Waiyan Phonemyat Report Assessment

by Waiyan Phone Myat

Submission date: 03-Dec-2021 09:58AM (UTC+0000)
Submission ID: 164796013
File name: 110681_Waiyan_Phone_Myat_Waiyan_Phonemyat_Report_Assessment_1842664_1841937895.docx (2.53M)
Word count: 1641
Character count: 10908

REPORT ASSESSMENT FOR SCRIPTING

Waiyan Phonemyat

Report and Submission

1(a)

In my submission pythons files are connected to each other's. There is a file which contain the main function that can run other function from its. It is called pcap_analyser.py. The filenames of other sub functions are packet.py, email_and_url.py, srt_dst.py and finally geolocation_kml.py. Since pcap_analyser.py is center file, it has to import other python filenames to in order to run functions of them. Inside the center file there are two function, main and enter function. The enter function is created specifically to open the pcap file (evidence-packet-analysis.pcap). To use dpkt.pcap.Reader method for reading the pcap file we need to import dkpt module.

In packet.py file there is a function named "capturingpcap (loop_for_return)". This can be used to calculate the packets amount, first and last timestamps and finally mean packet length of TCP, UDP and IGMP Protocols. Using the variable buffer (buf) we can count the lengths of all three protocols then divided them with number of packets to get mean packet length of each other's.

In order to extract the 'TO' and 'FROM' email from the evidence-pcap-file we need the function findingemail (loop_for_return). To find the image files uri path based on the extensions we need to declare the TCP protocol from the buffer and also request the http from tcp method.

To extract the source and destination ip addresses inside src_dst.py file, I created the list that include the source and destination ip address and put it in the dictionary format to sort and count the packets.

In geolocation_kml.py python file I had to install a geo location database file in order to put the longitude, latitude, city and country. Then declare the the kml new point with newly inserted data then kml file is ready to run on the google earth.

1(b)

Exception handling

There are 3 main error that can be found in my 5 python files, they are FileNotFoundError, ZeroDivisionError and Exception. We can solve them by catching between try and except. In first packet_analyser.py file there is a FileNotFoundError in case there is no pcap file inputted. In second file packet.py there is a ZeroDivisionError. In other file I added the exception method to catch the errors. Among those problems ZeroDivisonError is tough to solve because it can occurred when the result of denominator or second arg of the division is zero.

1(c)

Finding the last time stamp of each packet type

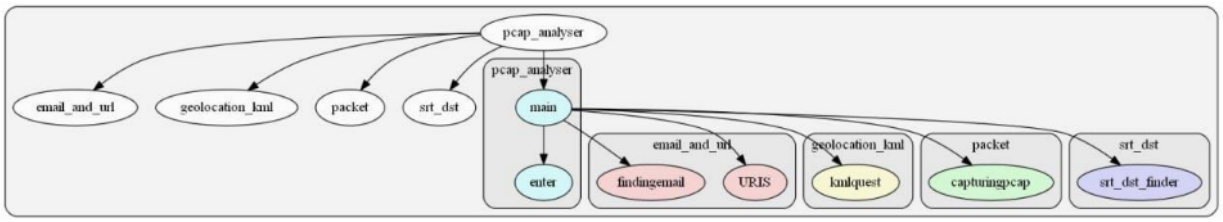
In packet.py file I found the first and last timestamps of 3 packets TCP, UDP and IGMP. At first I created the timestamp list for those 3 packets such as (tcptimestamplist=[], udptimestamplist=[] and igmptimestamplist=[]). Then I inserted the data of ts into those list with append method. Since this idea of method is list we can find the last timestamp by outputting the last number of it. In this case [0] is the first value in the list and [-1] is the last one. To print for the three lists we can do that by (print(tcptimestamplist [-1]), (print(udptimestamplist [-1]) and (print(igmptimestamplist [-1]))

1(d)

Creating the KML file with a suitable description for each point

To create a preferable kml file firstly we need to put a path of a geo database file inside the geolocation_kml.py file. Then declare a list and put the destination ip addresses into that list the count the list and put it inside the dictionary. Inside that dictionary there are key and values. There state the longitude, latitude, country and city and add those to the kml.newpoint function. In kml.newpoint there are name, coordinates and description. I added the country for the name, longitude latitude into coordinates and finally add the city name to description. I also add the packet counts into the description with the value (v). Finally save the kml file into the current working folder like (kml.save("cwkml.kml")). If I print it kml file will be created automatically.

2. Dependency Diagram



Figurer 2.1 : Dependency Diagram

3. Table

Number	Modules	Functions	Description
1	pcap_analyser.py	def enter(loop_for_return) def main()	enter is to open the pcap file, parse it and then close main is the main stage of calling the other functions
2	packet.py	def capturingpcap(loop_for_return)	capturingpcap is used for TCP,UDP and IGMP
3	email_and_url.py	def findingemail(loop_for_return) def URIS(loop_for_return)	findingemail is to find to and from email types URIS is to find the image file inside that uri.
4	srt_dst.py	def srt_dst_finder(loop_for_return)	srt_dst_finder is to find the source and destination ip addresses.
5	geolocation_kml.py	def kmlquest(loop_for_return)	kmlquest is used to create the longitude, latitude, country and city.

4. Code for my scripts

pcap_analyser.py

```
"""import the modules  to  call the function"""
import dpkt
import packet
import srt_dst
import email_and_url
import geolocation_kml



def enter(evidance_pcap_file):
    # This script is created to open the pcap file
    pcapfile_open = open(evidance_pcap_file, "rb")
    pcap = dpkt.pcap.Reader(pcapfile_open)
    # To store the buffer and timestamp inside the list
    try:
        loop_for_return = []

        for (ts, buf) in pcap:
            loop_for_return.append([ts, buf])
        pcapfile_open.close()

        return loop_for_return
    except FileNotFoundError:
        print(f"file is not found")



def main():
    # This script is to return and call the function of other modules
```

```
    evidance_pcap_file = "evidence-packet-analysis.pcap"
    loop_for_return = enter(evidance_pcap_file)
    # function calls of the modules
    packet.capturingpcap(loop_for_return)
    email_and_url.findingemail(loop_for_return)
    email_and_url.URIS(loop_for_return)
    srt_dst.srt_dst_finder(loop_for_return)
    geolocation_kml.kmlquest(loop_for_return)



if __name__ == "__main__":
    main()
```

packet.py

```
from datetime import datetime
import dpkt
from prettytable import PrettyTable



def capturingpcap(loop_for_return):

    #timestamp, length and count of tcp, udp and igmp
    tcp_length = []
    tcp_total = 0
    tcptimestamplist = []

    udp_length = []
    udp_total = 0
    udptimestamplist = []
```

```
igmp_length = []
igmp_total = 0
igmptimestamplist = []
try:
    for (ts, buf) in loop_for_return:
        eth = dpkt.ethernet.Ethernet(buf)
        ip_a = eth.data

        #condition checking of TCP, UDP and IGMP

        if ip_a.p == dpkt.ip.IP_PROTO_TCP:

            tcp_total += 1
            tcp_length.append(len(buf))
            tcptimestamplist.append(
                datetime.fromtimestamp(ts).strftime("%m/%d/%y %H:%M:%S")
            )

        if ip_a.p == dpkt.ip.IP_PROTO_UDP:

            udp_total += 1
            udp_length.append(len(buf))
            udptimestamplist.append(
                datetime.fromtimestamp(ts).strftime("%m/%d/%y %H:%M:%S")
            )

        if ip_a.p == dpkt.ip.IP_PROTO_IGMP:

            igmp_total += 1
            igmp_length.append(len(buf))
```

```
        igmptimestamplist.append(
            datetime.fromtimestamp(ts).strftime("%m/%d/%y %H:%M:%S")
        )

    #to calculate the mean we need to divided the length of buffer with total protocol
    mean_tcp_packet_length = sum(tcp_length) / tcp_total
    mean_udp_packet_length = sum(udp_length) / udp_total
    mean_igmp_packet_length = sum(igmp_length) / igmp_total

    #Printing stage of TCP, UDP and IGMP results within the tables"""
    #I took a refernce of the table from the site (https://zetcode.com/python/prettytable/)
    x = PrettyTable()
    x.field_names = [
        "Data_Types",
        "PacketsNumber",
        "First Timestamp",
        "Last Timestamp",
        "MeanPacketLength",
    ]
    x.add_row(
        [
            "TCP",
            tcp_total,
            tcptimestamplist[0],
            tcptimestamplist[-1],
            mean_tcp_packet_length,
        ]
    )
    x.add_row(
        [
            "UDP",
```

```
            udp_total,
            udptimestamplist[0],
            udptimestamplist[-1],
            mean_udp_packet_length,
        ]
    )
    x.add_row(
        [
            "IGMP",
            igmp_total,
            igmptimestamplist[0],
            igmptimestamplist[-1],
            mean_igmp_packet_length,
        ]
    )
    print(x)

    with open("output.txt", "w") as file:
        file.write(str(x) + "\n\n")
        file.close()
    return x
except ZeroDivisionError as err:
    print(f"Error there is no value inside", err)
```

email_and_url.py

```
import re
import os
import os.path
import dpkt



def findingemail(loop_for_return):
    to_mail_list = []
    from_mail_list = []

    for unused_ts, buf in loop_for_return:
        eth = dpkt.ethernet.Ethernet(buf)
        ip_a = eth.data
        tcp = ip_a.data
        if ip_a.p == dpkt.ip.IP_PROTO_TCP:
            string = tcp.data.decode("utf-8", "ignore")
            to_mail = re.findall(
                r"TO:\s?<([A-Za-z0-9]+)*[A-Za-z0-9]+@[A-Za-z0-9-]+(.[A-Zla-z]{2,})+>",
                string,
                re.I,
            )
            from_mail = re.findall(
                r"FROM:\s?<([A-Za-z0-9]+)*[A-Za-z0-9]+@[A-Za-z0-9-]+(.[A-Zla-z]{2,})+>",
                string,
                re.I,
            )

            for tomail in to_mail:
```

```
            print(f"The to mails are {tomail}")
        if len(to_mail):
            to_mail_list.append(to_mail)
        for frommail in to_mail:
            print(f"The from mails are{frommail}")
        if len(from_mail):
            from_mail_list.append(from_mail)



def URIS(loop_for_return):
    string = ""
    try:
        for (unused_ts, buf) in loop_for_return:

            eth = dpkt.ethernet.Ethernet(buf)
            ip_a = eth.data
            tcp = ip_a.data

#Took the reference of stating condition of http from the https://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file/"""

            if ip_a.p == dpkt.ip.IP_PROTO_TCP:

                if tcp.dport == 80 and len(tcp.data) > 0:

                    http = dpkt.http.Request(tcp.data)
                    if http.method == "GET":

                        uripath = http.uri.lower()
                        if ".jpg" in uripath:
                            print(f"JPG URI: {uripath}")
```

```
                    string += f"JPG URI: {uripath}\n"
                    print(
                        f"Extracted JPG filename: {os.path.basename(uripath)}"
                    )
                    string += (
                        f"Extracted JPG filename: {os.path.basename(uripath)}\n"
                    )
                if ".png" in uripath:
                    print(f"PNG URI: {uripath}")
                    string += f"PNG URI: {uripath}\n"
                    print(f"Extracted PNG filename {os.path.basename(uripath)}")
                    string += (
                        f"Extracted PNG filename {os.path.basename(uripath)}\n"
                    )

                if ".gif" in uripath:
                    print(f"GIF URI {uripath}")
                    string += f"GIF URI {uripath}\n"
                    print(
                        f"Extracted GIF filename:{os.path.basename(uripath)}"
                    )
                    string += (
                        f"Extracted GIF filename: {os.path.basename(uripath)}\n"
                    )

    with open("output.txt", "a") as file:
        file.write(string + "\n\n")
        file.close()
except Exception as error:
    print(f"Wrong data", error)
```

srt_dst.py

```
from collections import Counter
import operator
import socket
import dpkt



def srt_dst_finder(loop_for_return):
    src_plus_dst_list = []
    string = ""
    total_ip_counted = {}
    try:

        for (unused_ts, buf) in loop_for_return:
            eth = dpkt.ethernet.Ethernet(buf)
            ip_a = eth.data
            src = socket.inet_ntoa(ip_a.src)
            dst = socket.inet_ntoa(ip_a.dst)
            src_plus_dst_list.append((src, dst))

        count_dictionary = Counter(src_plus_dst_list)
        # i took a reference of how to use itemgetter from stack overflow
        #      https://stackoverflow.com/questions/18595686/how-do-operator-itemgetter-and-sort-
work
        sort_dictionary = sorted(
            count_dictionary.items(), key=operator.itemgetter(1), reverse=True
        )

        for key, value in sort_dictionary:
            total_ip_counted.setdefault(key, []).append(value)
```

```
        print(total_ip_counted)

        string += f"{total_ip_counted}"
        with open("output.txt", "a") as file:
            file.write(string + "\n\n")
            file.close()

    except Exception:
        print(f"There is no data inside")
    return sort_dictionary
```

geolocation_kml.py

```
from collections import Counter
import socket
import dpkt
import simplekml
import geoip2.database



def kmlquest(loop_for_return):
    reader = geoip2.database.Reader(r"E:\python\Coursework\GeoLite2-City_20190129.mmdb")

    total_ip_list = []
    dictionary_of_ip = {}
    kml = simplekml.Kml()
```

```
try:
    for (unused_ts, buf) in loop_for_return:
        eth = dpkt.ethernet.Ethernet(buf)
        ip_a = eth.data
        dst = socket.inet_ntoa(ip_a.dst)
        total_ip_list.append(dst)
        dictionary_of_ip = dict(Counter(total_ip_list))

    for k, v_count in dictionary_of_ip.items():
        try:
            # took a reference from lab 9 Geoloaction inside the ENU moodle
            rec = reader.city(k)
            read_long = rec.location.longitude
            read_lat = rec.location.latitude
            read_country = rec.country.name
            read_city = rec.city.name
            # took a reference from the napier_KML.py from Topic9file
            kml.newpoint(
                name=f"{read_country}",
                coords=[(f"{read_long}", f"{read_lat}")],
                description=f"City Name:{read_city} Packets Counted:{v_count}",
            )

        except Exception:
            pass
    kml.save("cwkml.kml")
    print(kml.kml())

    with open("output.txt", "a") as file:
        file.write(str(kml.kml()) + "\n\n")
        file.close()
```

```
except Exception:
    print("There is an error")
```

5. Before running the file and After Running the File

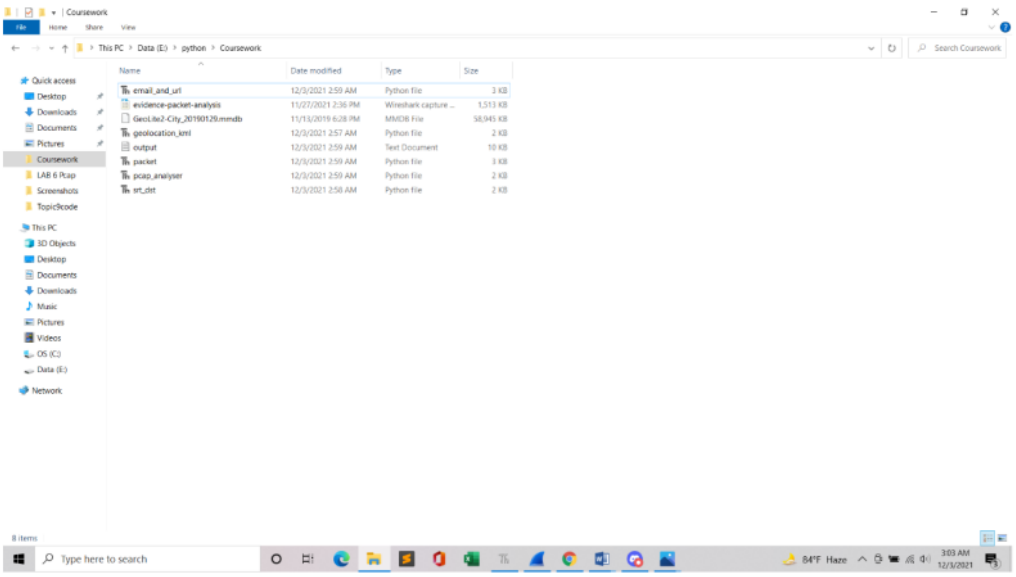


Figure 5.1: Before Running File

After running the file KML file is added

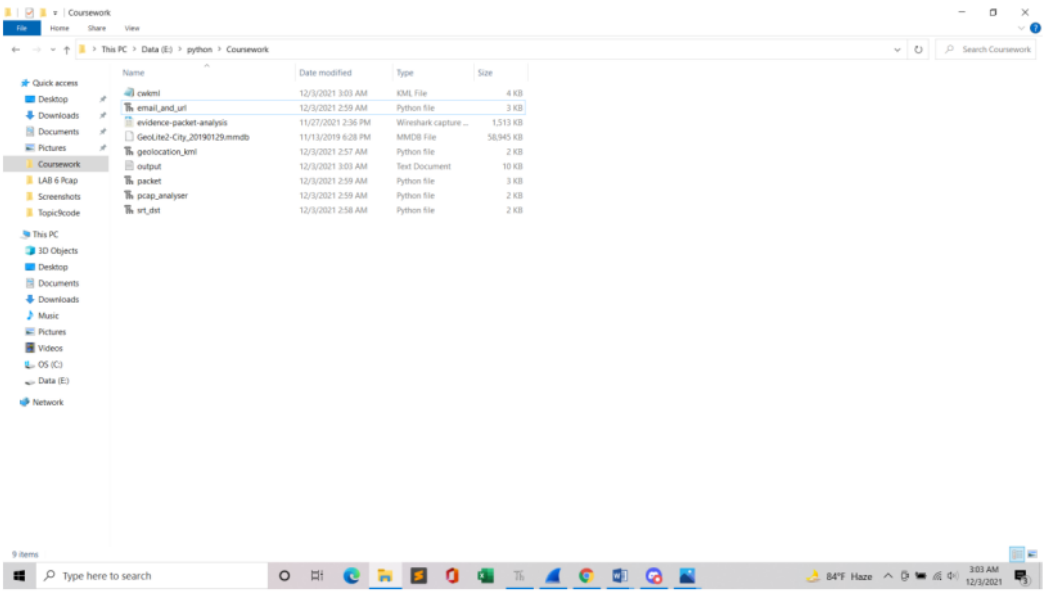


Figure 5.2: After Running File

6. Screenshots of KML file and testing it on KML file on google earth

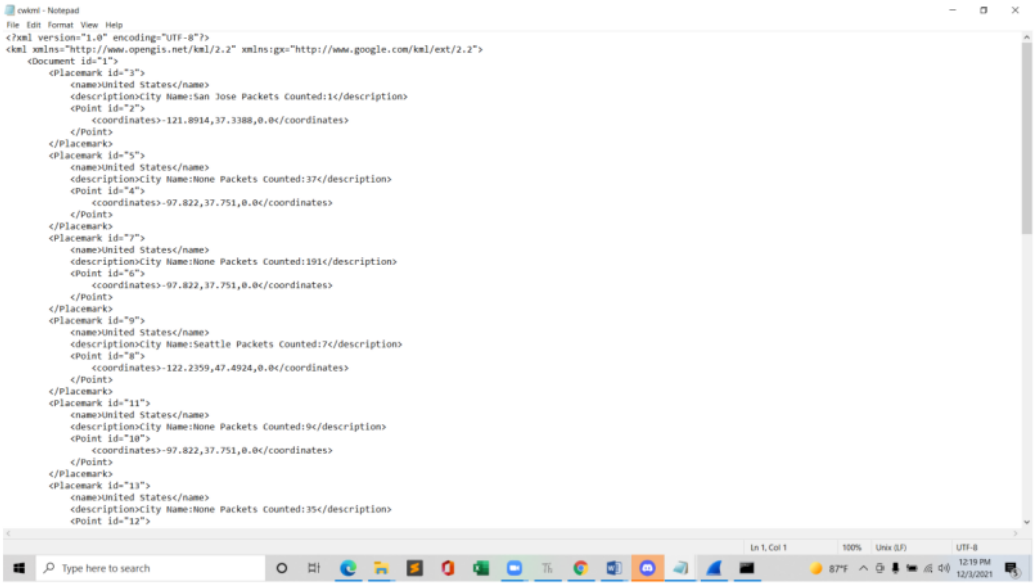


Figure 6.1 KML file information

In this screen shot there are many points and location of the country with exact longitude and latitude including city name and packet counts.

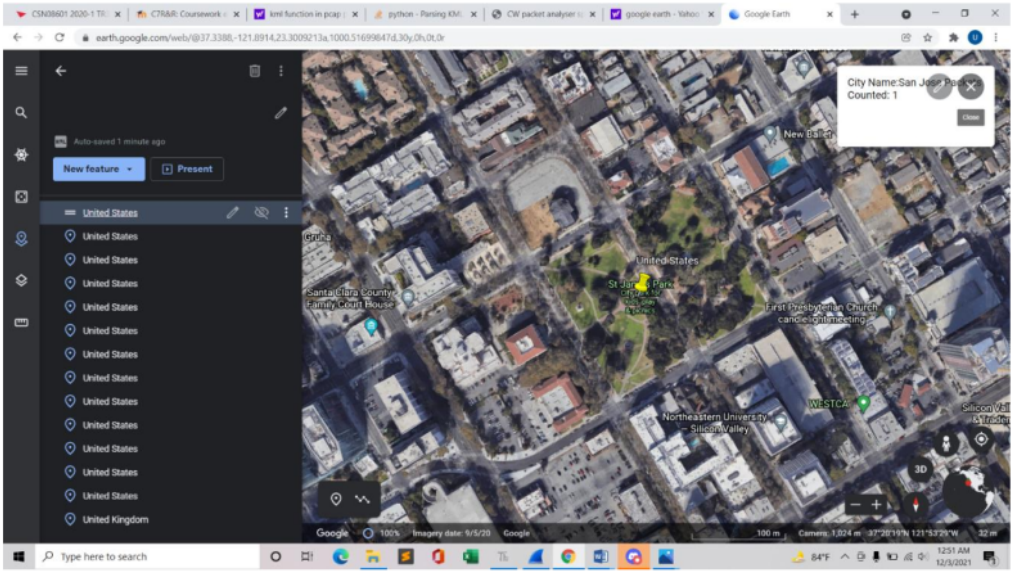


Figure 6.2 KML file on google earth

7. Evidence of pylint and pycodestyle

```
E:\python\Coursework>pylint pcap_analyser.py
************* Module pcap_analyser
pcap_analyser.py:9:0: C0116: Missing function or method docstring (missing-function-docstring)
pcap_analyser.py:17:13: C0103: Variable name "ts" doesn't conform to snake_case naming style (invalid-name)
pcap_analyser.py:23:14: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)
pcap_analyser.py:9:0: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)
pcap_analyser.py:11:20: R1732: Consider using 'with' for resource-allocating operations (consider-using-with)
pcap_analyser.py:26:0: C0116: Missing function or method docstring (missing-function-docstring)

-----------------------------------------------------------------
Your code has been rated at 7.69/10 (previous run: 7.69/10, +0.00)
```

Figure 7.1: pylint and pycodestyle of pcap_analyser.py

```
E:\python\Coursework>pylint packet.py
************* Module packet
packet.py:1:0: C0114: Missing module docstring (missing-module-docstring)
packet.py:6:0: C0116: Missing function or method docstring (missing-function-docstring)
packet.py:6:0: R0914: Too many local variables (20/15) (too-many-locals)
packet.py:21:13: C0103: Variable name "ts" doesn't conform to snake_case naming style (invalid-name)
packet.py:58:8: C0103: Variable name "x" doesn't conform to snake_case naming style (invalid-name)
packet.py:95:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
packet.py:100:14: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)
packet.py:6:0: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)

-----------------------------------------------------------------
Your code has been rated at 8.18/10 (previous run: 7.95/10, +0.23)
```

Figure 7.2: pylint of packet.py

```
E:\python\Coursework>pylint email_and_url.py
************* Module email_and_url
email_and_url.py:49:0: C0301: Line too long (139/100) (line-too-long)
email_and_url.py:1:0: C0114: Missing module docstring (missing-module-docstring)
email_and_url.py:8:0: C0116: Missing function or method docstring (missing-function-docstring)
email_and_url.py:40:0: C0103: Function name "URIS" doesn't conform to snake_case naming style (invalid-name)
email_and_url.py:40:0: C0116: Missing function or method docstring (missing-function-docstring)
email_and_url.py:89:11: W0703: Catching too general exception Exception (broad-except)
email_and_url.py:86:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
email_and_url.py:90:14: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)
email_and_url.py:42:4: R1702: Too many nested blocks (6/5) (too-many-nested-blocks)

-----------------------------------------------------------------
Your code has been rated at 8.36/10 (previous run: 7.82/10, +0.55)
```

Figure 7.3: pylint of email_and_url.py

```
E:\python\Coursework>pylint srt_dst.py
************* Module srt_dst
srt_dst.py:1:0: C0114: Missing module docstring (missing-module-docstring)
srt_dst.py:8:0: C0116: Missing function or method docstring (missing-function-docstring)
srt_dst.py:37:11: W0703: Catching too general exception Exception (broad-except)
srt_dst.py:33:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
srt_dst.py:38:14: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)

-----------------------------------------------------------------
Your code has been rated at 8.15/10 (previous run: 7.78/10, +0.37)
```

Figure 7.4: pylint of srt_dst.py

```
E:\python\Coursework>pylint geolocation_kml.py
************* Module geolocation_kml
geolocation_kml.py:1:0: C0114: Missing module docstring (missing-module-docstring)
geolocation_kml.py:9:0: C0116: Missing function or method docstring (missing-function-docstring)
geolocation_kml.py:9:0: R0914: Too many local variables (18/15) (too-many-locals)
geolocation_kml.py:47:11: W0703: Catching too general exception Exception (broad-except)
geolocation_kml.py:39:19: W0703: Catching too general exception Exception (broad-except)
geolocation_kml.py:44:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

-------------------------------------------------------------------
Your code has been rated at 8.24/10 (previous run: 7.94/10, +0.29)
```

Figure 7.5: pylint of geolocation_kml.py

```
E:\python\Coursework>pycodestyle pcap_analyser.py

E:\python\Coursework>pycodestyle packet.py
packet.py:8:5: E265 block comment should start with '# '
packet.py:25:13: E265 block comment should start with '# '
packet.py:51:9: E265 block comment should start with '# '
packet.py:51:80: E501 line too long (90 > 79 characters)
packet.py:56:9: E265 block comment should start with '# '
packet.py:57:9: E265 block comment should start with '# '
packet.py:57:80: E501 line too long (95 > 79 characters)

E:\python\Coursework>pycodestyle email_and_url.py
email_and_url.py:8:1: E303 too many blank lines (3)
email_and_url.py:19:80: E501 line too long (86 > 79 characters)
email_and_url.py:24:80: E501 line too long (88 > 79 characters)
email_and_url.py:49:1: E265 block comment should start with '# '
email_and_url.py:49:80: E501 line too long (139 > 79 characters)
email_and_url.py:63:80: E501 line too long (86 > 79 characters)
email_and_url.py:66:80: E501 line too long (88 > 79 characters)
email_and_url.py:71:80: E501 line too long (88 > 79 characters)
email_and_url.py:73:80: E501 line too long (87 > 79 characters)
email_and_url.py:80:80: E501 line too long (85 > 79 characters)
email_and_url.py:83:80: E501 line too long (88 > 79 characters)

E:\python\Coursework>pycodestyle srt_dst.py

E:\python\Coursework>pycodestyle geolocation_kml.py
geolocation_kml.py:9:80: E501 line too long (88 > 79 characters)
geolocation_kml.py:35:80: E501 line too long (83 > 79 characters)
```

Figure 7.6 pycodestyle of all 5 files