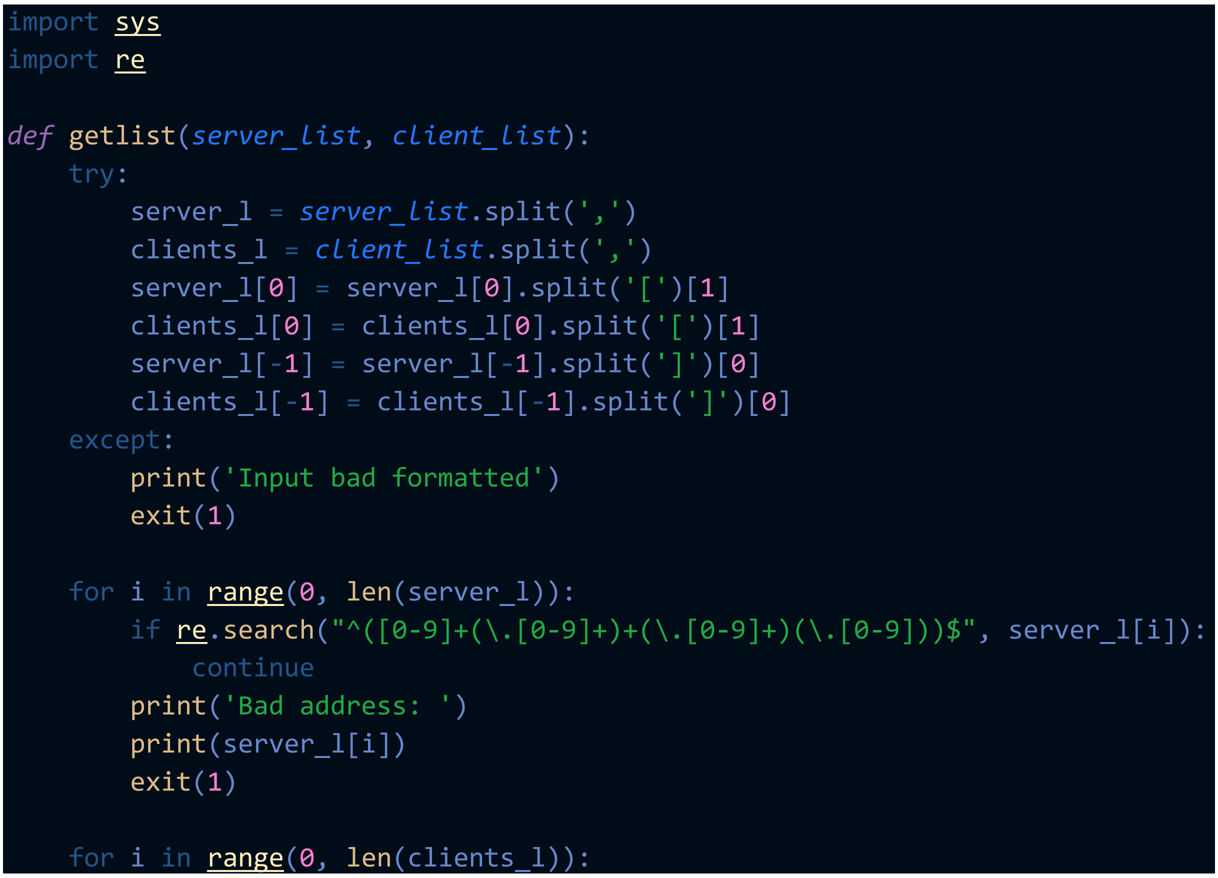


```
        flag_ricevuti[5] += packet[-4] #RST

with open('C:/Users/mailt/OneDrive/Desktop/project/tmp/totaltraffic.txt', 'r')
as f, open('C:/Users/mailt/OneDrive/Desktop/project/tmp/countedtraffic.txt',
'w') as f1:
    for line in f:
        packet = line.split(', ')
        k += 1
        try:
            packet[0] = packet[0].split('[')[1]
            packet[-1] = packet[-1].split(']')[0]
            packet[0] = packet[0].split("'")[1]
            packet[8] = packet[8].split("'")[1]
            packet[10] = packet[10].split("'")[1]
        except:
            print("Error during packet reading")
            print("packet" + str(packet))
            if input('Press enter to ignore: ') != '\n':
                exit(1)

        for i in range(0, 28):
            if i == 0 or i == 8 or i == 10:
                continue

            packet[i] = int(packet[i])

        if packet[10] in clients:
            data.append('Attack')
            frame_byte_client[0] += 1
            frame_byte_client[2] += packet[2]
            frame_byte_server[1] += 1
            frame_byte_server[3] += packet[2]

        elif packet[10] in servers:
            data.append('Normal')
            frame_byte_client[1] += 1
            frame_byte_client[3] += packet[2]
            frame_byte_server[0] += 1
            frame_byte_server[2] += packet[2]
        else:
            print(k)
            print('OTHER IP FOUNDED:' + packet[10])
            continue

        flag_check(packet)

        for i in range(0, 4):
```

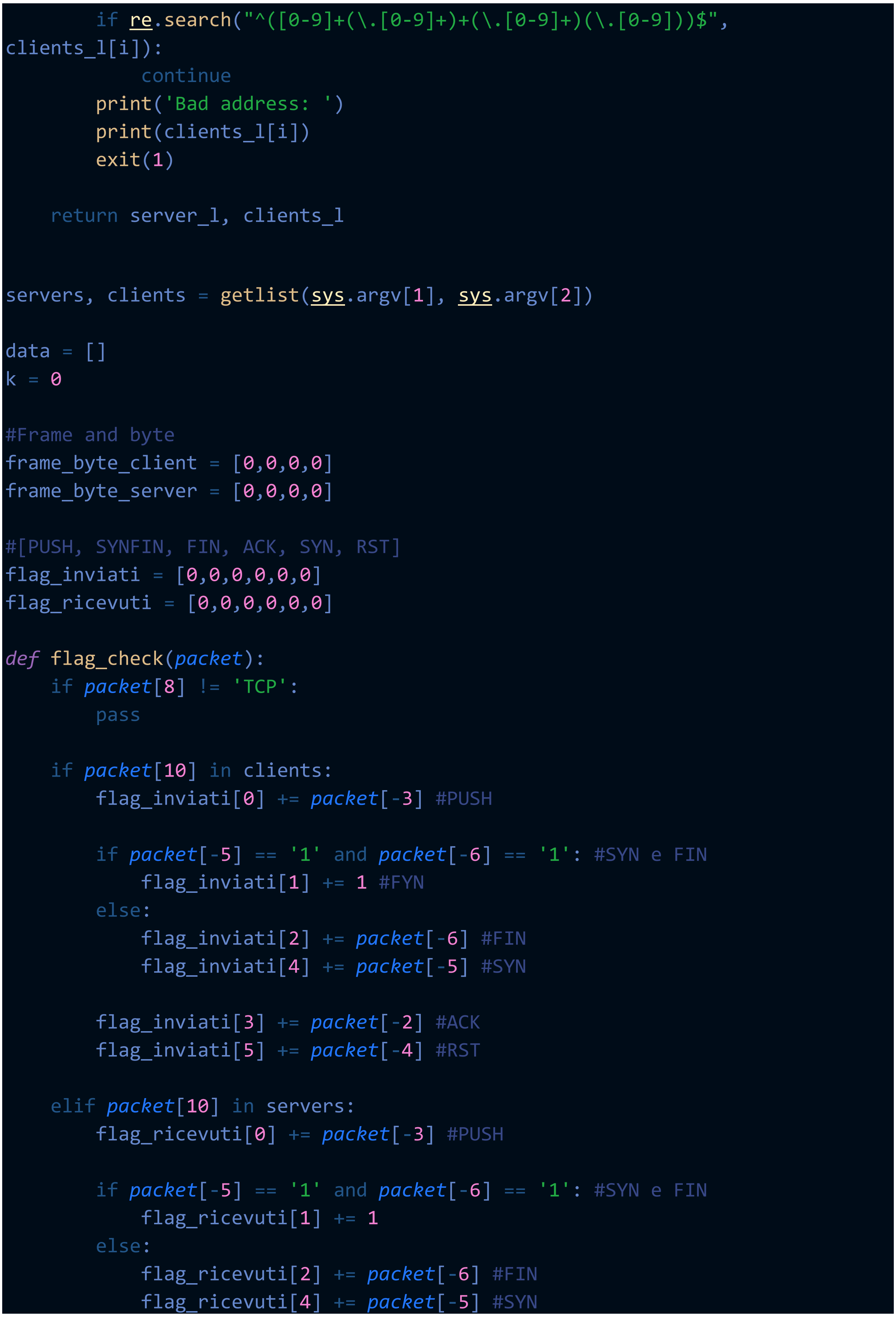


```
            data.append(frame_byte_client[i])

        for i in range(0, 6):
            data.append(flag_inviati[i])
            data.append(flag_ricevuti[i])

        for i in range(0, 4):
            data.append(frame_byte_server[i])

        data.append(packet[10])

        #packet.pop(10)
        """
        for i in range(0, len(data)):
            packet.append(data[i])"""

        f1.write(str(data))
        f1.write('\n')
        #print(k)
        data = []
```

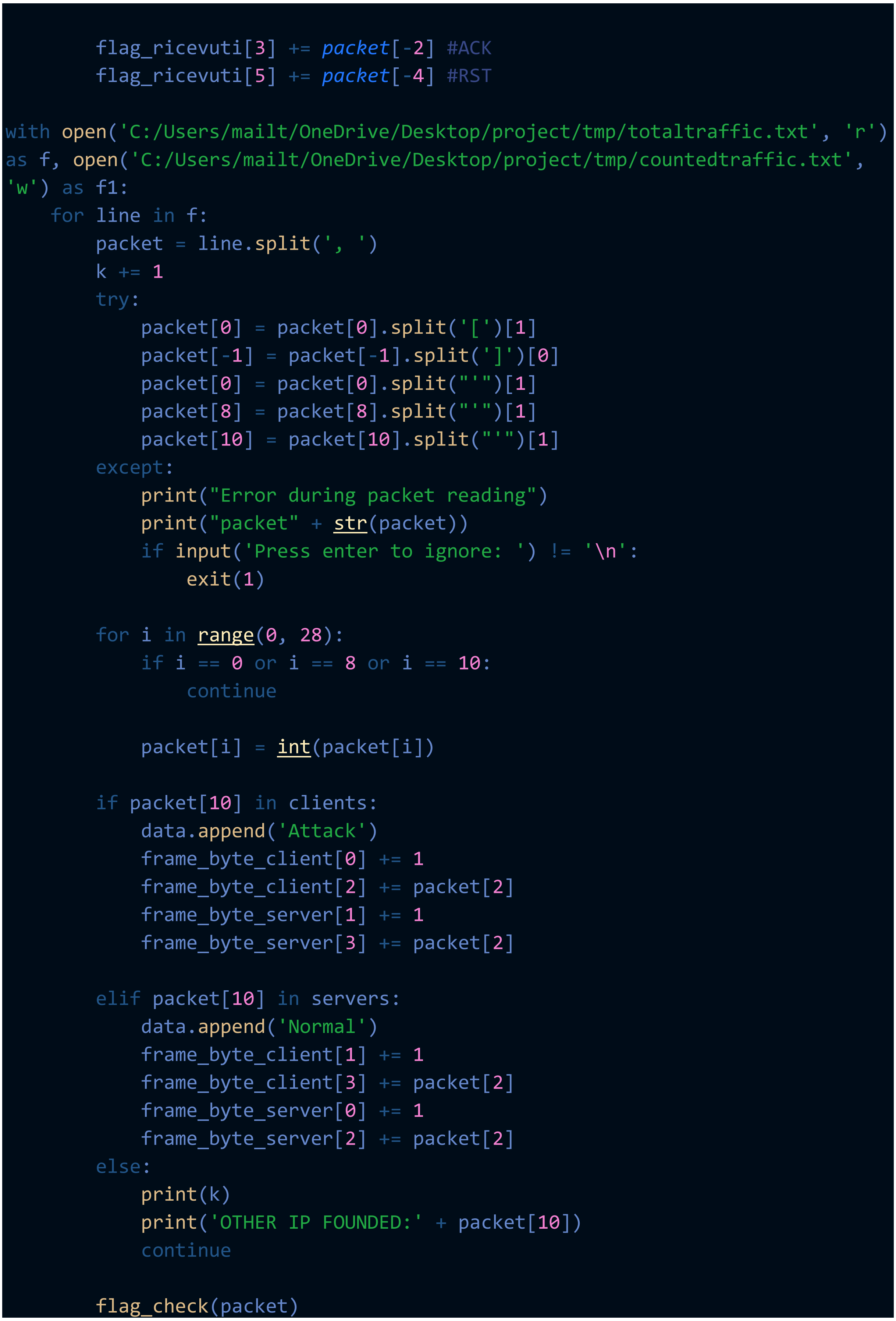
CounterHistory.py:



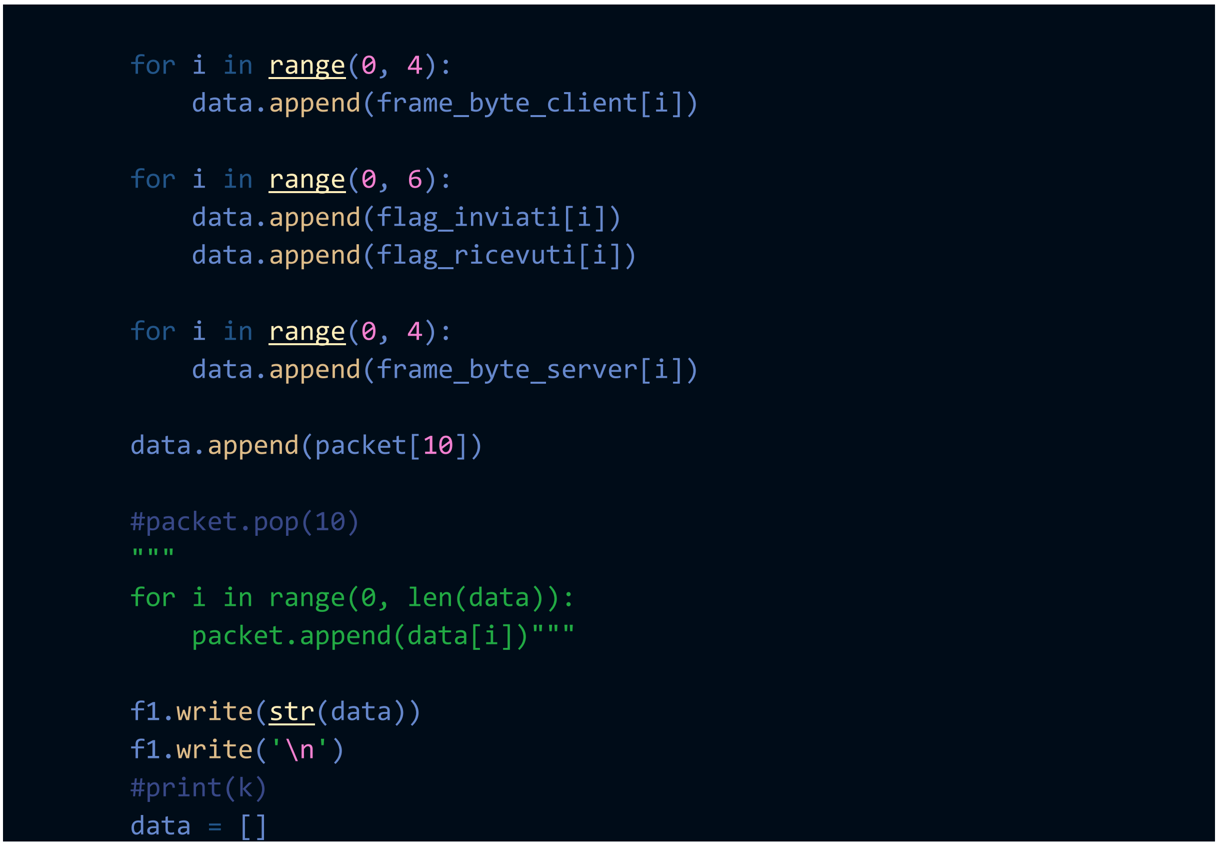
```
import sys
import re

def getlist(server_list, client_list):
    try:
        server_l = server_list.split(',')
        clients_l = client_list.split(',')
        server_l[0] = server_l[0].split('[')[1]
        clients_l[0] = clients_l[0].split('[')[1]
        server_l[-1] = server_l[-1].split(']')[0]
        clients_l[-1] = clients_l[-1].split(']')[0]
    except:
        print('Input bad formatted')
        exit(1)

    for i in range(0, len(server_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$", server_l[i]):
            continue
        print('Bad address: ')
        print(server_l[i])
        exit(1)

    for i in range(0, len(clients_l)):
```



```
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$",
clients_l[i]):
            continue
        print('Bad address: ')
        print(clients_l[i])
        exit(1)

    return server_l, clients_l


servers, clients = getlist(sys.argv[1], sys.argv[2])

data = []
k = 0

#Frame and byte
frame_byte_client = [0,0,0,0]
frame_byte_server = [0,0,0,0]

#[PUSH, SYNFIN, FIN, ACK, SYN, RST]
flag_inviati = [0,0,0,0,0,0]
flag_ricevuti = [0,0,0,0,0,0]

def flag_check(packet):
    if packet[8] != 'TCP':
        pass

    if packet[10] in clients:
        flag_inviati[0] += packet[-3] #PUSH

        if packet[-5] == '1' and packet[-6] == '1': #SYN e FIN
            flag_inviati[1] += 1 #FYN
        else:
            flag_inviati[2] += packet[-6] #FIN
            flag_inviati[4] += packet[-5] #SYN

        flag_inviati[3] += packet[-2] #ACK
        flag_inviati[5] += packet[-4] #RST

    elif packet[10] in servers:
        flag_ricevuti[0] += packet[-3] #PUSH

        if packet[-5] == '1' and packet[-6] == '1': #SYN e FIN
            flag_ricevuti[1] += 1
        else:
            flag_ricevuti[2] += packet[-6] #FIN
            flag_ricevuti[4] += packet[-5] #SYN
```

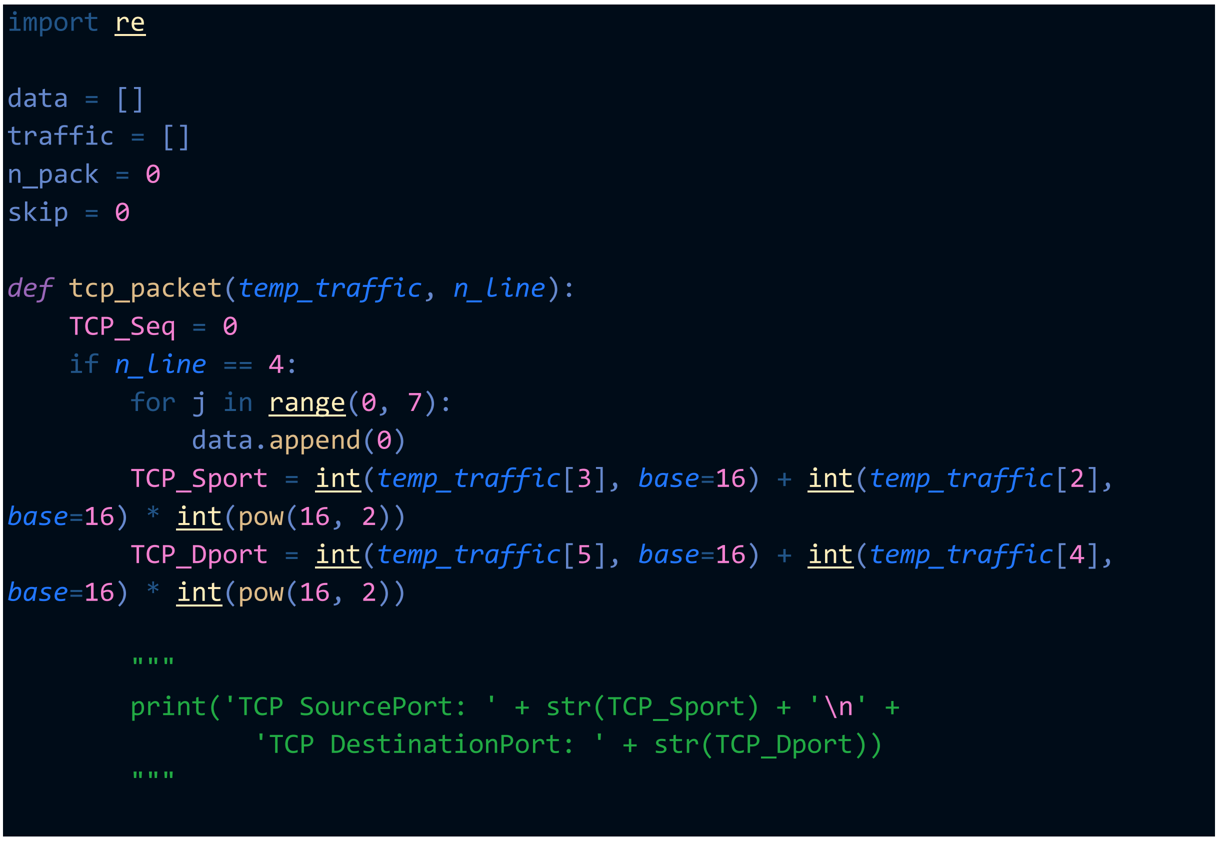


```
        flag_ricevuti[3] += packet[-2] #ACK
        flag_ricevuti[5] += packet[-4] #RST

with open('C:/Users/mailt/OneDrive/Desktop/project/tmp/totaltraffic.txt', 'r')
as f, open('C:/Users/mailt/OneDrive/Desktop/project/tmp/countedtraffic.txt',
'w') as f1:
    for line in f:
        packet = line.split(', ')
        k += 1
        try:
            packet[0] = packet[0].split('[')[1]
            packet[-1] = packet[-1].split(']')[0]
            packet[0] = packet[0].split("'")[1]
            packet[8] = packet[8].split("'")[1]
            packet[10] = packet[10].split("'")[1]
        except:
            print("Error during packet reading")
            print("packet" + str(packet))
            if input('Press enter to ignore: ') != '\n':
                exit(1)

        for i in range(0, 28):
            if i == 0 or i == 8 or i == 10:
                continue

            packet[i] = int(packet[i])

        if packet[10] in clients:
            data.append('Attack')
            frame_byte_client[0] += 1
            frame_byte_client[2] += packet[2]
            frame_byte_server[1] += 1
            frame_byte_server[3] += packet[2]

        elif packet[10] in servers:
            data.append('Normal')
            frame_byte_client[1] += 1
            frame_byte_client[3] += packet[2]
            frame_byte_server[0] += 1
            frame_byte_server[2] += packet[2]
        else:
            print(k)
            print('OTHER IP FOUNDED:' + packet[10])
            continue

        flag_check(packet)
```



```
        for i in range(0, 4):
            data.append(frame_byte_client[i])

        for i in range(0, 6):
            data.append(flag_inviati[i])
            data.append(flag_ricevuti[i])

        for i in range(0, 4):
            data.append(frame_byte_server[i])

        data.append(packet[10])

        #packet.pop(10)
        """
        for i in range(0, len(data)):
            packet.append(data[i])"""

        f1.write(str(data))
        f1.write('\n')
        #print(k)
        data = []
```

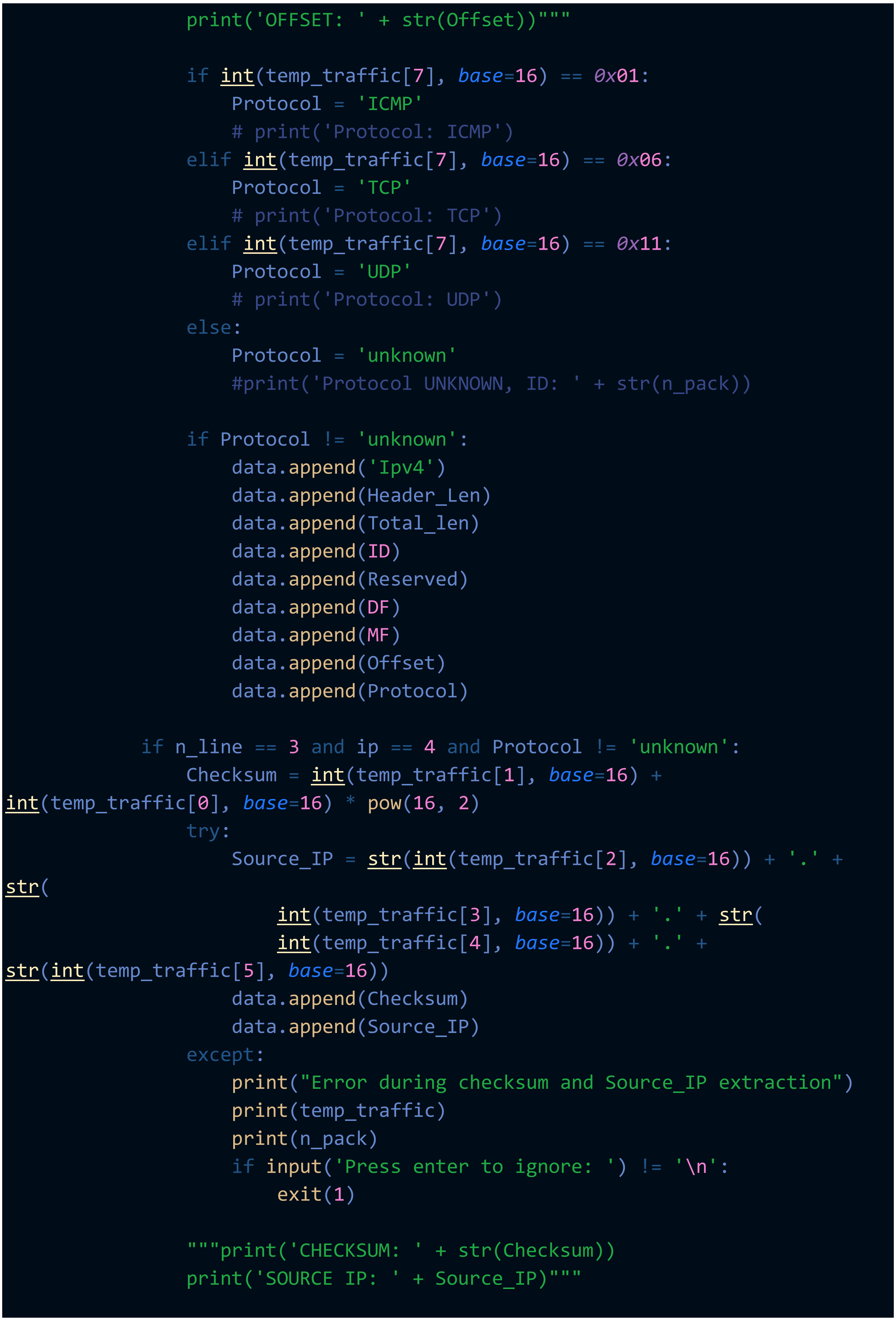
extractor.py:



```
import re

data = []
traffic = []
n_pack = 0
skip = 0

def tcp_packet(temp_traffic, n_line):
    TCP_Seq = 0
    if n_line == 4:
        for j in range(0, 7):
            data.append(0)
        TCP_Sport = int(temp_traffic[3], base=16) + int(temp_traffic[2],
base=16) * int(pow(16, 2))
        TCP_Dport = int(temp_traffic[5], base=16) + int(temp_traffic[4],
base=16) * int(pow(16, 2))

        """
        print('TCP SourcePort: ' + str(TCP_Sport) + '\n' +
                'TCP DestinationPort: ' + str(TCP_Dport))
        """
        data.append(TCP_Sport)
        data.append(TCP_Dport)

        TCP_Seq = int(temp_traffic[7], base=16) * int(pow(16, 4)) +
int(temp_traffic[6], base=16) * int(pow(16, 3))
    if n_line == 5:
        TCP_Seq += int(temp_traffic[1], base=16) + int(temp_traffic[0],
base=16) * int(pow(16, 2))
        TCP_Ack = int(temp_traffic[5], base=16) + int(temp_traffic[4],
base=16) * int(pow(16, 2)) + int(temp_traffic[3],

                          base=16) * int(
            pow(16, 3)) + int(temp_traffic[2], base=16) * int(pow(16, 4))
        TCP_Ffin = int(temp_traffic[7], base=16) & 0x1
        TCP_Fsyn = int((int(temp_traffic[7], base=16) & 0x2) / 2)
        TCP_Frst = int((int(temp_traffic[7], base=16) & 0x4) / 4)
        TCP_Fpush = int((int(temp_traffic[7], base=16) & 0x8) / 8)
        TCP_Fack = int((int(temp_traffic[7], base=16) & 0x10) / 16)
        TCP_Furg = int((int(temp_traffic[7], base=16) & 0x20) / 32)

        """print('Seq: ' + str(TCP_Seq))
        print('Ack: ' + str(TCP_Ack))
        print('TCP_Ffin: {}, TCP_Fsyn: {}, TCP_Frst: {}, TCP_Fpush: {},
TCP_Fack: {}, TCP_Furg: {}'.format(TCP_Ffin, TCP_Fsyn, TCP_Frst, TCP_Fpush,
TCP_Fack, TCP_Furg))
        """


        data.append(TCP_Seq)
        data.append(TCP_Ack)
        data.append(TCP_Ffin)
        data.append(TCP_Fsyn)
        data.append(TCP_Frst)
        data.append(TCP_Fpush)
        data.append(TCP_Fack)
        data.append(TCP_Furg)
    else:
        pass


def udp_packet(temp_traffic, n_line):
    if n_line == 4:
        UDP_Sport = int(temp_traffic[3], base=16) + int(temp_traffic[2],
base=16) * int(pow(16, 2))
        UDP_Dport = int(temp_traffic[5], base=16) + int(temp_traffic[4],
base=16) * int(pow(16, 2))
        UDP_Len = int(temp_traffic[7], base=16) + int(temp_traffic[6],
base=16) * int(pow(16, 2))
```

```
        """print('UDP SourcePort: ' + str(UDP_Sport) + '\n' +
              'UDP DestinationPort: ' + str(UDP_Dport) + '\n' +
              'UDP Lenght: ' + str(UDP_Len))"""

        data.append(UDP_Sport)
        data.append(UDP_Dport)
        data.append(UDP_Len)

    elif n_line == 5:
        UDP_Checksum = int(temp_traffic[1], base=16) + int(temp_traffic[0],
base=16) * int(pow(16, 2))

        # print('UDP Checksum: ' + str(UDP_Checksum))

        data.append(UDP_Checksum)

        for j in range(0, 13):
            data.append(0)
    else:
        pass


def icmp_packet(temp_traffic, n_line):
    if n_line == 4:
        for j in range(0, 4):
            data.append(0)

        ICMP_Type = int(temp_traffic[2], base=16)
        ICMP_Code = int(temp_traffic[3], base=16)
        ICMP_Checksum = int(temp_traffic[5], base=16) + int(temp_traffic[4],
base=16) * int(pow(16, 2))

        """print('ICMP Type: ' + str(ICMP_Type))
        print('ICMP Code: ' + str(ICMP_Code))
        print('ICMP Checksum: ' + str(ICMP_Checksum))"""

        data.append(ICMP_Type)
        data.append(ICMP_Code)
        data.append(ICMP_Checksum)

        for j in range(0, 10):
            data.append(0)
    else:
        pass


temp_traffic = []
```

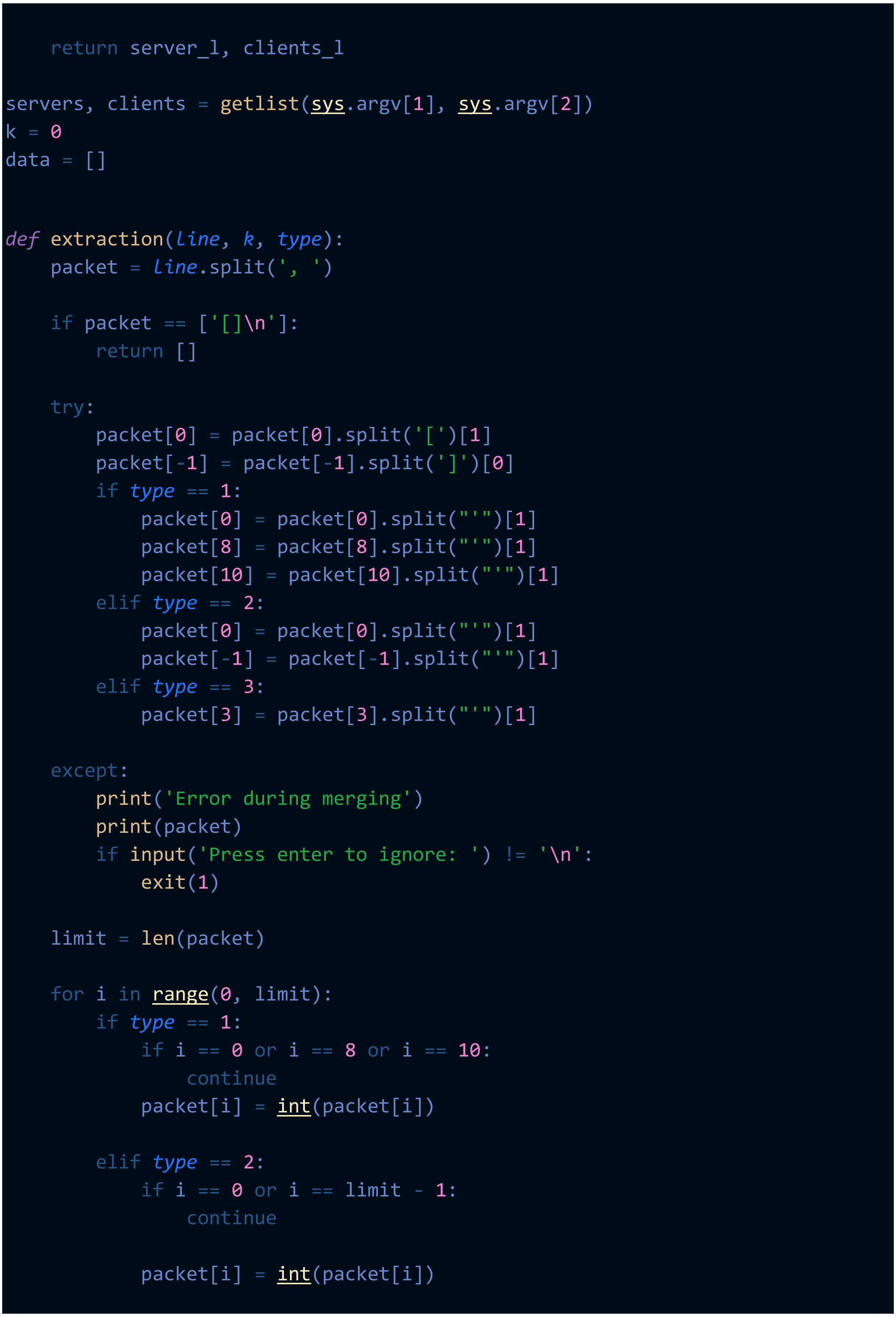
```
n_line = 0
with open("totaltraffic.c", "r") as f,
open("C:/Users/mailt/OneDrive/Desktop/project/tmp/totaltraffic.txt", "w") as
f1:
    while True:
        # Read line
        line = f.readline()

        # Check EOF
        if not line:
            break

        # Check blank line or packet's first line (comment)
        if line.strip() and not bool(re.search("^(\/\*) Frame \(*[0-9]*
bytes\) (\*\/)$", line)):

            # Check End of Line
            if bool(re.search("};\n", line)):

                f1.write(str(data))
                f1.write('\n')
                skip = 0
                n_pack += 1
                n_line = 0
                data = []
                continue

            if skip:
                continue

            if bool(re.search("\/\* Reassembled SMTP \(11 bytes\) \*\/",
line)):
                skip = 1
                continue

            # Check array C declaration
            if bool(re.search("static const unsigned char pkt*[0-9]*\[*[0-
9]*\] = {", line)):
                dim = re.split("[\[\]]", line)[1]
                continue

            # Analyze packet: it has 8 columns composed by hexadecimal bytes
and 16 ASCII bytes
            temp_traffic = line.split(', ')

            # Remove 16 ASCII bytes alongside the data offset (last item)
            for i in range(0, len(temp_traffic)):
                if re.search("\/\* .* \*\/", temp_traffic[i]):
                    if re.split(" *\/\* .* \*\/", temp_traffic[i])[0] == '':
                        temp_traffic.pop(i)
                    else:
                        temp_traffic[i] = re.split(" *\/\* .* \*\/",
temp_traffic[i])[0]

            if n_line == 1:
                try:
                    if temp_traffic[4] == '0x08' and temp_traffic[5] ==
'0x00':
                        # print('Type of IP: Ipv4')
                        ip = 4
                        Header_Len = int((int(temp_traffic[6], base=16) & 0xf)
* 32 / 8)
                        # print('Header_Len: ' + str(Header_Len))
                    elif temp_traffic[5] == '0xdd':
                        ip = 6
                    else:
                        ip = 0
                        # print('Protocol: ' + str(n_pack))
                        print(temp_traffic)

                except:
                    print('Exception during extraction: ' + str(n_pack))
                    if input('Press enter to ignore: ') != '\n':
                        exit(1)

            if n_line == 2 and ip == 4:
                Total_len = int(temp_traffic[1], base=16) +
int(temp_traffic[0], base=16) * int(pow(16, 2))
                # print('Total Len: ' + str(Total_len))

                ID = int(temp_traffic[3], base=16) + int(temp_traffic[2],
base=16) * int(pow(16, 2))
                # print('ID: ' + str(ID))

                Reserved = int((int(temp_traffic[4], base=16) & 0x80) / 128)
                DF = int((int(temp_traffic[4], base=16) & 0x40) / 64)
                MF = int((int(temp_traffic[4], base=16) & 0x20) / 32)
                Offset = int(temp_traffic[5], base=16) +
int(int(temp_traffic[4], base=16) & 0x1f) * int(pow(16, 2))

                """print('FLAGS: \n' +
                      'Reserved: ' + str(Reserved) + '\n' +
                      "Don't Fragments: " + str(DF) + '\n' +
                      "More Fragments: " + str(MF))
```



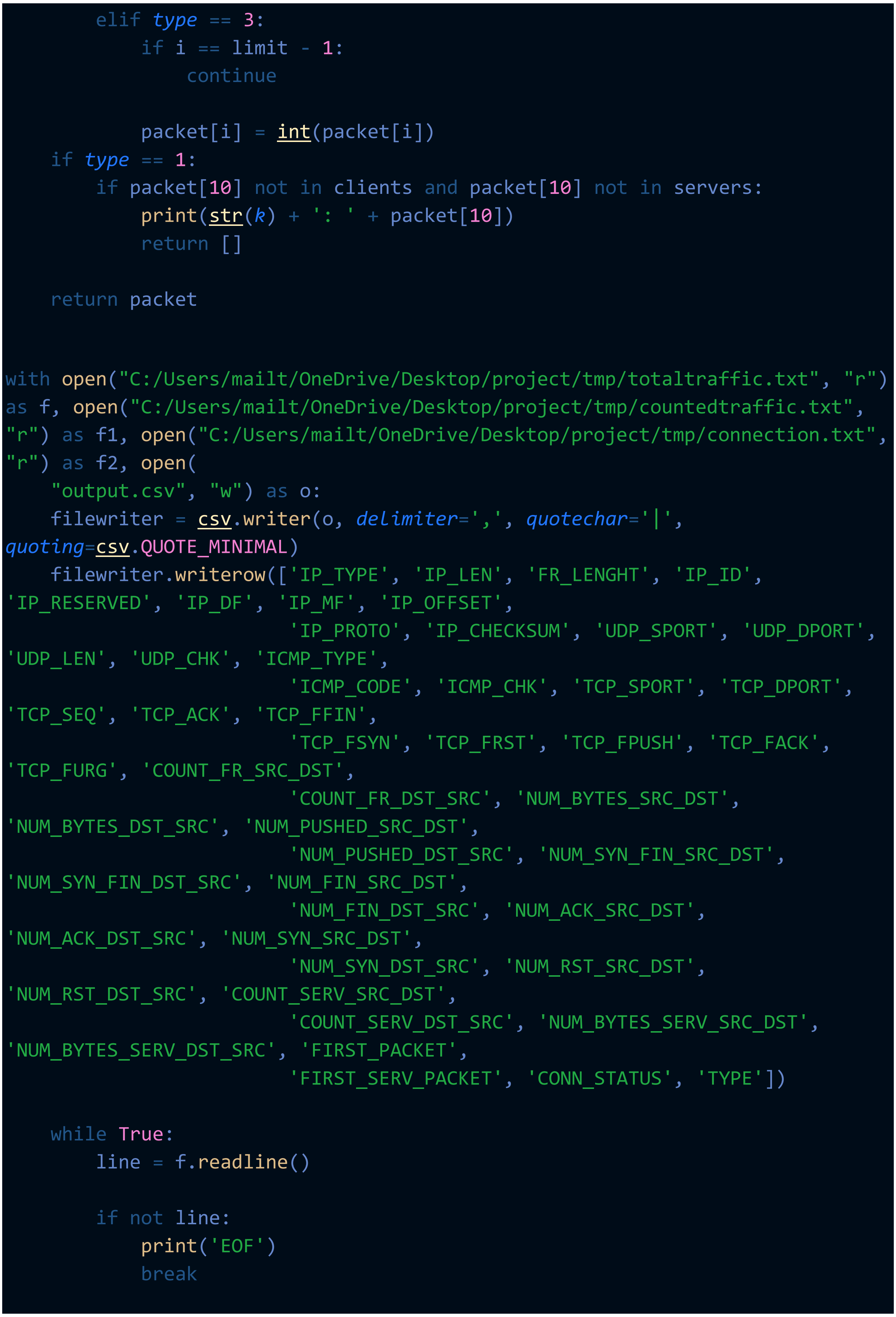
```
                print('OFFSET: ' + str(Offset))"""

                if int(temp_traffic[7], base=16) == 0x01:
                    Protocol = 'ICMP'
                    # print('Protocol: ICMP')
                elif int(temp_traffic[7], base=16) == 0x06:
                    Protocol = 'TCP'
                    # print('Protocol: TCP')
                elif int(temp_traffic[7], base=16) == 0x11:
                    Protocol = 'UDP'
                    # print('Protocol: UDP')
                else:
                    Protocol = 'unknown'
                    #print('Protocol UNKNOWN, ID: ' + str(n_pack))

                if Protocol != 'unknown':
                    data.append('Ipv4')
                    data.append(Header_Len)
                    data.append(Total_len)
                    data.append(ID)
                    data.append(Reserved)
                    data.append(DF)
                    data.append(MF)
                    data.append(Offset)
                    data.append(Protocol)

            if n_line == 3 and ip == 4 and Protocol != 'unknown':
                Checksum = int(temp_traffic[1], base=16) +
int(temp_traffic[0], base=16) * pow(16, 2)
                try:
                    Source_IP = str(int(temp_traffic[2], base=16)) + '.' +
str(
                        int(temp_traffic[3], base=16)) + '.' + str(
                        int(temp_traffic[4], base=16)) + '.' +
str(int(temp_traffic[5], base=16))
                    data.append(Checksum)
                    data.append(Source_IP)
                except:
                    print("Error during checksum and Source_IP extraction")
                    print(temp_traffic)
                    print(n_pack)
                    if input('Press enter to ignore: ') != '\n':
                        exit(1)

                """print('CHECKSUM: ' + str(Checksum))
                print('SOURCE IP: ' + Source_IP)"""
            # Enter in header protocol used
            if n_line > 3 and ip == 4:
                if Protocol == 'ICMP':
                    icmp_packet(temp_traffic, n_line)
                elif Protocol == 'TCP':
                    tcp_packet(temp_traffic, n_line)
                elif Protocol == 'UDP':
                    udp_packet(temp_traffic, n_line)

            # Next line
            n_line += 1
```

mergefile.py:

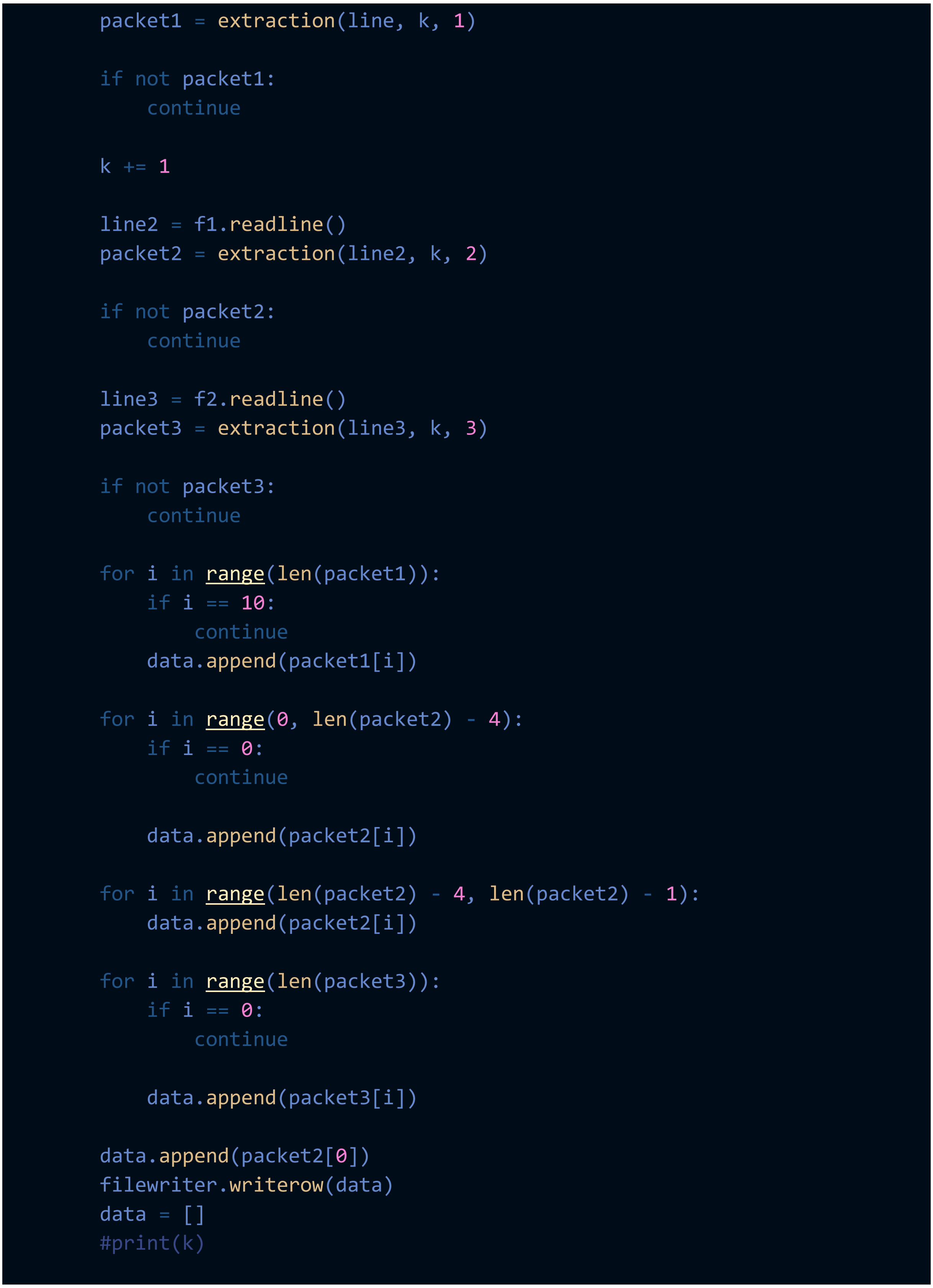
```
import csv
import re
import sys




def getlist(server_list, client_list):
    try:
        server_l = server_list.split(',')
        clients_l = client_list.split(',')
        server_l[0] = server_l[0].split('[')[1]
        clients_l[0] = clients_l[0].split('[')[1]
        server_l[-1] = server_l[-1].split(']')[0]
        clients_l[-1] = clients_l[-1].split(']')[0]
    except:
        print('Input bad formatted')
        exit(1)

    for i in range(0, len(server_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$", server_l[i]):
            continue
        print('Bad address: ')
        print(server_l[i])
        exit(1)

    for i in range(0, len(clients_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$",
clients_l[i]):
            continue
        print('Bad address: ')
        print(clients_l[i])
        exit(1)
```



```
    return server_l, clients_l

servers, clients = getlist(sys.argv[1], sys.argv[2])
k = 0
data = []


def extraction(line, k, type):
    packet = line.split(', ')

    if packet == ['[]\n']:
        return []

    try:
        packet[0] = packet[0].split('[')[1]
        packet[-1] = packet[-1].split(']')[0]
        if type == 1:
            packet[0] = packet[0].split("'")[1]
            packet[8] = packet[8].split("'")[1]
            packet[10] = packet[10].split("'")[1]
        elif type == 2:
            packet[0] = packet[0].split("'")[1]
            packet[-1] = packet[-1].split("'")[1]
        elif type == 3:
            packet[3] = packet[3].split("'")[1]

    except:
        print('Error during merging')
        print(packet)
        if input('Press enter to ignore: ') != '\n':
            exit(1)

    limit = len(packet)

    for i in range(0, limit):
        if type == 1:
            if i == 0 or i == 8 or i == 10:
                continue
            packet[i] = int(packet[i])

        elif type == 2:
            if i == 0 or i == limit - 1:
                continue

            packet[i] = int(packet[i])
```

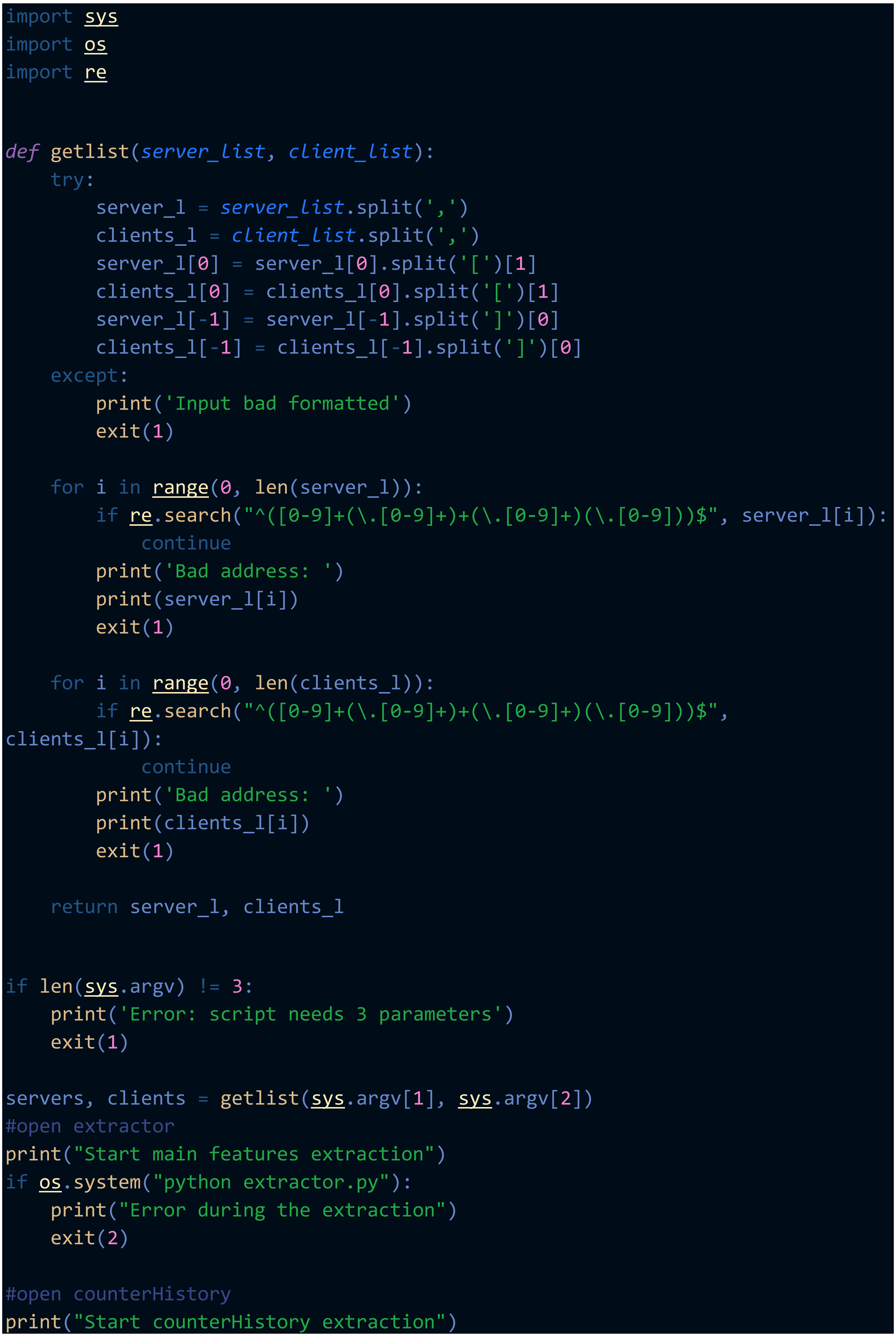


```
        elif type == 3:
            if i == limit - 1:
                continue

            packet[i] = int(packet[i])
    if type == 1:
        if packet[10] not in clients and packet[10] not in servers:
            print(str(k) + ': ' + packet[10])
            return []

    return packet


with open("C:/Users/mailt/OneDrive/Desktop/project/tmp/totaltraffic.txt", "r")
as f, open("C:/Users/mailt/OneDrive/Desktop/project/tmp/countedtraffic.txt",
"r") as f1, open("C:/Users/mailt/OneDrive/Desktop/project/tmp/connection.txt",
"r") as f2, open(
    "output.csv", "w") as o:
    filewriter = csv.writer(o, delimiter=',', quotechar='|',
quoting=csv.QUOTE_MINIMAL)
    filewriter.writerow(['IP_TYPE', 'IP_LEN', 'FR_LENGHT', 'IP_ID',
'IP_RESERVED', 'IP_DF', 'IP_MF', 'IP_OFFSET',
                         'IP_PROTO', 'IP_CHECKSUM', 'UDP_SPORT', 'UDP_DPORT',
'UDP_LEN', 'UDP_CHK', 'ICMP_TYPE',
                         'ICMP_CODE', 'ICMP_CHK', 'TCP_SPORT', 'TCP_DPORT',
'TCP_SEQ', 'TCP_ACK', 'TCP_FFIN',
                         'TCP_FSYN', 'TCP_FRST', 'TCP_FPUSH', 'TCP_FACK',
'TCP_FURG', 'COUNT_FR_SRC_DST',
                         'COUNT_FR_DST_SRC', 'NUM_BYTES_SRC_DST',
'NUM_BYTES_DST_SRC', 'NUM_PUSHED_SRC_DST',
                         'NUM_PUSHED_DST_SRC', 'NUM_SYN_FIN_SRC_DST',
'NUM_SYN_FIN_DST_SRC', 'NUM_FIN_SRC_DST',
                         'NUM_FIN_DST_SRC', 'NUM_ACK_SRC_DST',
'NUM_ACK_DST_SRC', 'NUM_SYN_SRC_DST',
                         'NUM_SYN_DST_SRC', 'NUM_RST_SRC_DST',
'NUM_RST_DST_SRC', 'COUNT_SERV_SRC_DST',
                         'COUNT_SERV_DST_SRC', 'NUM_BYTES_SERV_SRC_DST',
'NUM_BYTES_SERV_DST_SRC', 'FIRST_PACKET',
                         'FIRST_SERV_PACKET', 'CONN_STATUS', 'TYPE'])

    while True:
        line = f.readline()

        if not line:
            print('EOF')
            break
```

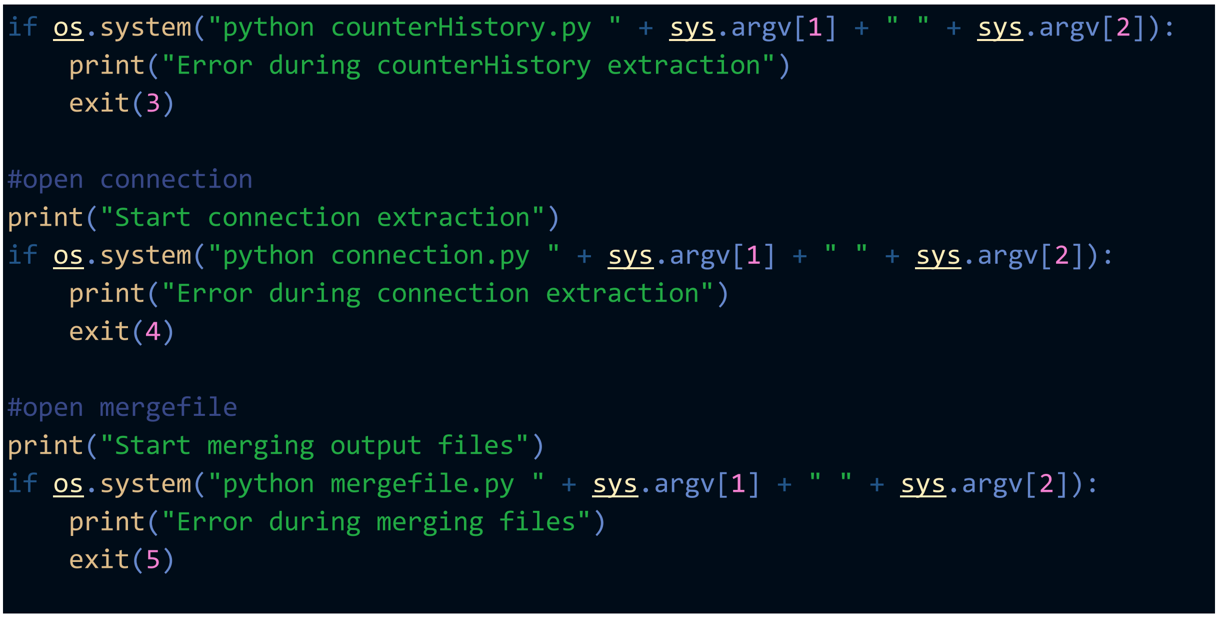


```
        packet1 = extraction(line, k, 1)

        if not packet1:
            continue

        k += 1

        line2 = f1.readline()
        packet2 = extraction(line2, k, 2)

        if not packet2:
            continue

        line3 = f2.readline()
        packet3 = extraction(line3, k, 3)

        if not packet3:
            continue

        for i in range(len(packet1)):
            if i == 10:
                continue
            data.append(packet1[i])

        for i in range(0, len(packet2) - 4):
            if i == 0:
                continue

            data.append(packet2[i])

        for i in range(len(packet2) - 4, len(packet2) - 1):
            data.append(packet2[i])

        for i in range(len(packet3)):
            if i == 0:
                continue

            data.append(packet3[i])

        data.append(packet2[0])
        filewriter.writerow(data)
        data = []
        #print(k)
```

Featureextractor.py:



```
import sys
import os
import re


def getlist(server_list, client_list):
    try:
        server_l = server_list.split(',')
        clients_l = client_list.split(',')
        server_l[0] = server_l[0].split('[')[1]
        clients_l[0] = clients_l[0].split('[')[1]
        server_l[-1] = server_l[-1].split(']')[0]
        clients_l[-1] = clients_l[-1].split(']')[0]
    except:
        print('Input bad formatted')
        exit(1)

    for i in range(0, len(server_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$", server_l[i]):
            continue
        print('Bad address: ')
        print(server_l[i])
        exit(1)

    for i in range(0, len(clients_l)):
        if re.search("^([0-9]+(\.[0-9]+)+(\.[0-9]+)(\.[0-9]))$",
clients_l[i]):
            continue
        print('Bad address: ')
        print(clients_l[i])
        exit(1)

    return server_l, clients_l


if len(sys.argv) != 3:
    print('Error: script needs 3 parameters')
    exit(1)

servers, clients = getlist(sys.argv[1], sys.argv[2])
#open extractor
print("Start main features extraction")
if os.system("python extractor.py"):
    print("Error during the extraction")
    exit(2)

#open counterHistory
print("Start counterHistory extraction")
```



```
if os.system("python counterHistory.py " + sys.argv[1] + " " + sys.argv[2]):
    print("Error during counterHistory extraction")
    exit(3)

#open connection
print("Start connection extraction")
if os.system("python connection.py " + sys.argv[1] + " " + sys.argv[2]):
    print("Error during connection extraction")
    exit(4)

#open mergefile
print("Start merging output files")
if os.system("python mergefile.py " + sys.argv[1] + " " + sys.argv[2]):
    print("Error during merging files")
    exit(5)
```

ML.ipynb:

```
import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import time


project=pd.read_csv("C:/Users/mailt/OneDrive/Desktop/project/output.csv")

project.describe()

cols=list(project.columns)

len(cols)

project.isnull().sum()

project.dtypes

num_cols = project._get_numeric_data().columns


cate_cols = list(set(project.columns)-set(num_cols))
```

```
cate_cols.remove('TYPE')
cate_cols.remove('IP_PROTO')
cate_cols
plt.hist(project[cate_cols])
df=project
df = df.dropna('columns')# drop columns with NaN

df = df[[col for col in df if df[col].nunique() > 1]]# keep columns where there are more than 1 unique values

corr = df.corr()

plt.figure(figsize =(15, 12))

sns.heatmap(corr)

plt.show()

cor_matrix = df.corr().abs()
print(cor_matrix)
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(np.bool))
print(upper_tri)
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.95)]
to_drop
df1 = df.drop(to_drop, axis=1)
df1.describe()
df1
fmap={'NEW':1,'CLOSED':2,'ESTABLISHED':3}
df1['CONN_STATUS']=df1['CONN_STATUS'].map(fmap)
df1['TYPE']=df1['TYPE'].astype('string')
df1['IP_PROTO']=df1['IP_PROTO'].astype('string')
```

```
df1.dtypes
df1.to_csv('best1.csv')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler


y=df1[['TYPE']]
x=df1.drop(['TYPE','IP_PROTO'],axis=1)
x
sc = MinMaxScaler()
x = sc.fit_transform(x)

# Split test and train data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.33, random_state = 42)
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clfg = GaussianNB()
start_time = time.time()
clfg.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfg.predict(X_train)
end_time = time.time()
```

```
print("Testing time: ", end_time-start_time)



print("Train score is:", clfg.score(X_train, y_train))
print("Test score is:", clfg.score(X_test, y_test))



from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

pred = clfg.predict(X_test)
print('Accuracy ',accuracy_score(y_test,pred))
print(classification_report(y_test,pred))
sns.heatmap(confusion_matrix(y_test,pred),annot=True,fmt='.2g')

# Decision Tree
from sklearn.tree import DecisionTreeClassifier

clfd = DecisionTreeClassifier(criterion ="entropy", max_depth = 4)
start_time = time.time()
clfd.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)



start_time = time.time()
y_test_pred = clfd.predict(X_train)
end_time = time.time()
print("Testing time: ", end_time-start_time)
print("Train score is:", clfd.score(X_train, y_train))
print("Test score is:", clfd.score(X_test, y_test))


from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

pred = clfd.predict(X_test)
print('Accuracy ',accuracy_score(y_test,pred))
print(classification_report(y_test,pred))
sns.heatmap(confusion_matrix(y_test,pred),annot=True,fmt='.2g')

from sklearn.ensemble import RandomForestClassifier

clfr = RandomForestClassifier(n_estimators = 30)
start_time = time.time()
clfr.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)


start_time = time.time()
y_test_pred = clfr.predict(X_train)
end_time = time.time()
print("Testing time: ", end_time-start_time)



print("Train score is:", clfr.score(X_train, y_train))
print("Test score is:", clfr.score(X_test, y_test))
```

```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

pred = clfr.predict(X_test)
print('Accuracy ',accuracy_score(y_test,pred))
print(classification_report(y_test,pred))
sns.heatmap(confusion_matrix(y_test,pred),annot=True,fmt='.2g')
from sklearn.svm import SVC

clfs = SVC(gamma = 'scale')
start_time = time.time()
clfs.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfs.predict(X_train)
end_time = time.time()
print("Testing time: ", end_time-start_time)

print("Train score is:", clfs.score(X_train, y_train))
print("Test score is:", clfs.score(X_test, y_test))
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

pred = clfs.predict(X_test)
print('Accuracy ',accuracy_score(y_test,pred))
print(classification_report(y_test,pred))
sns.heatmap(confusion_matrix(y_test,pred),annot=True,fmt='.2g')
from sklearn.linear_model import LogisticRegression
```

```
clfl = LogisticRegression(max_iter = 1200000)
start_time = time.time()
clfl.fit(X_train, y_train.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)

start_time = time.time()
y_test_pred = clfl.predict(X_train)
end_time = time.time()
print("Testing time: ", end_time-start_time)



print("Train score is:", clfl.score(X_train, y_train))
print("Test score is:", clfl.score(X_test, y_test))



from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

pred = clfl.predict(X_test)
print('Accuracy ',accuracy_score(y_test,pred))
print(classification_report(y_test,pred))
sns.heatmap(confusion_matrix(y_test,pred),annot=True,fmt='.2g')
```

4. Snapshots

Fig 1) Virtual client-server architecture:

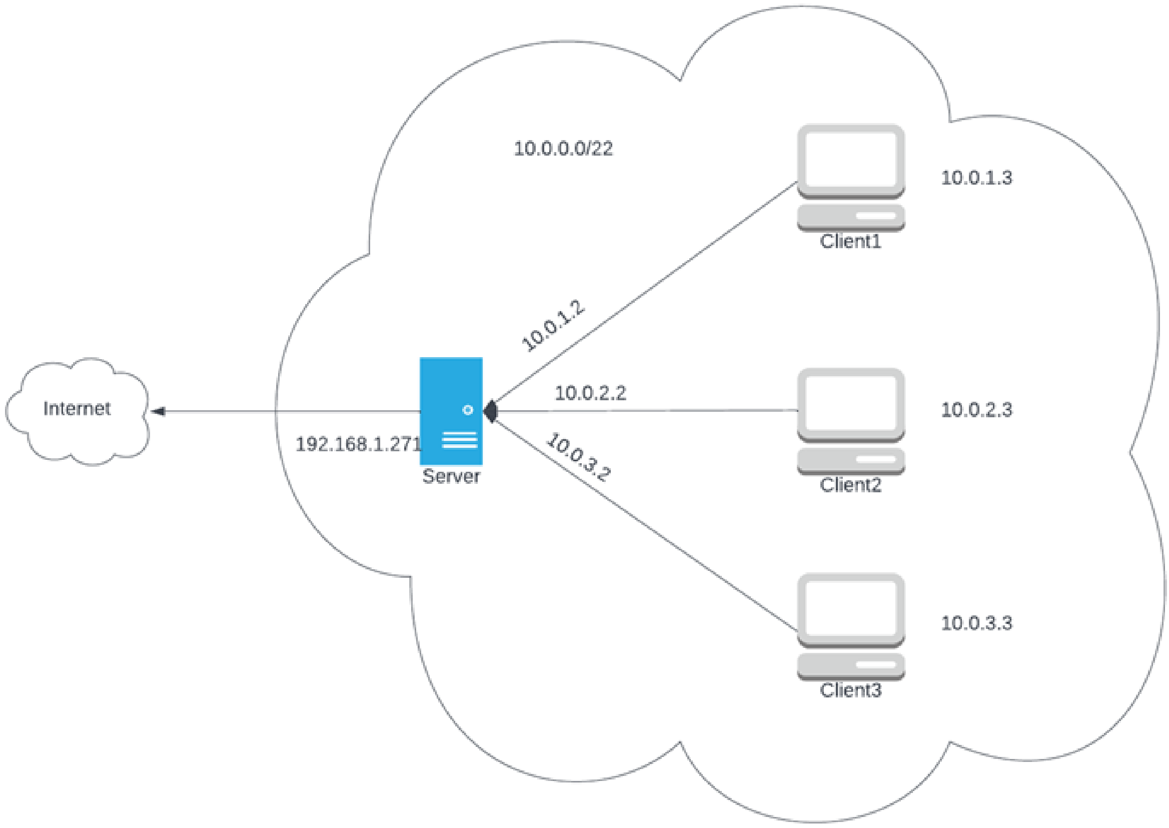


Fig 2) Linux implementation:

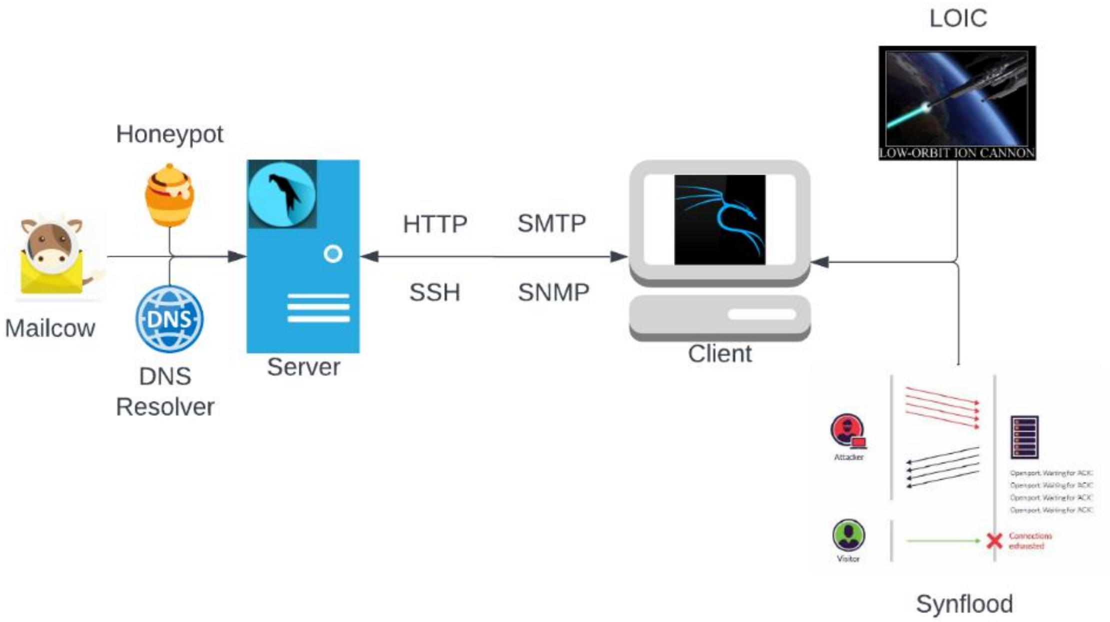
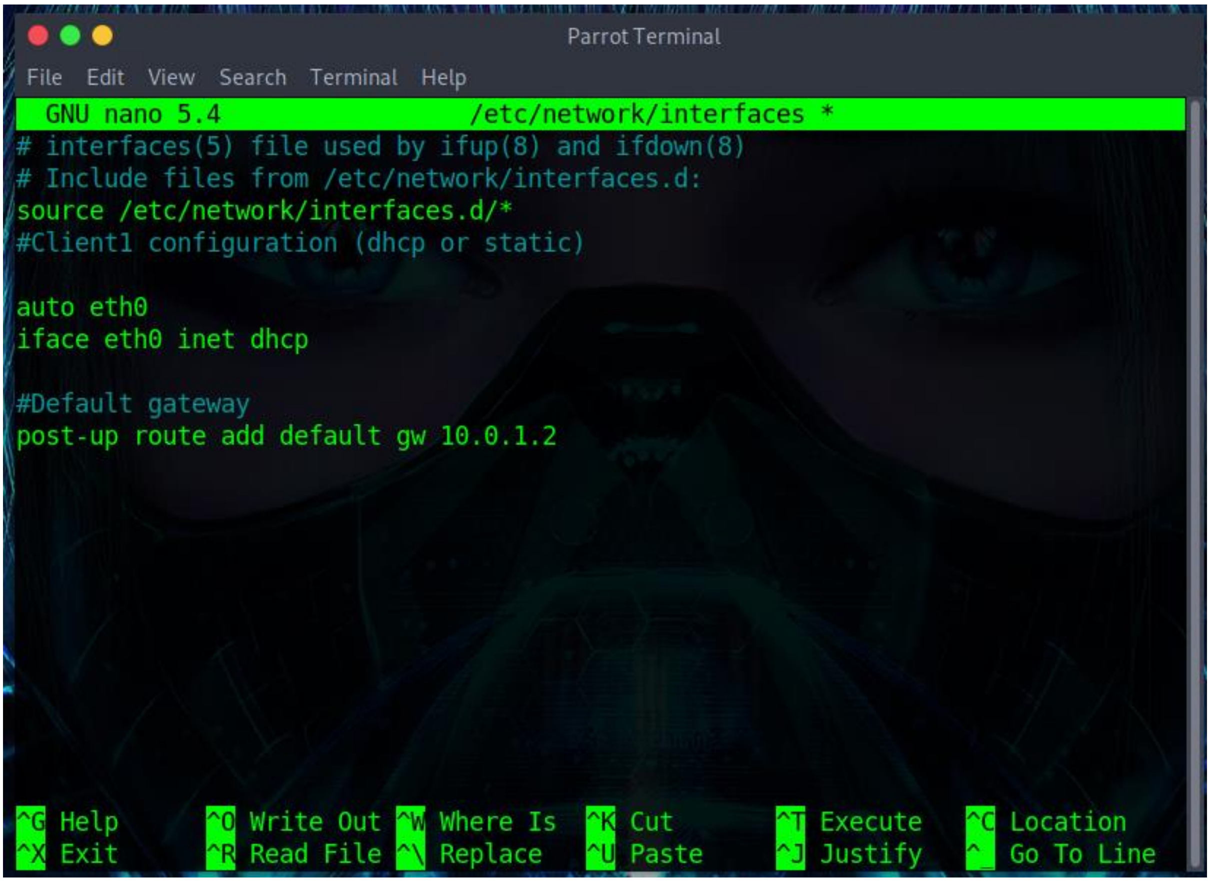


Fig 3) Network setup:



```
  GNU nano 5.4                    /etc/network/interfaces *
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*
#Client1 configuration (dhcp or static)

auto eth0
iface eth0 inet dhcp

#Default gateway
post-up route add default gw 10.0.1.2









^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

Fig 4) Attack Implementation:

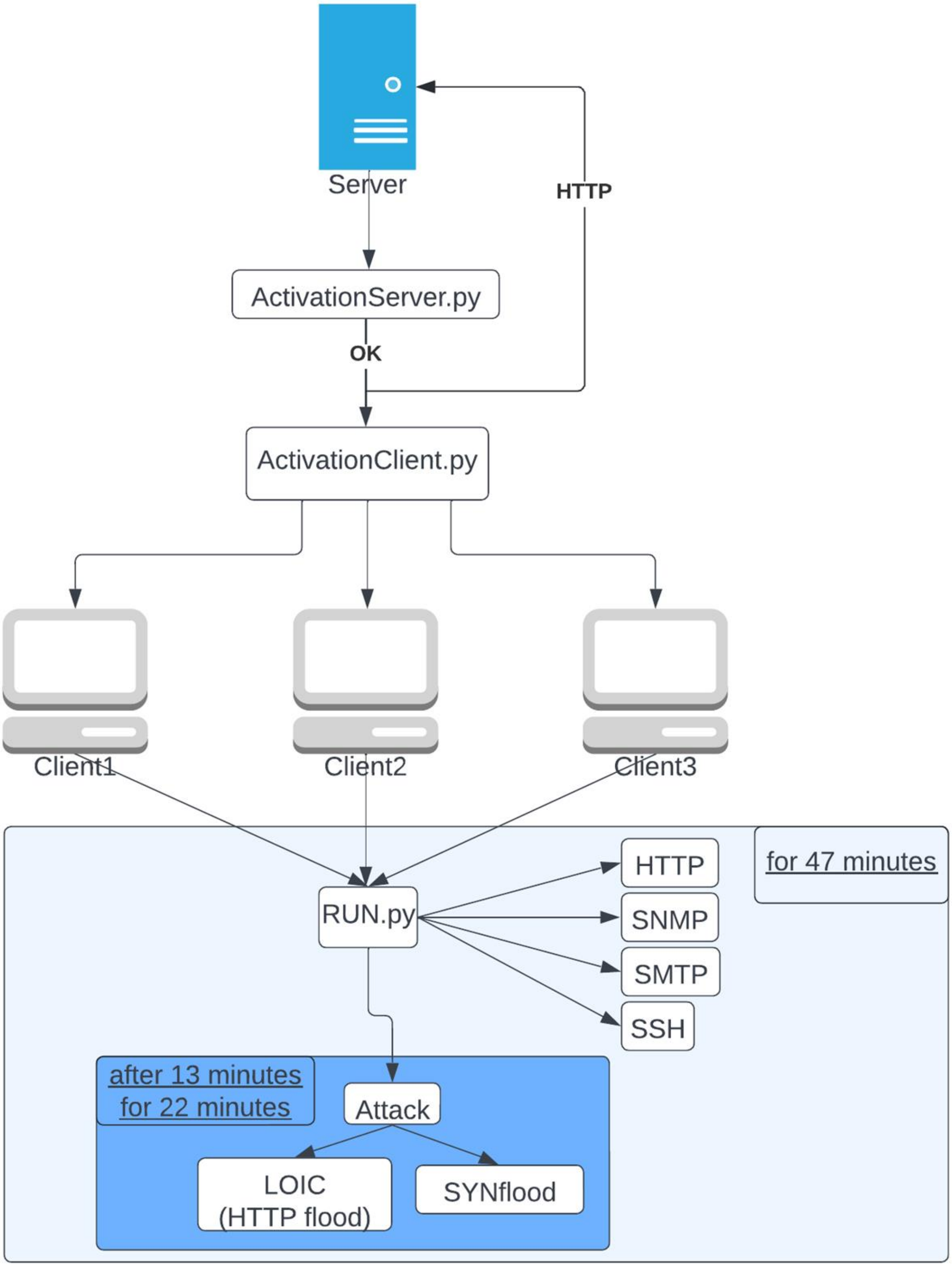
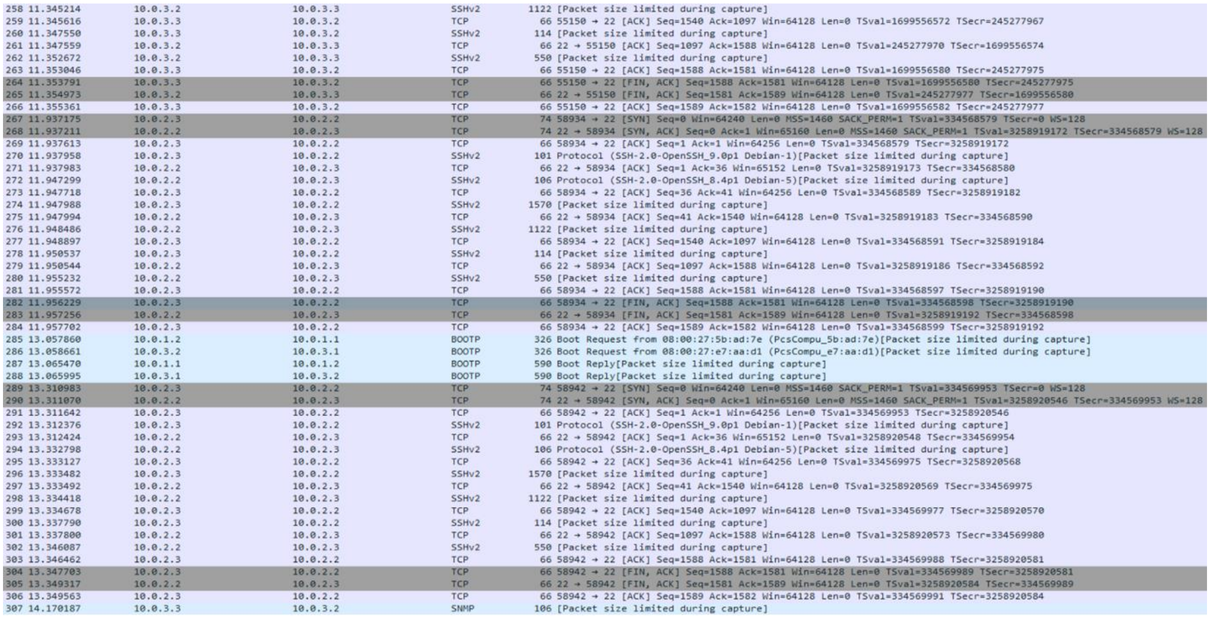
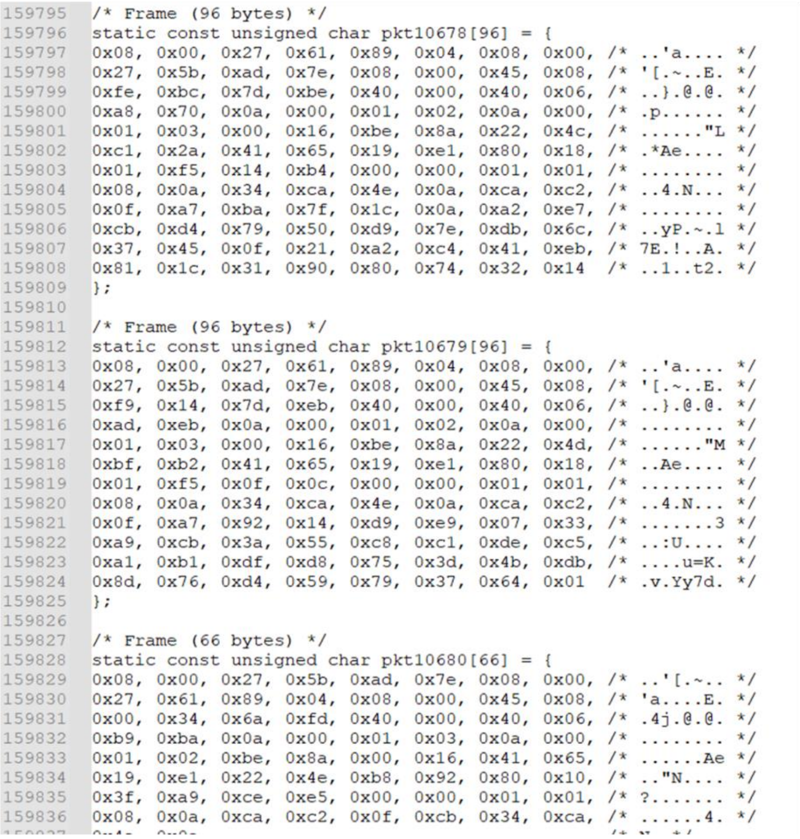


Fig 5) WireShark output converted to C array:

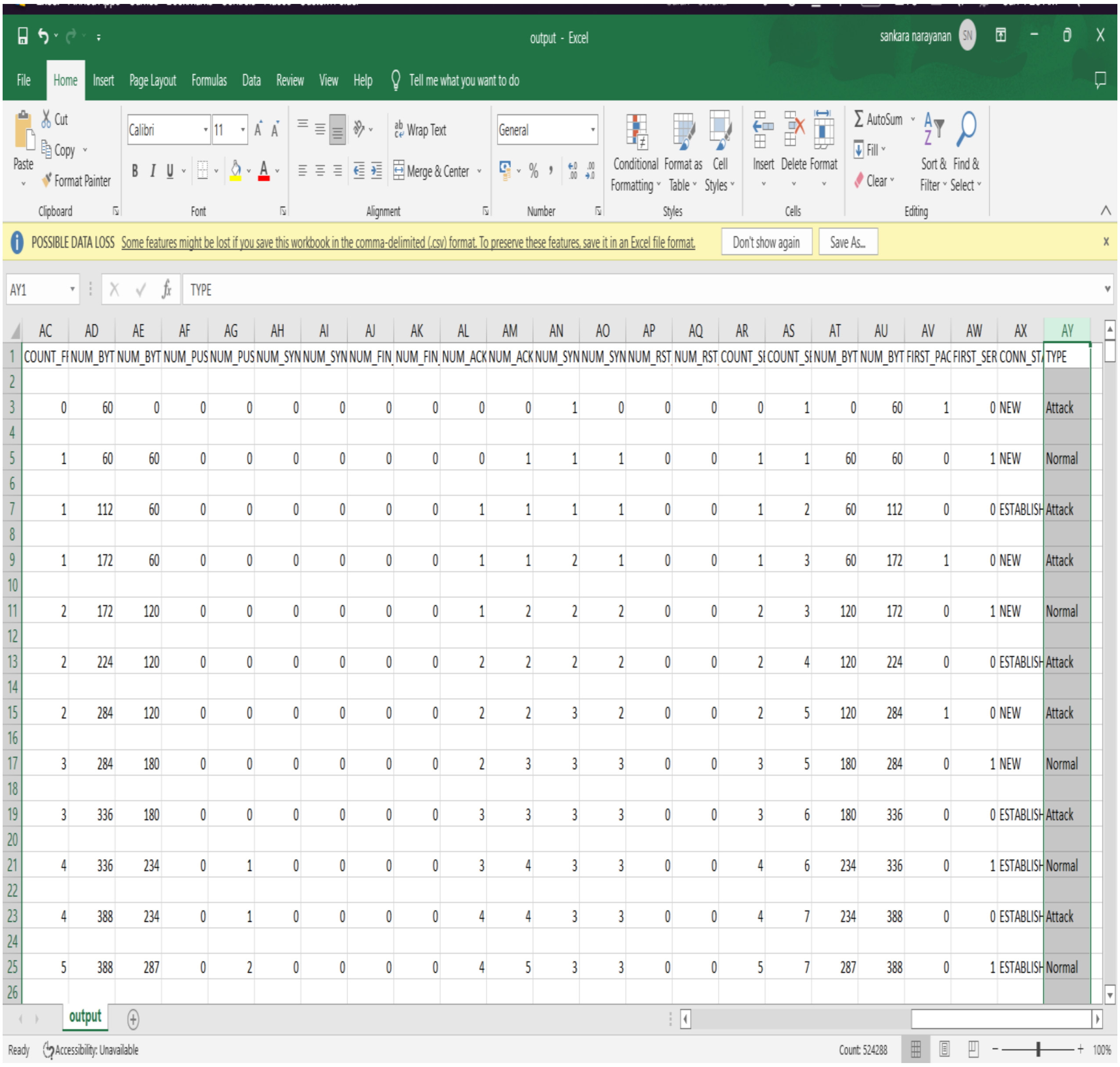


```
258 11.345214   10.0.3.2   10.0.3.3   SSHv2   1122 [Packet size limited during capture]
259 11.345616   10.0.3.3   10.0.3.2   TCP       66 55150 → 22 [ACK] Seq=1540 Ack=1097 Win=64128 Len=0 TSval=1699556572 TSecr=245277967
260 11.347550   10.0.3.3   10.0.3.2   SSHv2    114 [Packet size limited during capture]
261 11.347559   10.0.3.2   10.0.3.3   TCP       66 22 → 55150 [ACK] Seq=1097 Ack=1588 Win=64128 Len=0 TSval=245277970 TSecr=1699556574
262 11.352672   10.0.3.2   10.0.3.3   SSHv2    550 [Packet size limited during capture]
263 11.353046   10.0.3.3   10.0.3.2   TCP       66 55150 → 22 [ACK] Seq=1588 Ack=1581 Win=64128 Len=0 TSval=1699556580 TSecr=245277975
264 11.353791   10.0.3.3   10.0.3.2   TCP       66 55150 → 22 [FIN, ACK] Seq=1588 Ack=1581 Win=64128 Len=0 TSval=1699556580 TSecr=245277975
265 11.354973   10.0.3.2   10.0.3.3   TCP       66 22 → 55150 [FIN, ACK] Seq=1581 Ack=1589 Win=64128 Len=0 TSval=245277977 TSecr=1699556580
266 11.355361   10.0.3.3   10.0.3.2   TCP       66 55150 → 22 [ACK] Seq=1589 Ack=1582 Win=64128 Len=0 TSval=1699556582 TSecr=245277977
267 11.937175   10.0.2.3   10.0.2.2   TCP       74 58934 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=334568579 TSecr=0 WS=128
268 11.937211   10.0.2.2   10.0.2.3   TCP       74 22 → 58934 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3258919172 TSecr=334568579 WS=128
269 11.937613   10.0.2.3   10.0.2.2   TCP       66 58934 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=334568579 TSecr=3258919172
270 11.937958   10.0.2.3   10.0.2.2   SSHv2    101 Protocol (SSH-2.0-OpenSSH_9.0p1 Debian-1)[Packet size limited during capture]
271 11.937983   10.0.2.2   10.0.2.3   TCP       66 22 → 58934 [ACK] Seq=1 Ack=36 Win=65152 Len=0 TSval=3258919173 TSecr=334568580
272 11.947299   10.0.2.2   10.0.2.3   SSHv2    106 Protocol (SSH-2.0-OpenSSH_8.4p1 Debian-5)[Packet size limited during capture]
273 11.947718   10.0.2.3   10.0.2.2   TCP       66 58934 → 22 [ACK] Seq=36 Ack=41 Win=64256 Len=0 TSval=334568589 TSecr=3258919182
274 11.947988   10.0.2.3   10.0.2.2   SSHv2   1570 [Packet size limited during capture]
275 11.947994   10.0.2.2   10.0.2.3   TCP       66 22 → 58934 [ACK] Seq=41 Ack=1540 Win=64128 Len=0 TSval=3258919183 TSecr=334568590
276 11.948486   10.0.2.2   10.0.2.3   SSHv2   1122 [Packet size limited during capture]
277 11.948897   10.0.2.3   10.0.2.2   TCP       66 58934 → 22 [ACK] Seq=1540 Ack=1097 Win=64128 Len=0 TSval=334568591 TSecr=3258919184
278 11.950537   10.0.2.3   10.0.2.2   SSHv2    114 [Packet size limited during capture]
279 11.950544   10.0.2.2   10.0.2.3   TCP       66 22 → 58934 [ACK] Seq=1097 Ack=1588 Win=64128 Len=0 TSval=3258919186 TSecr=334568592
280 11.955232   10.0.2.2   10.0.2.3   SSHv2    550 [Packet size limited during capture]
281 11.955572   10.0.2.3   10.0.2.2   TCP       66 58934 → 22 [ACK] Seq=1588 Ack=1581 Win=64128 Len=0 TSval=334568597 TSecr=3258919190
282 11.956229   10.0.2.3   10.0.2.2   TCP       66 58934 → 22 [FIN, ACK] Seq=1588 Ack=1581 Win=64128 Len=0 TSval=334568598 TSecr=3258919190
283 11.957256   10.0.2.2   10.0.2.3   TCP       66 22 → 58934 [FIN, ACK] Seq=1581 Ack=1589 Win=64128 Len=0 TSval=3258919192 TSecr=334568598
284 11.957702   10.0.2.3   10.0.2.2   TCP       66 58934 → 22 [ACK] Seq=1589 Ack=1582 Win=64128 Len=0 TSval=334568599 TSecr=3258919192
285 13.057860   10.0.1.2   10.0.1.1   BOOTP    326 Boot Request from 08:00:27:5b:ad:7e (PcsCompu_5b:ad:7e)[Packet size limited during capture]
286 13.058661   10.0.3.2   10.0.3.1   BOOTP    326 Boot Request from 08:00:27:e7:aa:d1 (PcsCompu_e7:aa:d1)[Packet size limited during capture]
287 13.065470   10.0.1.1   10.0.1.2   BOOTP    590 Boot Reply[Packet size limited during capture]
288 13.065995   10.0.3.1   10.0.3.2   BOOTP    590 Boot Reply[Packet size limited during capture]
289 13.310983   10.0.2.3   10.0.2.2   TCP       74 58942 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=334569953 TSecr=0 WS=128
290 13.311070   10.0.2.2   10.0.2.3   TCP       74 22 → 58942 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3258920546 TSecr=334569953 WS=128
291 13.311642   10.0.2.3   10.0.2.2   TCP       66 58942 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=334569953 TSecr=3258920546
292 13.312376   10.0.2.3   10.0.2.2   SSHv2    101 Protocol (SSH-2.0-OpenSSH_9.0p1 Debian-1)[Packet size limited during capture]
293 13.312424   10.0.2.2   10.0.2.3   TCP       66 22 → 58942 [ACK] Seq=1 Ack=36 Win=65152 Len=0 TSval=3258920548 TSecr=334569954
294 13.332798   10.0.2.2   10.0.2.3   SSHv2    106 Protocol (SSH-2.0-OpenSSH_8.4p1 Debian-5)[Packet size limited during capture]
295 13.333127   10.0.2.3   10.0.2.2   TCP       66 58942 → 22 [ACK] Seq=36 Ack=41 Win=64256 Len=0 TSval=334569975 TSecr=3258920568
296 13.333482   10.0.2.3   10.0.2.2   SSHv2   1570 [Packet size limited during capture]
297 13.333492   10.0.2.2   10.0.2.3   TCP       66 22 → 58942 [ACK] Seq=41 Ack=1540 Win=64128 Len=0 TSval=3258920569 TSecr=334569975
298 13.334418   10.0.2.2   10.0.2.3   SSHv2   1122 [Packet size limited during capture]
299 13.334678   10.0.2.3   10.0.2.2   TCP       66 58942 → 22 [ACK] Seq=1540 Ack=1097 Win=64128 Len=0 TSval=334569977 TSecr=3258920570
300 13.337790   10.0.2.3   10.0.2.2   SSHv2    114 [Packet size limited during capture]
301 13.337800   10.0.2.2   10.0.2.3   TCP       66 22 → 58942 [ACK] Seq=1097 Ack=1588 Win=64128 Len=0 TSval=3258920573 TSecr=334569980
302 13.346087   10.0.2.2   10.0.2.3   SSHv2    550 [Packet size limited during capture]
303 13.346462   10.0.2.3   10.0.2.2   TCP       66 58942 → 22 [ACK] Seq=1588 Ack=1581 Win=64128 Len=0 TSval=334569988 TSecr=3258920581
304 13.347703   10.0.2.3   10.0.2.2   TCP       66 58942 → 22 [FIN, ACK] Seq=1588 Ack=1581 Win=64128 Len=0 TSval=334569989 TSecr=3258920581
305 13.349317   10.0.2.2   10.0.2.3   TCP       66 22 → 58942 [FIN, ACK] Seq=1581 Ack=1589 Win=64128 Len=0 TSval=3258920584 TSecr=334569989
306 13.349563   10.0.2.3   10.0.2.2   TCP       66 58942 → 22 [ACK] Seq=1589 Ack=1582 Win=64128 Len=0 TSval=334569991 TSecr=3258920584
307 14.170187   10.0.3.3   10.0.3.2   SNMP     106 [Packet size limited during capture]
```



```c
/* Frame (96 bytes) */
static const unsigned char pkt10678[96] = {
0x08, 0x00, 0x27, 0x61, 0x89, 0x04, 0x08, 0x00, /* ..'a.... */
0x27, 0x5b, 0xad, 0x7e, 0x08, 0x00, 0x45, 0x08, /* '[.~..E. */
0xfe, 0xbc, 0x7d, 0xbe, 0x40, 0x00, 0x40, 0x06, /* ..}.@.@. */
0xa8, 0x70, 0x0a, 0x00, 0x01, 0x02, 0x0a, 0x00, /* .p...... */
0x01, 0x03, 0x00, 0x16, 0xbe, 0x8a, 0x22, 0x4c, /* ......"L */
0xc1, 0x2a, 0x41, 0x65, 0x19, 0xe1, 0x80, 0x18, /* .*Ae.... */
0x01, 0xf5, 0x14, 0xb4, 0x00, 0x00, 0x01, 0x01, /* ........ */
0x08, 0x0a, 0x34, 0xca, 0x4e, 0x0a, 0xca, 0xc2, /* ..4.N... */
0x0f, 0xa7, 0xba, 0x7f, 0x1c, 0x0a, 0xa2, 0xe7, /* ........ */
0xcb, 0xd4, 0x79, 0x50, 0xd9, 0x7e, 0xdb, 0x6c, /* ..yP.~.l */
0x37, 0x45, 0x0f, 0x21, 0xa2, 0xc4, 0x41, 0xeb, /* 7E.!..A. */
0x81, 0x1c, 0x31, 0x90, 0x80, 0x74, 0x32, 0x14  /* ..1..t2. */
};

/* Frame (96 bytes) */
static const unsigned char pkt10679[96] = {
0x08, 0x00, 0x27, 0x61, 0x89, 0x04, 0x08, 0x00, /* ..'a.... */
0x27, 0x5b, 0xad, 0x7e, 0x08, 0x00, 0x45, 0x08, /* '[.~..E. */
0xf9, 0x14, 0x7d, 0xeb, 0x40, 0x00, 0x40, 0x06, /* ..}.@.@. */
0xad, 0xeb, 0x0a, 0x00, 0x01, 0x02, 0x0a, 0x00, /* ........ */
0x01, 0x03, 0x00, 0x16, 0xbe, 0x8a, 0x22, 0x4d, /* ......"M */
0xbf, 0xb2, 0x41, 0x65, 0x19, 0xe1, 0x80, 0x18, /* ..Ae.... */
0x01, 0xf5, 0x0f, 0x0c, 0x00, 0x00, 0x01, 0x01, /* ........ */
0x08, 0x0a, 0x34, 0xca, 0x4e, 0x0a, 0xca, 0xc2, /* ..4.N... */
0x0f, 0xa7, 0x92, 0x14, 0xd9, 0xe9, 0x07, 0x33, /* .......3 */
0xa9, 0xcb, 0x3a, 0x55, 0xc8, 0xc1, 0xde, 0xc5, /* ..:U.... */
0xa1, 0xb1, 0xdf, 0xd8, 0x75, 0x3d, 0x4b, 0xdb, /* ....u=K. */
0x8d, 0x76, 0xd4, 0x59, 0x79, 0x37, 0x64, 0x01  /* .v.Yy7d. */
};

/* Frame (66 bytes) */
static const unsigned char pkt10680[66] = {
0x08, 0x00, 0x27, 0x5b, 0xad, 0x7e, 0x08, 0x00, /* ..'[.~.. */
0x27, 0x61, 0x89, 0x04, 0x08, 0x00, 0x45, 0x08, /* 'a....E. */
0x00, 0x34, 0x6a, 0xfd, 0x40, 0x00, 0x40, 0x06, /* .4j.@.@. */
0xb9, 0xba, 0x0a, 0x00, 0x01, 0x03, 0x0a, 0x00, /* ........ */
0x01, 0x02, 0xbe, 0x8a, 0x00, 0x16, 0x41, 0x65, /* ......Ae */
0x19, 0xe1, 0x22, 0x4e, 0xb8, 0x92, 0x80, 0x10, /* .."N.... */
0x3f, 0xa9, 0xce, 0xe5, 0x00, 0x00, 0x01, 0x01, /* ?....... */
0x08, 0x0a, 0xca, 0xc2, 0x0f, 0xcb, 0x34, 0xca, /* ......4. */
```

Fig 6) Output.csv:



POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

AY1 TYPE

	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY
1	COUNT_FI	NUM_BYT	NUM_BYT	NUM_PUS	NUM_PUS	NUM_SYN	NUM_SYN	NUM_FIN_	NUM_FIN_	NUM_ACK	NUM_ACK	NUM_SYN	NUM_SYN	NUM_RST	NUM_RST	COUNT_SI	COUNT_SI	NUM_BYT	NUM_BYT	FIRST_PAC	FIRST_SER	CONN_STA	TYPE
2																							
3	0	60	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	60	1	0	NEW	Attack
4																							
5	1	60	60	0	0	0	0	0	0	0	1	1	1	0	0	1	1	60	60	0	1	NEW	Normal
6																							
7	1	112	60	0	0	0	0	0	0	1	1	1	1	0	0	1	2	60	112	0	0	ESTABLISH	Attack
8																							
9	1	172	60	0	0	0	0	0	0	1	1	2	1	0	0	1	3	60	172	1	0	NEW	Attack
10																							
11	2	172	120	0	0	0	0	0	0	1	2	2	2	0	0	2	3	120	172	0	1	NEW	Normal
12																							
13	2	224	120	0	0	0	0	0	0	2	2	2	2	0	0	2	4	120	224	0	0	ESTABLISH	Attack
14																							
15	2	284	120	0	0	0	0	0	0	2	2	3	2	0	0	2	5	120	284	1	0	NEW	Attack
16																							
17	3	284	180	0	0	0	0	0	0	2	3	3	3	0	0	3	5	180	284	0	1	NEW	Normal
18																							
19	3	336	180	0	0	0	0	0	0	3	3	3	3	0	0	3	6	180	336	0	0	ESTABLISH	Attack
20																							
21	4	336	234	0	1	0	0	0	0	3	4	3	3	0	0	4	6	234	336	0	1	ESTABLISH	Normal
22																							
23	4	388	234	0	1	0	0	0	0	4	4	3	3	0	0	4	7	234	388	0	0	ESTABLISH	Attack
24																							
25	5	388	287	0	2	0	0	0	0	4	5	3	3	0	0	5	7	287	388	0	1	ESTABLISH	Normal
26																							

Fig 7) Correlation Heatmap of all 51 features:

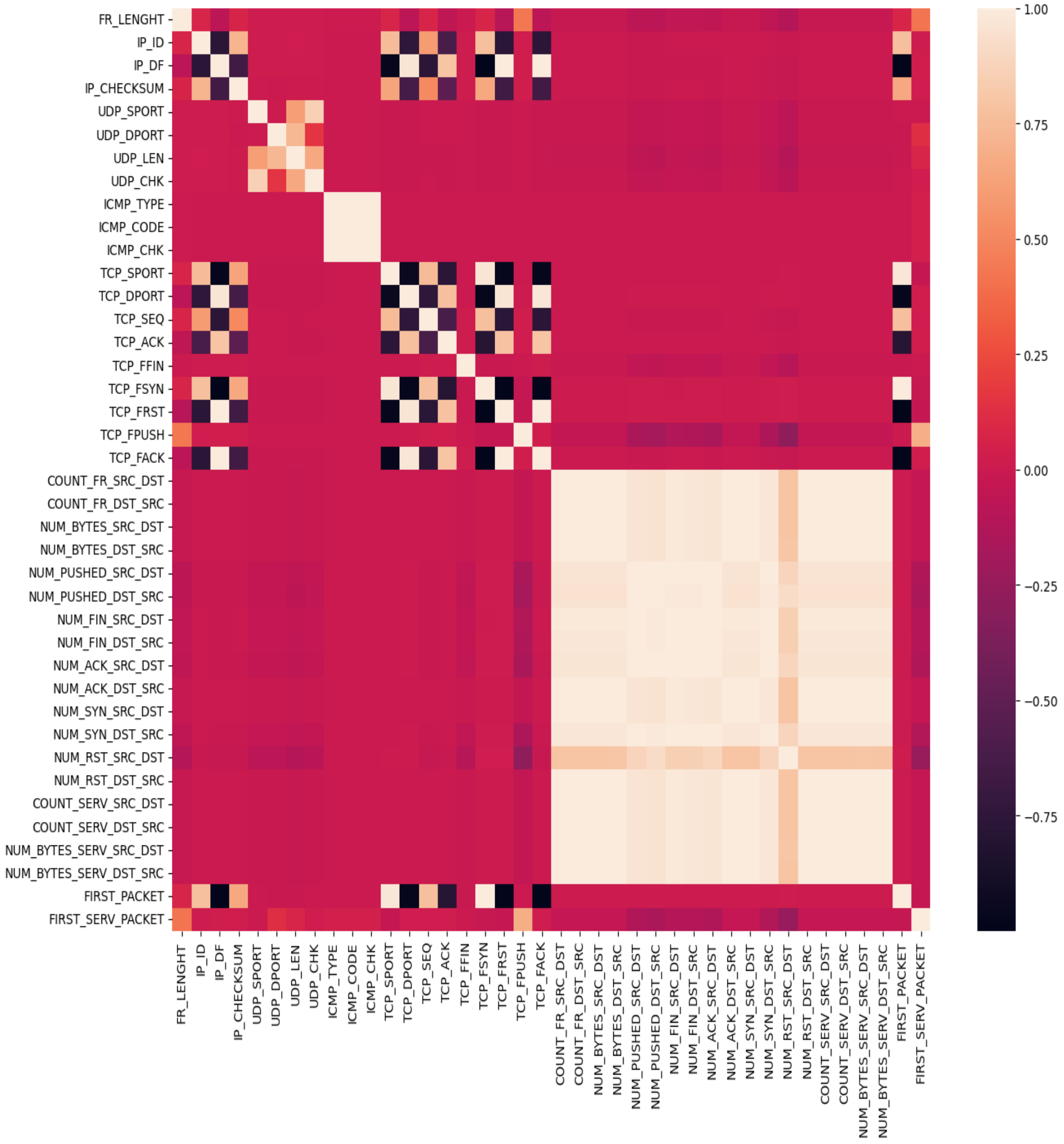
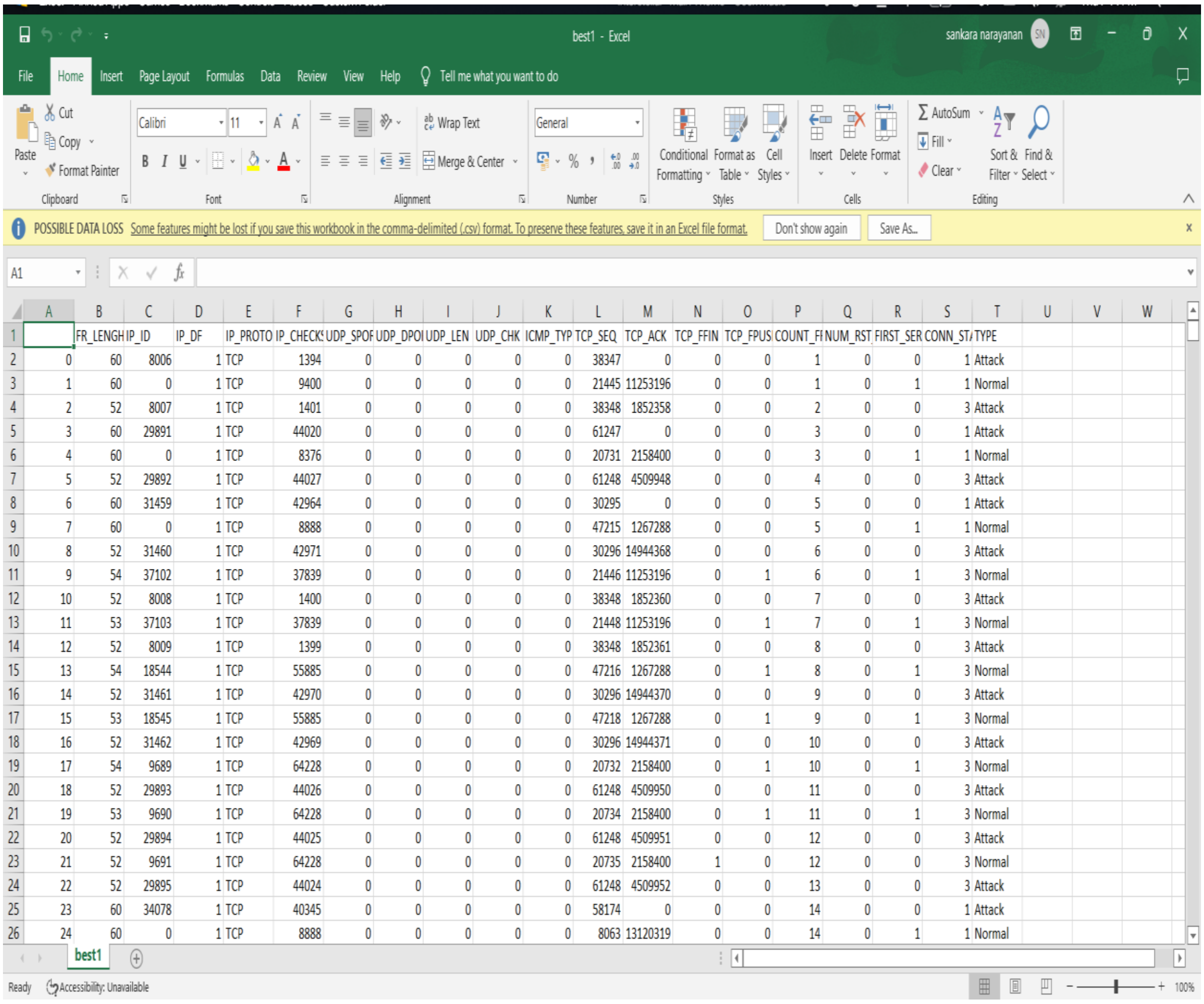


Fig 8) Best1.csv after preprocessing:



POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1		FR_LENGH	IP_ID	IP_DF	IP_PROTO	IP_CHECKS	UDP_SPOF	UDP_DPOI	UDP_LEN	UDP_CHK	ICMP_TYP	TCP_SEQ	TCP_ACK	TCP_FFIN	TCP_FPUS	COUNT_FI	NUM_RST	FIRST_SER	CONN_STA	TYPE
2	0	60	8006	1	TCP	1394	0	0	0	0	0	38347	0	0	0	1	0	0	1	Attack
3	1	60	0	1	TCP	9400	0	0	0	0	0	21445	11253196	0	0	1	0	1	1	Normal
4	2	52	8007	1	TCP	1401	0	0	0	0	0	38348	1852358	0	0	2	0	0	3	Attack
5	3	60	29891	1	TCP	44020	0	0	0	0	0	61247	0	0	0	3	0	0	1	Attack
6	4	60	0	1	TCP	8376	0	0	0	0	0	20731	2158400	0	0	3	0	1	1	Normal
7	5	52	29892	1	TCP	44027	0	0	0	0	0	61248	4509948	0	0	4	0	0	3	Attack
8	6	60	31459	1	TCP	42964	0	0	0	0	0	30295	0	0	0	5	0	0	1	Attack
9	7	60	0	1	TCP	8888	0	0	0	0	0	47215	1267288	0	0	5	0	1	1	Normal
10	8	52	31460	1	TCP	42971	0	0	0	0	0	30296	14944368	0	0	6	0	0	3	Attack
11	9	54	37102	1	TCP	37839	0	0	0	0	0	21446	11253196	0	1	6	0	1	3	Normal
12	10	52	8008	1	TCP	1400	0	0	0	0	0	38348	1852360	0	0	7	0	0	3	Attack
13	11	53	37103	1	TCP	37839	0	0	0	0	0	21448	11253196	0	1	7	0	1	3	Normal
14	12	52	8009	1	TCP	1399	0	0	0	0	0	38348	1852361	0	0	8	0	0	3	Attack
15	13	54	18544	1	TCP	55885	0	0	0	0	0	47216	1267288	0	1	8	0	1	3	Normal
16	14	52	31461	1	TCP	42970	0	0	0	0	0	30296	14944370	0	0	9	0	0	3	Attack
17	15	53	18545	1	TCP	55885	0	0	0	0	0	47218	1267288	0	1	9	0	1	3	Normal
18	16	52	31462	1	TCP	42969	0	0	0	0	0	30296	14944371	0	0	10	0	0	3	Attack
19	17	54	9689	1	TCP	64228	0	0	0	0	0	20732	2158400	0	1	10	0	1	3	Normal
20	18	52	29893	1	TCP	44026	0	0	0	0	0	61248	4509950	0	0	11	0	0	3	Attack
21	19	53	9690	1	TCP	64228	0	0	0	0	0	20734	2158400	0	1	11	0	1	3	Normal
22	20	52	29894	1	TCP	44025	0	0	0	0	0	61248	4509951	0	0	12	0	0	3	Attack
23	21	52	9691	1	TCP	64228	0	0	0	0	0	20735	2158400	1	0	12	0	0	3	Normal
24	22	52	29895	1	TCP	44024	0	0	0	0	0	61248	4509952	0	0	13	0	0	3	Attack
25	23	60	34078	1	TCP	40345	0	0	0	0	0	58174	0	0	0	14	0	0	1	Attack
26	24	60	0	1	TCP	8888	0	0	0	0	0	8063	13120319	0	0	14	0	1	1	Normal

Fig 9) Confusion Matrix Heatmap of Naïve Bayes:

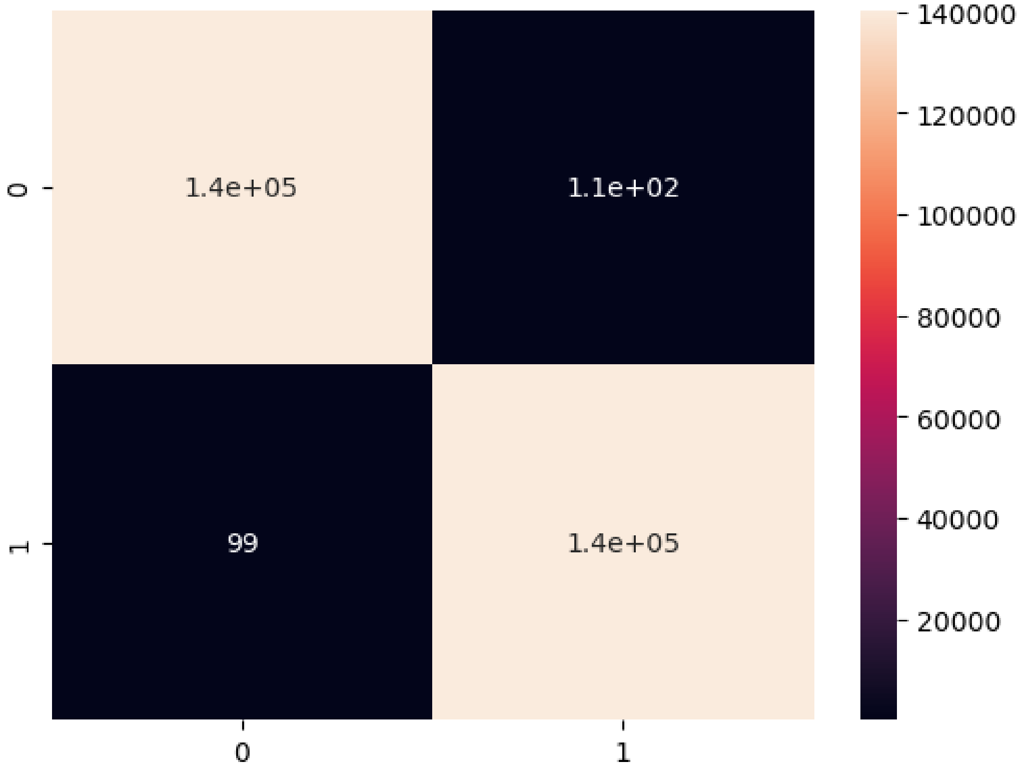


Fig 10) Confusion Matrix Heatmap of DecisionTree:

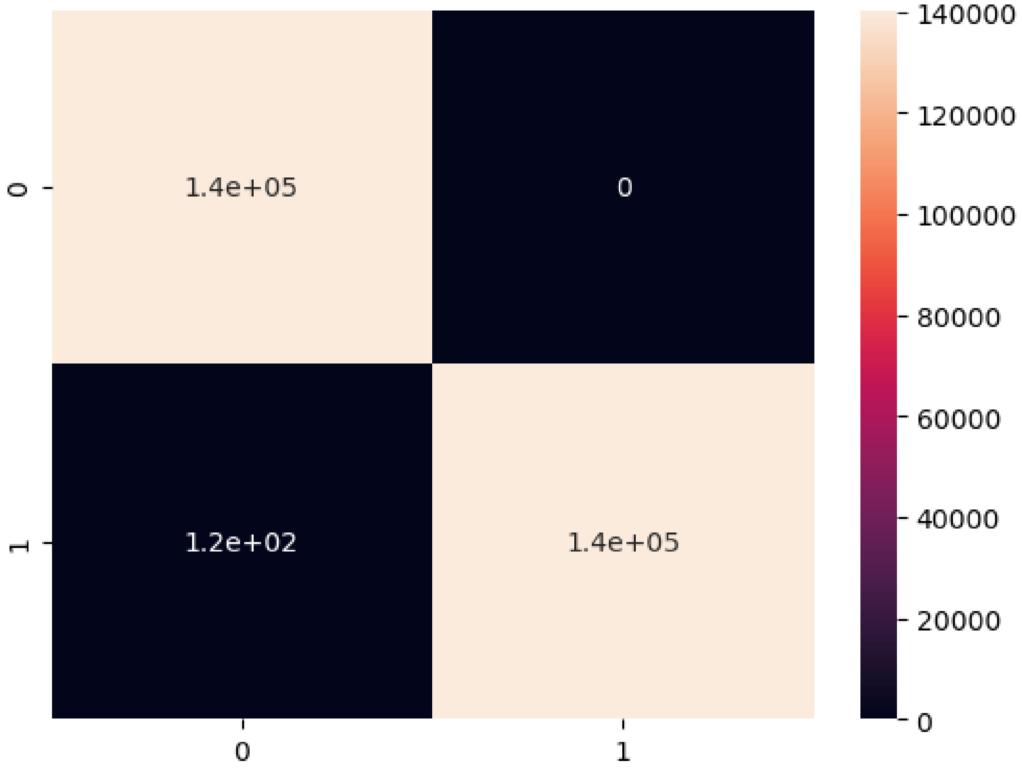


Fig 11) Confusion Matrix Heatmap of Random Classifier:

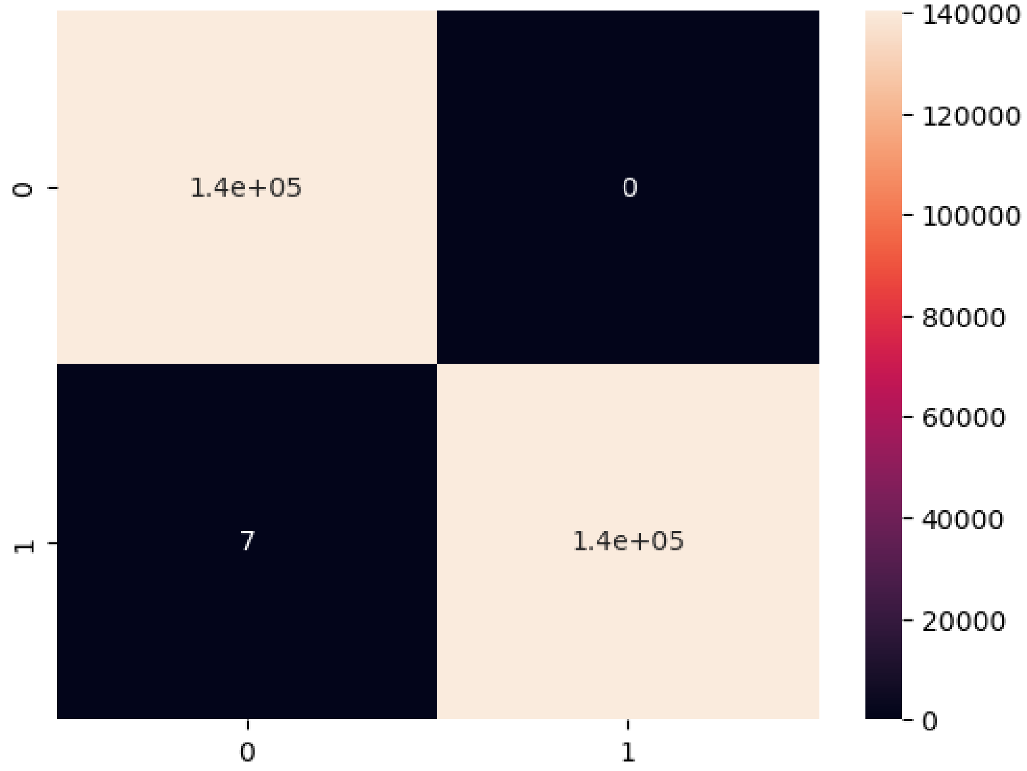


Fig 12) Confusion Matrix Heatmap of SVC:

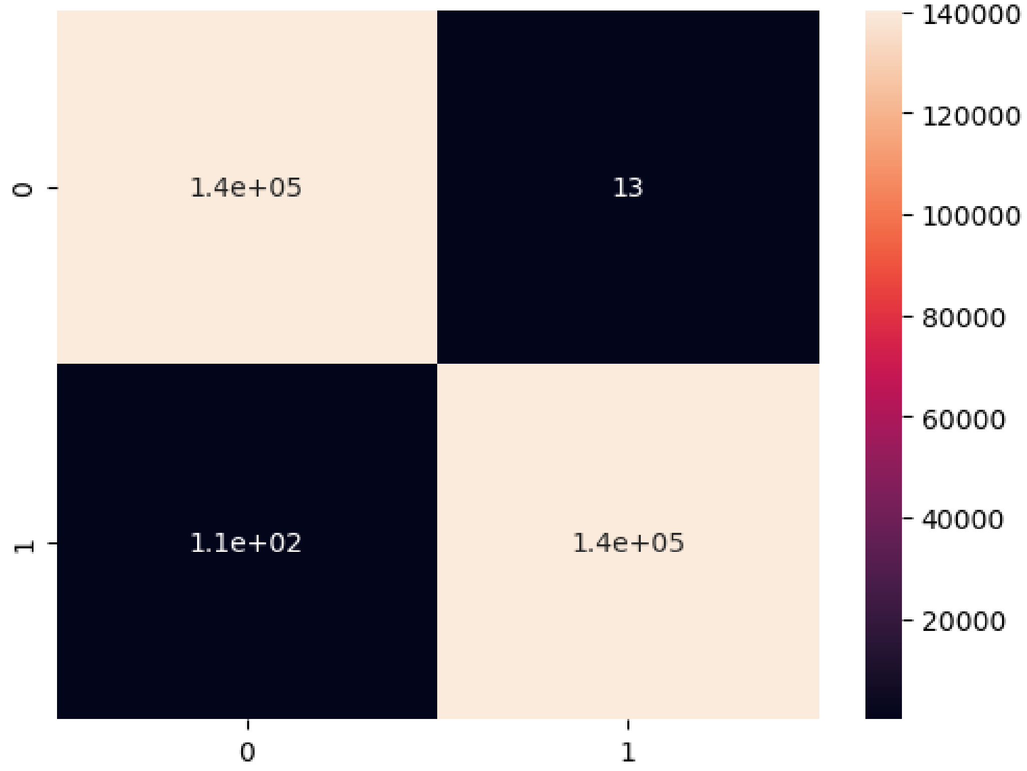
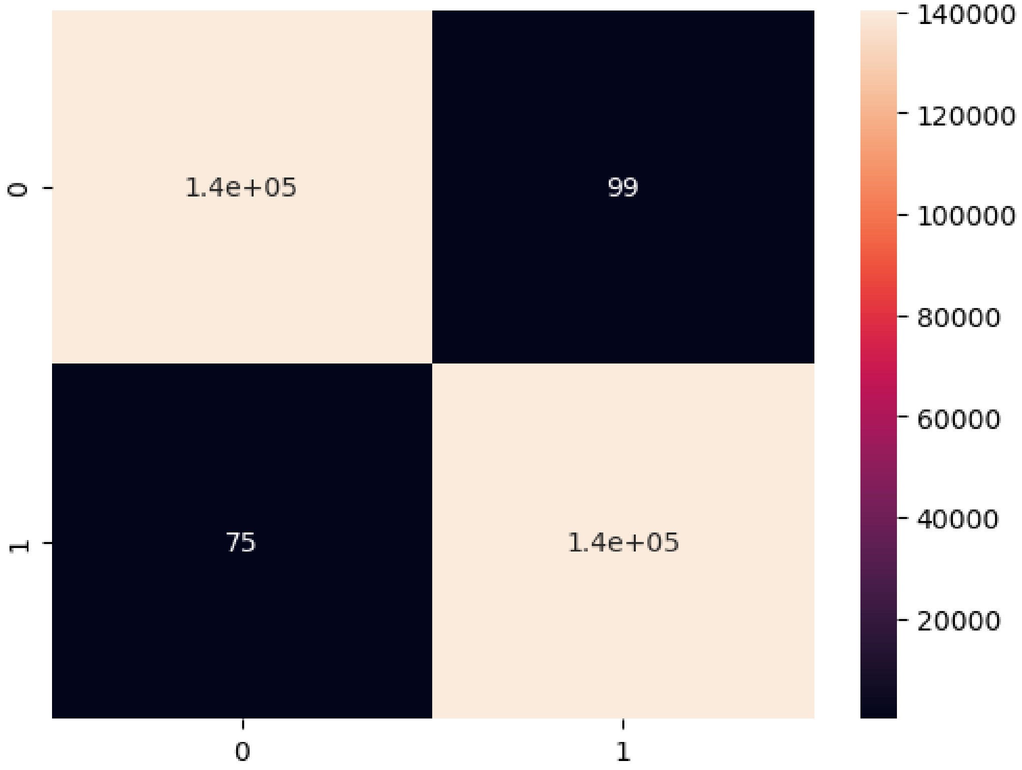


Fig 13) Confusion Matrix Heatmap of Logistic Regression:



5.

Results

50 Features + Target:

1. IP_TYPE
2. IP_LEN
3. FR_LENGHT
4. IP_ID
5. IP_RESERVED
6. IP_DF
7. IP_MF
8. IP_OFFSET
9. IP_PROTO
10. IP_CHECKSUM
11. UDP_SPORT
12. UDP_DPORT
13. UDP_LEN
14. UDP_CHK
15. ICMP_TYPE
16. ICMP_CODE
17. ICMP_CHK
18. TCP_SPORT
19. TCP_DPORT
20. TCP_SEQ
21. TCP_ACK
22. TCP_FFIN
23. TCP_FSYN
24. TCP_FRST
25. TCP_FPUSH
26. TCP_FACK
27. TCP_FURG
28. COUNT_FR_SRC_DST
29. COUNT_FR_DST_SRC
30. NUM_BYTES_SRC_DST
31. NUM_BYTES_DST_SRC
32. NUM_PUSHED_SRC_DST
33. NUM_PUSHED_DST_SRC
34. NUM_SYN_FIN_SRC_DST
35. NUM_SYN_FIN_DST_SRC
36. NUM_FIN_SRC_DST
37. NUM_FIN_DST_SRC
38. NUM_ACK_SRC_DST
39. NUM_ACK_DST_SRC
40. NUM_SYN_SRC_DST
41. NUM_SYN_DST_SRC
42. NUM_RST_SRC_DST
43. NUM_RST_DST_SRC
44. COUNT_SERV_SRC_DST
45. COUNT_SERV_DST_SRC

46. NUM_BYTES_SERV_SRC_DST
47. NUM_BYTES_SERV_DST_SRC
48. FIRST_PACKET
49. FIRST_SERV_PACKET
50. CONN_STATUS
51. TYPE

SRC_DST and DST_SRC indicate the direction of the transmission which will be sent and received respectively. The target variable can take on two values: Attack: if the packet comes from the client; Normal: if the packet comes from the server;

Columns with correlation more than 0.95:

['ICMP_CODE', 'ICMP_CHK', 'TCP_SPORT', 'TCP_DPORT', 'TCP_FSYN', 'TCP_FRST', 'TCP_FACK', 'COUNT_FR_DST_SRC', 'NUM_BYTES_SRC_DST', 'NUM_BYTES_DST_SRC', 'NUM_PUSHED_SRC_DST', 'NUM_PUSHED_DST_SRC', 'NUM_FIN_SRC_DST', 'NUM_FIN_DST_SRC', 'NUM_ACK_SRC_DST', 'NUM_ACK_DST_SRC', 'NUM_SYN_SRC_DST', 'NUM_SYN_DST_SRC', 'NUM_RST_DST_SRC', 'COUNT_SERV_SRC_DST', 'COUNT_SERV_DST_SRC', 'NUM_BYTES_SERV_SRC_DST', 'NUM_BYTES_SERV_DST_SRC', 'FIRST_PACKET']

are dropped.

After training,

Final output:

Ml technique	Training time in seconds	Testing time in seconds	Train accuracy out of 1	Test accuracy out of 1
Naïve bayes	0.8466770648956299	0.24017620086669922	0.9993334350721907	0.9992449873570997
Decision tree classifier	1.5850307941436768	0.04646468162536621	0.9995807657427725	0.9995655115922931
Random forest classifier	9.116121053695679	1.2806715965270996	1.0	0.9999750703372627
SVC	25.29870915412903	23.65485382080078	0.9996070775162388	0.9995619502119021
Logistic regression	2.761035203933716	0.029804706573486328	0.9994000915649717	0.9993803198119591

Random Forest Classifier yields the highest accuracy of 0.99997 on test data

6. Conclusion and Future Works

In future, we will conduct an extensive study of ML algorithms to provide better solution for the IDS. New models can be trained by using more diverse data by simulating more protocols and recording packets for longer duration of time.

My model only predicts whether a intrusion is happened or not , it can be further extended to predict different types of intrusions. Keeping firewall as a primary defense , intrusion detection system can be deployed as a secondary protection.

7. References

1. A. Shiravi, H. Shiravi, M. Tavallaee, and A. a. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," Comput. Secur., vol. 31, no. 3, pp. 357–374, 2012.

2. CAIDA. The cooperative association for internet data analysis [online] available: http://www.caida.org/. Accessed Apr./2018.

3. C. Gates and C. Taylor, "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," Proc. 2006 Work. New Secur. Paradig., pp. 21–29, 2007.

4. C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, "Intrusion detection by machine learning: A review," Expert Syst. Appl., vol. 36, no. 10, pp. 11994–12000, 2009.

5. H. Ringberg, M. Roughan, and J. Rexford, "The need for simulation in evaluating anomaly detectors," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 1, p. 55, 2008.

6. G. Stein, B. Chen, A. S. Wu, and K. a. Hua, “Decision tree classifier for network intrusion detection with GA-based feature selection,” Proc. 43rd Annu. southeast Reg. Conf. - ACM-SE 43, vol. 2, p. 136, 2005.

7. I. Corona, G. Giacinto, and F. Roli, “Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues,” Inf. Sci. (Ny)., vol. 239, pp. 201–225, Aug. 2013.

8. E. K. Viegas, A. O. Santin, and L. S. Oliveira, “Toward a reliable anomaly-based intrusion detection in real-world environments,” Comput. Networks, vol. 127, pp. 200-216, 2017.

9. E. Viegas, A. Santin, A. França, R. Jasinski, V. Pedroni, and L. Oliveira, “Towards an Energy-Efficient Anomaly-Based Intrusion Detection Engine for Embedded Systems,” IEEE Trans. On Computers, vol. 66, pp. 163–177, 2017.

10. K. Kendall "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", 1999

11. Labs. Kaspersky. 2014. Security Bulletin 2014. 2014.

12. Lawrence Berkeley Nationatiol Laboratory, The internet traffic archive [online] available: http://ita.ee.lbl.gov/index.html. Accessed April./2018.

13. E. Viegas, A. Santin, V. Abreu, and L. S. Oliveira, “Stream learning and anomaly-based intrusion detection in the adversarial settings,” in Proceedings - IEEE Symposium on Computers and Communications, 2017.