Ye Htet Kyaw

27. November. 2022

Scripting for Cybersecurity and Network

Report For Application

#### Report

# 1. Briefly explain the overall structure of your code and your rationale for this. Mention any variations you considered.

In this python scripting module, there are three files. They are

1. pcap\_analyzer.py
2. packet\_information.py
3. graph\_visualizer.py

## pcap\_analyzer.py

It is the main files of the whole scripting module. If you want run the program you need to run this file. This script can access the functions from the other python files. By Importing them.

## packet\_information.py

This file has no main function. If you run this file, you will not see any output because it has no main function and it is designed to call from the other files. There are five functions in this file. They are

1. prase\_packets
   * this function returns the python object of ‘.pcap’ file inputed.
2. show\_summerize\_table
   * this function print first timestamp, last timestamp, mean packets length and number of packets of each protocol type.
3. packet\_ip\_info
   * It show the ip\_address pair and number of their packet and return this data
4. email\_image
   * It find the image download and email sent and received from the packets
5. dt\_from\_ts
   * It returns the UTC format from the input timestamp. (It’s not in coursework. I did it because I don’t want to duplicate the codes)
6. is\_other
   * It returns the boolean value whether the packet is not TCP, IGMP, UDP.

## graph\_visualizer.py

This file doesn’t have main function like pcap\_analyzer.py. It needs to import from the other files to call the function in this file. There are 2 functions in the graph\_visualizer.py. They are

1. graph\_visualize
   * it show the network graph of IP address pairs and their traffic count, numbers of nodes, numbers of edges and which nodes are weakly or strongly connected each other.
2. timestamp\_linechart
   * it shows the line chart of the IP address pairs’ network traffic against the time.

# b. Briefly explain your overall approach to exception handling in this application, and discuss one example of this in detail.

In this application, tried to covered all of the error exception. Like running application without inputting the ‘.pcap’ files, using the functions without arguments, unpacking the packet in the wrong way.

For example, email\_image function of packet\_information.py file can occur the error exception when trying to request the data of TCP from the wrong packet. So, application need to handle this error by catching with the related class. Now program is catching the error by using Except keyword and the class is the ‘dpkt.UnpackError’. When this error occur again, It can handle the program to write the error to the text file of error log under the output folder. So, the application cannot be stop by the error and user can also check the error message after the program run.

Furthermore, application covered user using the functions without arguments from the main function by catching ‘TypeError’.

# c. Adaptively handling any type of packet for the summary, not just the 3 types that occur in the test pcap

After creating the prase\_packets functions, program can access the packets from the “.pcap” file. For the types of the packets, it create the constant list of packet type. And then application loop the buffer from the data of prase\_packets function and changing the buffer to the ethernet data. And it filter the IP from the ethernet data.

And then program loop the constant list of packet type and checking whether the data from the IP is the class of TCP or UDP or IGMP. If one of them is true, it put the value to the packet\_info dictionary variable with the related keys of each type.

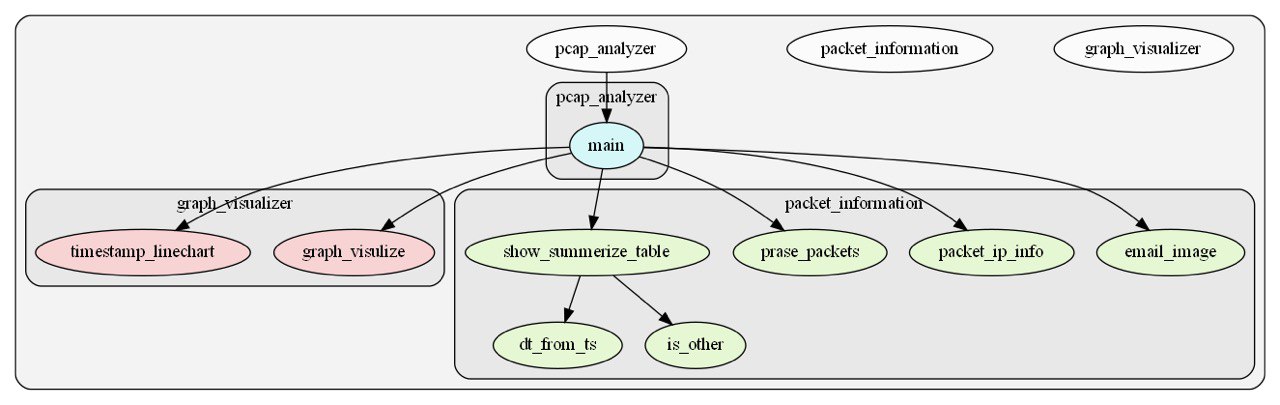
For the other packets type, application has extra function call “is\_other” that is checking whether the packet is not TCP or UDP or IGMP. If it is not, it return TRUE else, it return FALSE. In this function, application check the packet is IGMP if it is not, application check the packet is UDP if it is not and it also check for TCP and if it is not three of it, program return TRUE. Using that function, it check the packet is whether others (exclude TCP,UDP,IGMP) packet or not. That’s how this application search the other packets from the pcap file.

## d. Counting and sorting the packets flowing between different pairs of IP addresses.

Counting and sorting the packets flowing between different pairs of IP addresses is the one of the functions of this application. It loop the buff and timestamp data of the data from the prase \_packet function. Changing buffer to the ethernet data and get source and destination data from the ethernet data frame. And then the list of ethernet data will be store as value and IP address pair will be store as key in the dictionary.

After the end of looping, program create a new dictionary and putting data to this how many network traffics are going through between the IP address pair by using the len() built-in function to the list of ethernet data list. Then sorting this dictionary with the lambda. This is how this application count and sort the packets flowing between different pairs of IP address.

# 2. Dependency Diagram of Application



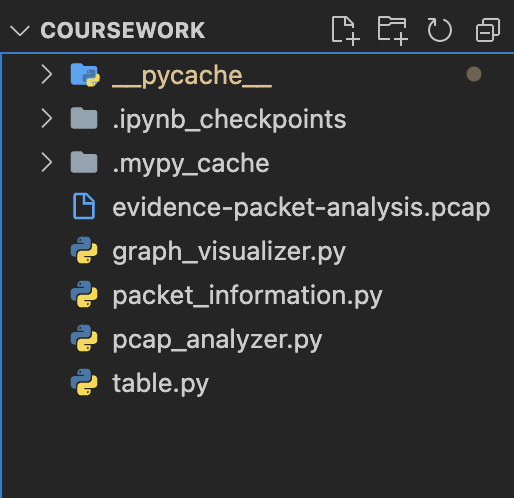
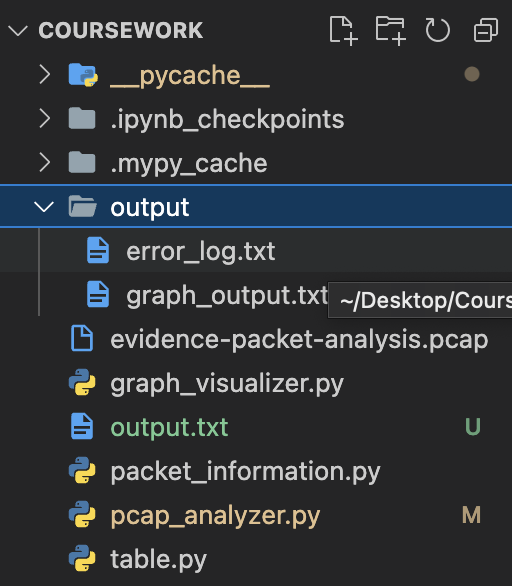
# Table Listing functions and module

|  |
| --- |
| List of the Modules in this application |
| pcap\_analyzer.py |
| packet\_ information.py |
| graph\_visualizer.py |

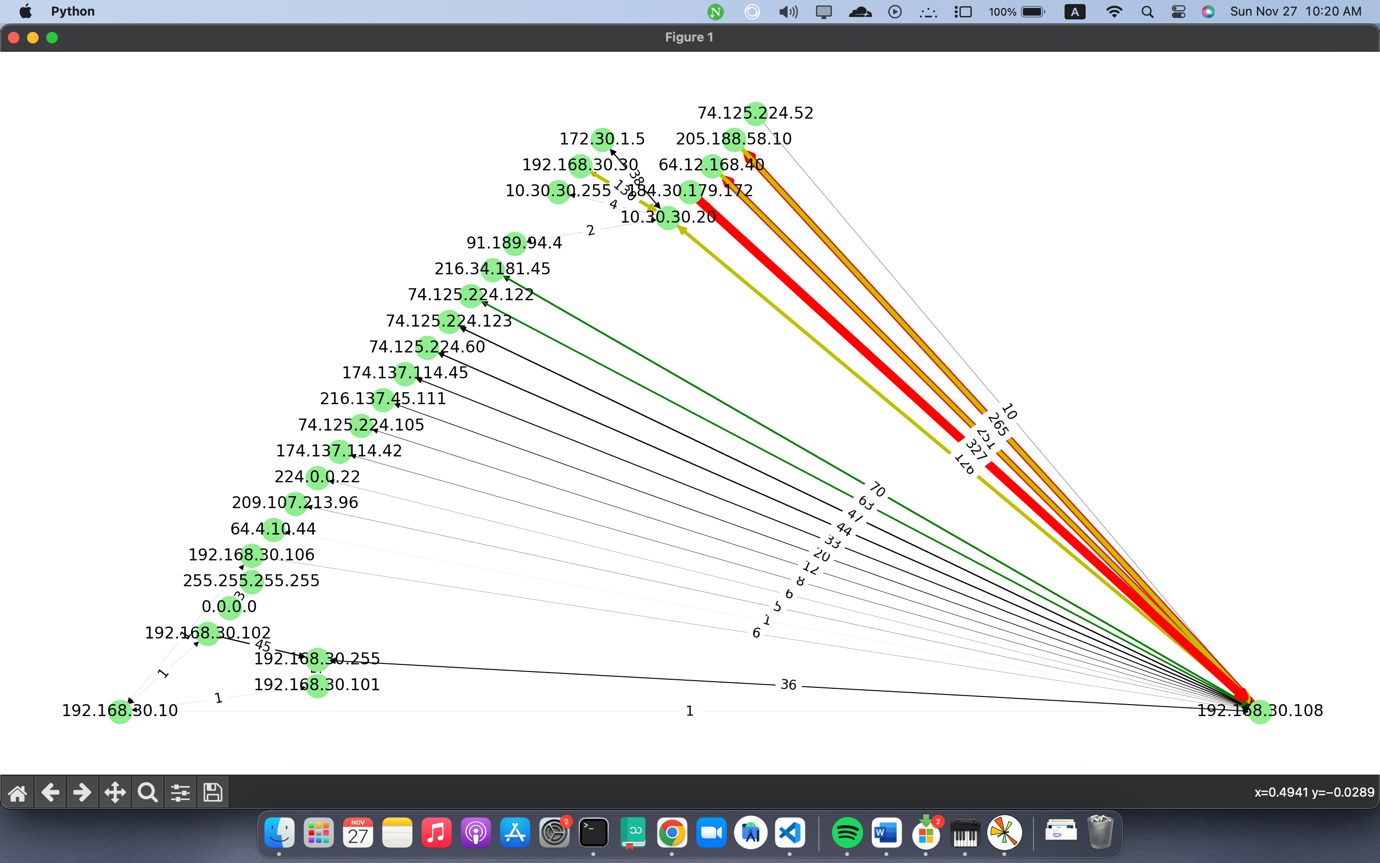
|  |  |
| --- | --- |
| List of the Functions in packet\_infomation.py | Description |
| prase\_packet | Changing pcap file to python data |
| Show\_summerize\_table | Showing table of packet numbers total buffer and man packets length of each type first time stamp, last timestamp |
| Packet\_ip\_info | Show the which ip address pair most going connected each other. |
| Email\_image | getting email and image from the packets. |
| Dt\_from\_ts | Changing to the date time to the inputed timestamp data |
| Is\_other | Checking the packet which are not TCP, UDP and IGMP |

|  |  |
| --- | --- |
| List of the Functions in graph\_visualizer.py | Description |
| Graph\_visualize | Showing the network graph to the user between IP address pair. |
| Timestamp\_linechart | Showing the network traffic as the line chart by time intervals to know which interval has most traffic. |

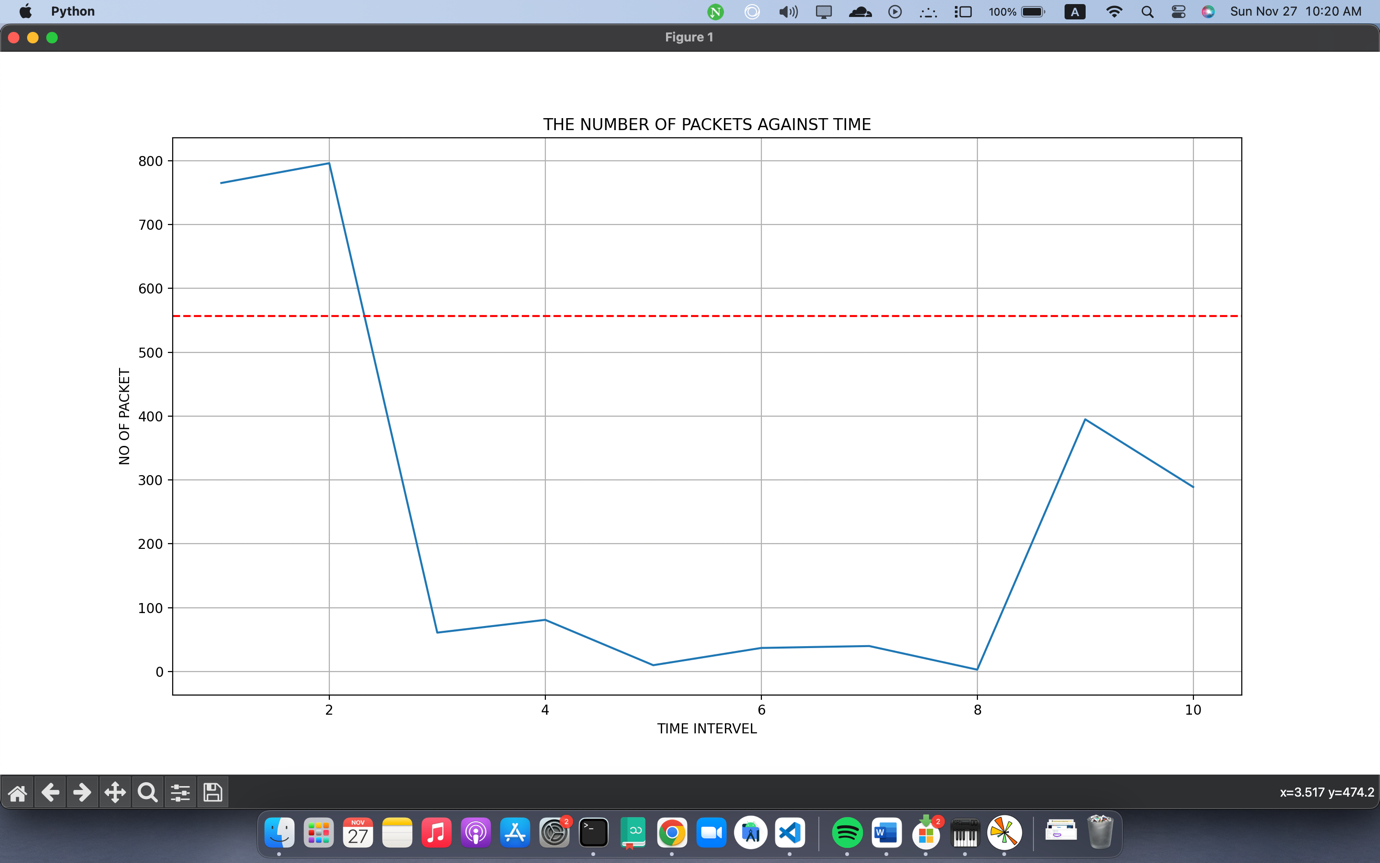
Screenshot of application folder before and after running the program.

## Screenshot of the Network graph



## Screenshot of the Timestamp line chat



# All of the code

Codes from **pcap\_analyzer.py**

'''

Run This File As main Program

'''

import sys

import packet\_information as pkt\_info

import graph\_visualizer as gv

def main():

'''

Main Function

can add for the additional .pcap information

'''

try:

with open('output.txt','w',encoding='utf-8') as sys.stdout:

packets = pkt\_info.prase\_packets(sys.argv[1])

pkt\_info.show\_summerize\_table(packets)

pkt\_info.email\_image(packets)

unused\_fullpkt, packets\_count = pkt\_info.packet\_ip\_info(packets)

gv.graph\_visulize(packets\_count)

gv.timestamp\_linechart(packets)

except IndexError:

print('[!] need to input one argument')

except TypeError:

print('[!] need to input one argument to function')

if \_\_name\_\_ == "\_\_main\_\_":

main()

Code from **packet\_information.py**

""""

This the course work module each function is one question

"""

import socket

import datetime

import sys

import re

import os

import dpkt

from tabulate import tabulate

def prase\_packets(pcap\_file):

'''

Prasing the packets from .pcap file to python data type

'''

# creating folder for the output files

if not os.path.exists('output'):

os.mkdir('output')

# getting error\_log.txt from sys.stderr

with open('output/error\_log.txt', 'w', encoding='utf-8') as sys.stderr:

print(f"[#] Prasing the packets from {pcap\_file}", file=sys.stderr)

try:

with open(pcap\_file, "rb") as file:

pcap = dpkt.pcap.Reader(file)

packets = []

for (time\_stamp, buff) in pcap:

packets.append((time\_stamp, buff))

return packets

except FileNotFoundError:

print('\t', "[!] file does not exist", file=sys.stderr)

return False

def dt\_from\_ts(time\_stamp):

'''

not for a coursework just try not to do repeatly

'''

return datetime.datetime.utcfromtimestamp(time\_stamp)

def is\_other(packet):

'''

Checking whether not TCP or UDP or IGMP

'''

try:

if not isinstance(packet, dpkt.igmp.IGMP):

if not isinstance(packet, dpkt.udp.UDP):

if not isinstance(packet.data, dpkt.tcp.TCP):

return True

else:

return False

except TypeError:

print('[!] Need to input one ip.data as arugment')

def show\_summerize\_table(packets):

"""

showing summerize table for TCP,UDP,IGMP and others

"""

with open('output/error\_log.txt', 'a', encoding='utf-8') as sys.stderr:

print('[#] Trying to show the summerize table', file=sys.stderr)

# creating constant variables for packet type

no\_total = 0

total\_buff = 0

packet\_type = {

'igmp': dpkt.igmp.IGMP,

'tcp': dpkt.tcp.TCP,

'udp': dpkt.udp.UDP}

packet\_info = {

'tcp': [], 'udp': [], 'igmp': [], 'other': [],

'l\_tcp': 0, 'l\_udp': 0, 'l\_igmp': 0, 'l\_other': 0,

'no\_tcp': 0, 'no\_udp': 0, 'no\_igmp': 0, 'no\_other': 0}

# creating constant variables for tables

table = []

table\_header = ['First Timestamp', 'Last Timestamp', 'Mean Packet Length', 'No of Packets']

pkt\_type = ['tcp', 'udp', 'igmp', 'other']

pkt\_type\_clone = pkt\_type.copy()

table\_index = [p\_type.upper() for p\_type in pkt\_type]

try:

# looping for the time stamp and buffer

for time\_stamp, buff in packets:

total\_buff += len(buff)

eth = dpkt.ethernet.Ethernet(buff)

ip\_frame = eth.data

# filtering the packets which are TCP,UDP,IGMP in the looping

for key, value in packet\_type.items():

if isinstance(ip\_frame.data, value):

packet\_info[f'no\_{key}'] += 1

packet\_info[key].append(time\_stamp)

packet\_info[f'l\_{key}'] += len(buff)

no\_total += 1

# filtering the packets which are not TCP,UDP,IGMP

if is\_other(ip\_frame):

packet\_info['no\_other'] += 1

packet\_info['other'].append(time\_stamp)

packet\_info['l\_other'] += len(buff)

no\_total += 1

# removing types of 0 packets from table

for value in pkt\_type\_clone:

if packet\_info[f'no\_{value}'] == 0:

pkt\_type.remove(value)

table\_index.remove(value.upper())

# appending data to the table list to show in the table

for value in pkt\_type:

table.append([

dt\_from\_ts(packet\_info[value][0]),

dt\_from\_ts(packet\_info[f'{value}'][-1]),

packet\_info[f'l\_{value}']/packet\_info[f'no\_{value}'],

packet\_info[f'no\_{value}']

])

# showing table to users

print(tabulate(table, headers=table\_header, showindex=table\_index, tablefmt='fancy\_grid'))

print(f"[\*]Total buffer : {total\_buff}")

print(f"[\*]No of total packet: {no\_total}", '\n')

except TypeError:

print('\t', '[!] the inputed arguments have to be list', file=sys.stderr)

def packet\_ip\_info(packets):

'''

Showing overall packet information

'''

full\_ip\_info = {}

sorted\_ip = {}

with open('output/error\_log.txt', 'a', encoding='utf-8') as sys.stderr:

print('[#] try to show overall packet information', file=sys.stderr)

try:

# getting packets by looping the inputed list

for unused\_time\_stamp, buff in packets:

eth = dpkt.ethernet.Ethernet(buff)

ip\_addr\_pair = (socket.inet\_ntoa(eth.data.src), socket.inet\_ntoa(eth.data.dst))

if (ip\_addr\_pair) not in full\_ip\_info.keys():

full\_ip\_info[ip\_addr\_pair] = [eth.data]

else:

full\_ip\_info[ip\_addr\_pair].append(eth.data)

# counting the length of ip\_address\_pair's traffics and create new variable

sorted\_ip = {keys: len(values) for keys, values in full\_ip\_info.items()}

sorted\_ip = dict(sorted(sorted\_ip.items(), key=lambda item: item[1]))

# changing IP dictoionary to List for tabulate

sorted\_ip\_traffic = [[f'{k[0]} -> {k[1]}', v] for k, v in sorted\_ip.items()]

table\_headers = ['IP Information', 'Numbers of Traffic']

print(tabulate(reversed(sorted\_ip\_traffic), headers=table\_headers, tablefmt='fancy\_grid'))

return full\_ip\_info, sorted\_ip

except TypeError:

print('\t', '[!] need to input one list[] argument', file=sys.stderr)

return 0, False

except dpkt.UnpackError as exp:

print('\t', exp, file=sys.stderr)

def email\_image(data\_list):

'''

Analyzing the image and email from the Packets

'''

with open('output/error\_log.txt', 'a') as sys.stderr:

print("[#] trying to show the email and files", file=sys.stderr)

url\_list = []

img\_ext = ['.gif', '.png', '.jpg', '.jpeg']

email = {'to': [], 'from': []}

# creating regular expression as constant

to\_pattern = re.compile(r'[\w+\s]\*To:\*\s?[<]\*\w+@\w+.\w+[>]\*', re.I)

from\_pattern = re.compile(r'[\w+\s]\*From:\*\s?[<]\*\w+@\w+.\w+[>]\*', re.I)

pattern = {'to': to\_pattern, 'from': from\_pattern}

for unused\_ts, buff in data\_list:

eth = dpkt.ethernet.Ethernet(buff)

# checking if packet is TCP

if isinstance(eth.data.data, dpkt.tcp.TCP):

ip\_frame = eth.data

src = socket.inet\_ntoa(ip\_frame.src)

dst = socket.inet\_ntoa(ip\_frame.dst)

tcp = ip\_frame.data

# checking for the SMTP protocol

try:

if tcp.dport in (587, 1687):

# decoding data from the tcp packet

data = tcp.data.decode('utf-8', 'ignore')

for key, value in pattern.items():

if not value.match(data) is None:

if not value.match(data).group().split('<')[1][0:-1] in email[key]:

email[key].append(value.match(data).group().split('<')[1][0:-1])

# checking HTTP or HTPPS by port

proto = "http://" if tcp.dport == 443 else "https://"

http = dpkt.http.Request(tcp.data)

if http.method == "GET":

uri = http.uri.lower()

for ext in img\_ext:

if ext in uri:

# creating the full url

full\_link = f"{proto}{http.headers['host']}{uri}"

# appeding full\_url and base to the url\_list

url\_list.append((full\_link, os.path.basename(uri)))

print(f"[\*]{src} downloaded {os.path.basename(uri)} from {dst}")

print(f"[\*]Full url: {full\_link}")

print(f"[\*]File Name: {os.path.basename(uri)}",'\n')

except dpkt.UnpackError as exp:

print('\t', exp, file=sys.stderr)

if len(url\_list) == 0:

print('[!] There is no file download in these packets')

# filtering the empty list for not to show empty table

if len(email['from']) != 0 and len(email['to']) != 0:

header\_list = [

'Emails detect To: format',

'Emails detect From: format']

print(tabulate(email, headers=header\_list, tablefmt='fancy\_grid'))

else:

print('[!] Emails not found in these packets', '\n')

Codes from **graph\_visualizer.py**

'''

This moudule show the packets with graph to check the packets easily

'''

import sys

import statistics as stats

import dpkt

import matplotlib.pyplot as plt

import networkx as nx

def graph\_visulize(data):

'''

Creating Graph to recognize easily the attacker's traffic

'''

graph = nx.DiGraph()

with open('output/graph\_output.txt', 'w') as sys.stdout:

try:

for key, value in dict(data).items():

# creating the color scheme for graph

if value < 50:

if value < 10:

print(f'[!]{key[0]} --> {key[1]} weakly connected')

colour = 'black'

elif value < 100:

colour = 'g'

elif value < 200:

colour = 'y'

print(f'[!]{key[0]} --> {key[1]} strongly connected')

else:

colour = 'r'

print(f'[!]{key[0]} --> {key[1]} strongly connected')

graph.add\_edge(key[0], key[1], width=value/100\*2, color=colour)

# getting width and colour list from the graph edges

colours = nx.get\_edge\_attributes(graph, 'color').values()

width = nx.get\_edge\_attributes(graph, 'width').values()

# choosing the layout for the graph

pos = nx.planar\_layout(graph)

nx.draw(graph,

pos,

edge\_color=colours,

width=tuple(width),

with\_labels=True,

node\_color='lightgreen')

nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=data)

plt.show()

# showing number of nodes and edges to the terminal for the user

print(f'[\*]number of nodes: {len(graph)}')

print(f'[\*]number of edges: {graph.number\_of\_edges()}')

except TypeError:

print('[!]graph\_visualize() need to be inputed one dict')

except dpkt.UnpackError as exp:

print(exp)

def timestamp\_linechart(pcap\_list):

'''

Showing line graph to user to check which interval has most traffic

'''

# opening .txt file for error or exception logs.

with open('output/error\_log.txt', 'a') as sys.stderr:

print(

'[#] showing line chart of traffic against the time',

file=sys.stderr)

try:

no\_intervel = 10

# duration of the pcap file

timestamp\_difference = pcap\_list[-1][0]-pcap\_list[0][0]

# calculating the one intervel

one\_intervel = timestamp\_difference/no\_intervel

timestamp\_counter = pcap\_list[0][0]

interval\_counter = 1

buff\_dict = {}

timestamp\_list = []

for (time\_stamp, buff) in pcap\_list:

if timestamp\_counter+one\_intervel >= time\_stamp:

if interval\_counter not in buff\_dict:

buff\_dict[interval\_counter] = [buff]

timestamp\_list.append(time\_stamp)

else:

buff\_dict[interval\_counter].append(buff)

else:

timestamp\_counter += one\_intervel

interval\_counter += 1

traffic\_intervals = {

keys: len(values) for keys, values in buff\_dict.items()}

# finding mean number of packets per interval

mean\_list = [values for keys, values in traffic\_intervals.items()]

mean = stats.mean(mean\_list)

threshold = mean+stats.stdev(mean\_list)

plt.plot(traffic\_intervals.keys(), traffic\_intervals.values())

plt.title('the number of packets against time'.upper())

plt.xlabel('time intervel'.upper())

plt.ylabel('no of packet'.upper())

plt.axhline(y=threshold, color='r', linestyle='--')

plt.grid(True)

plt.show()

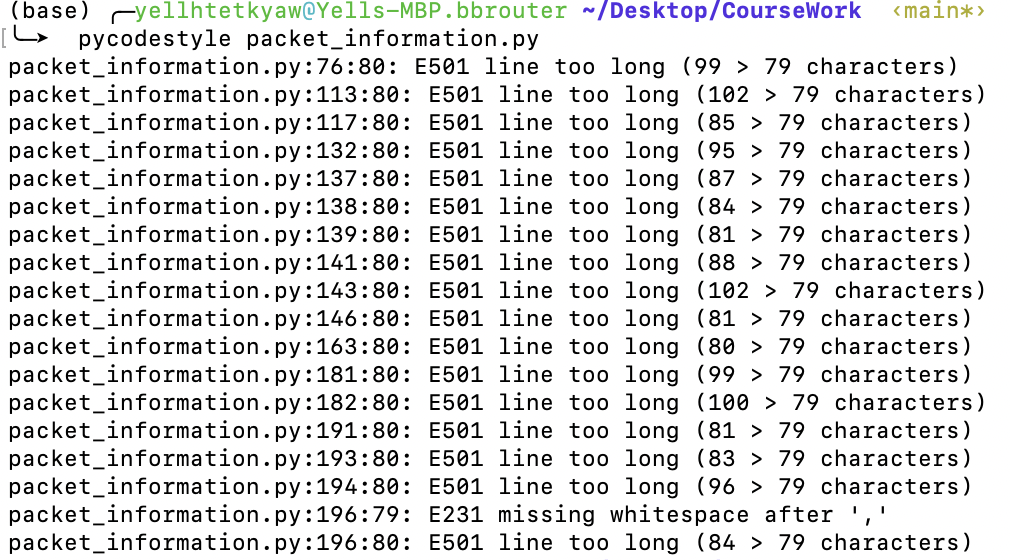
except TypeError:

print(

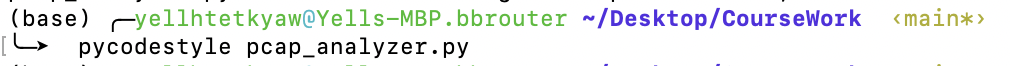
'\t',

'[!] need to input list with (timestamp and buffer)',file=sys.stderr)

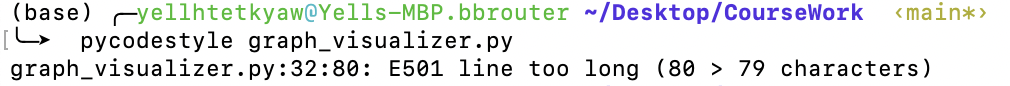
### Pycodestyle output for packet\_information.py



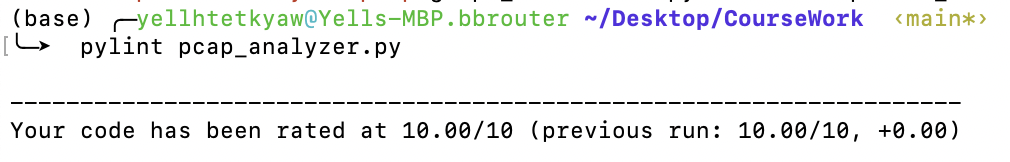
### Pycodestyle output for pcap\_analyzer.py



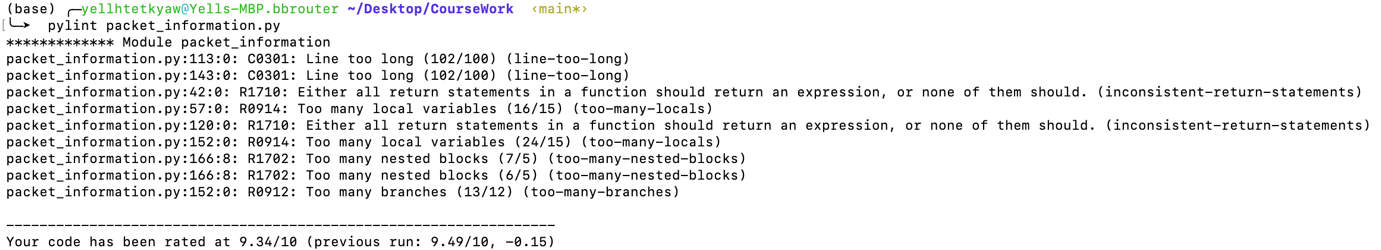
### Pycodestyle output for graph\_visualizer.py



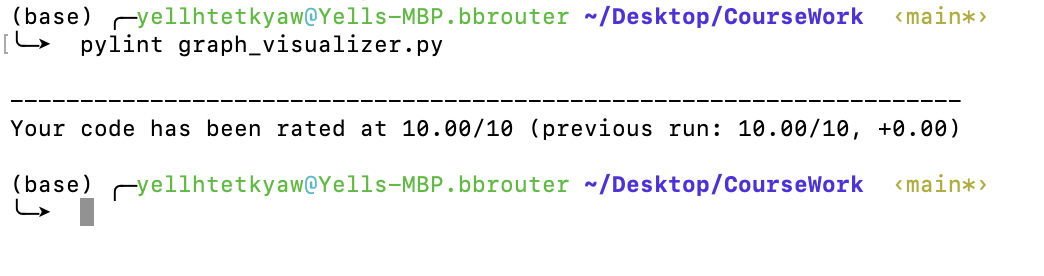
### Pylint output for pcap\_analyzer.py



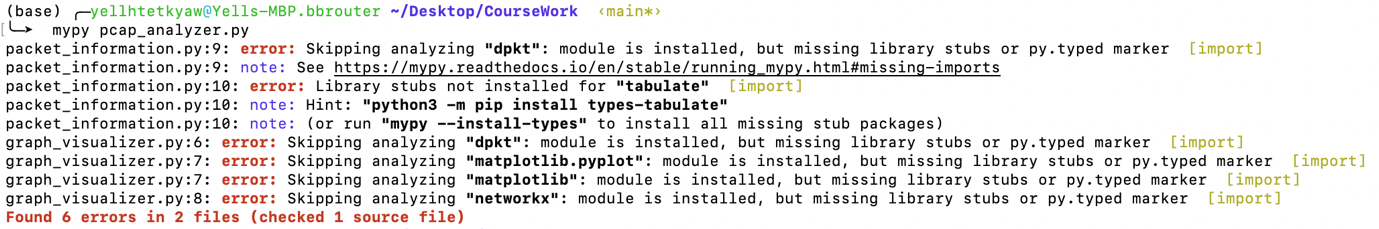
### Pylint output for packet\_information.py



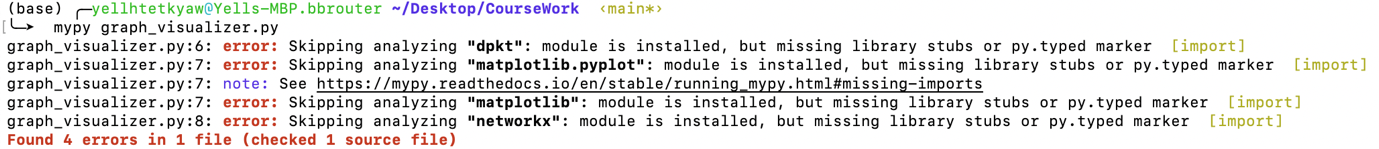
### Pylint output for graph\_visualizer.py



### Mypy output for pcap\_analyzer.py



### Mypy output for graph\_visualizer.py



### Mypy output for packet\_information.py

