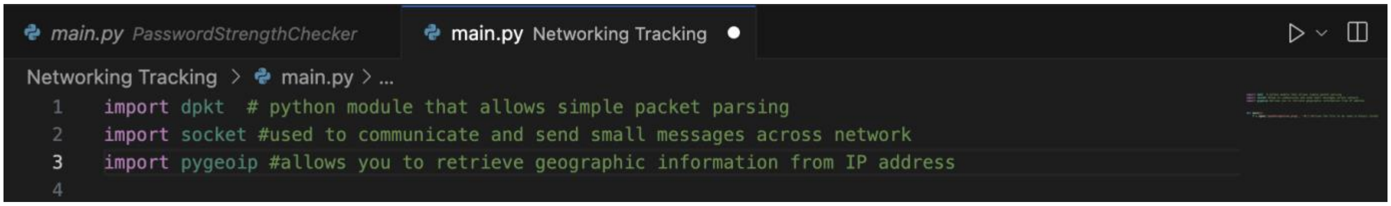
Visual Network Tracking using Wireshark
By Pranav

Basic Idea of the Python Program:

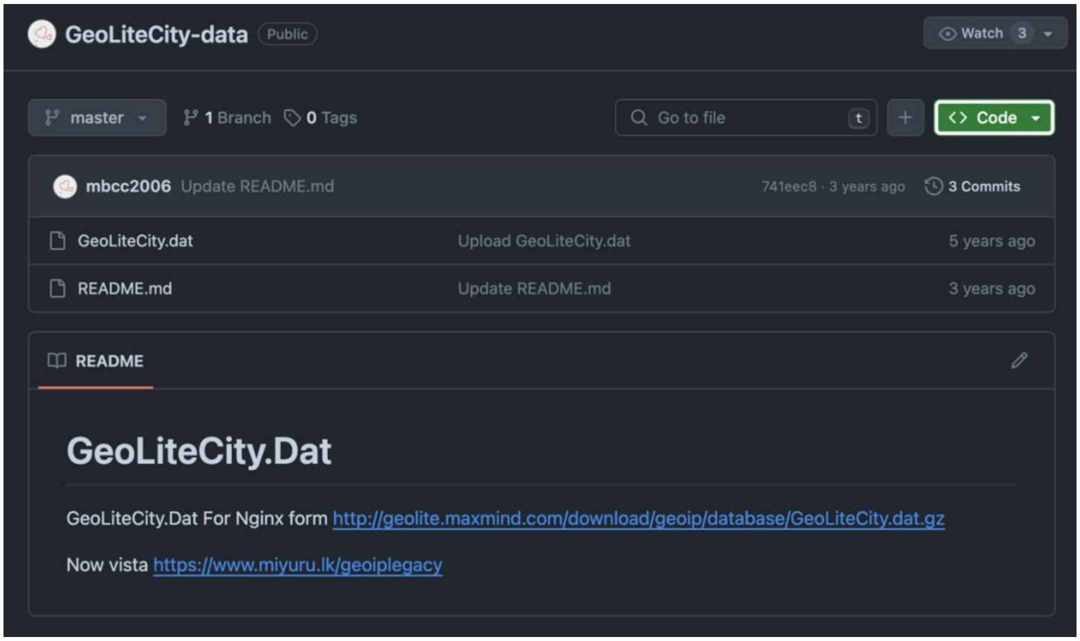
- The Python program takes a packet capture file **(packet.pcap)** as input from WireShark, processes the network traffic data contained within, and extracts source and destination IP addresses.
- It then utilizes a GeoIP database **(GeoLiteCity.dat)** to map these IP addresses to geographical coordinates (latitude and longitude).
- The program generates a KML file **(output.kml)** containing Placemarks for each IP communication, allowing visualization of the geographical distribution of network traffic.
- Overall, the program provides a visual representation of the geographic locations involved in network communications captured in the packet capture file.

Imported Libraries


main.py PasswordStrengthChecker
main.py Networking Tracking
Networking Tracking > main.py > ...

```python
import dpkt  # python module that allows simple packet parsing
import socket #used to communicate and send small messages across network
import pygeoip #allows you to retrieve geographic information from IP address

```

Download GeoLiteCity.Dat, database (GeoLiteCity.dat) for IP geolocation.



mbcc2006 Update README.md 741eec8 · 3 years ago 3 Commits

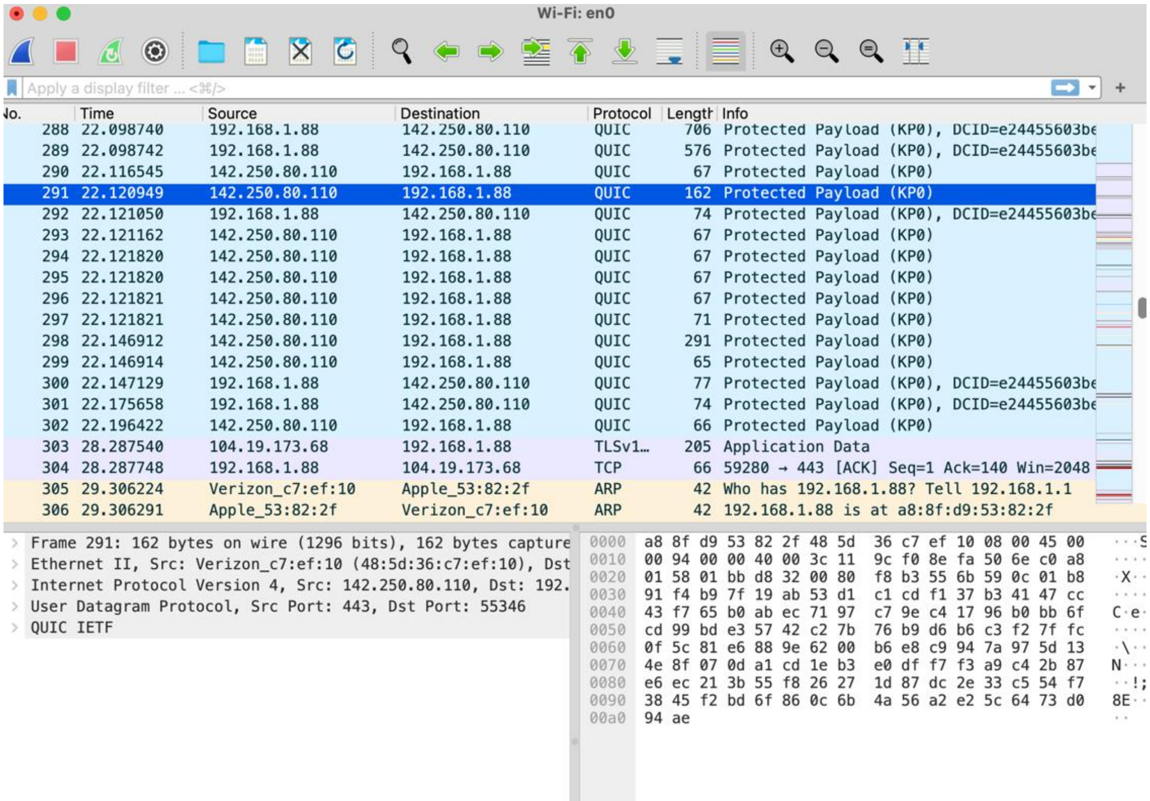
GeoLiteCity.dat	Upload GeoLiteCity.dat	5 years ago
README.md	Update README.md	3 years ago

GeoLiteCity.Dat

GeoLiteCity.Dat For Nginx form http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz

Now vista https://www.miyuru.lk/geoiplegacy

Capture and save the TCP replay packet captures.

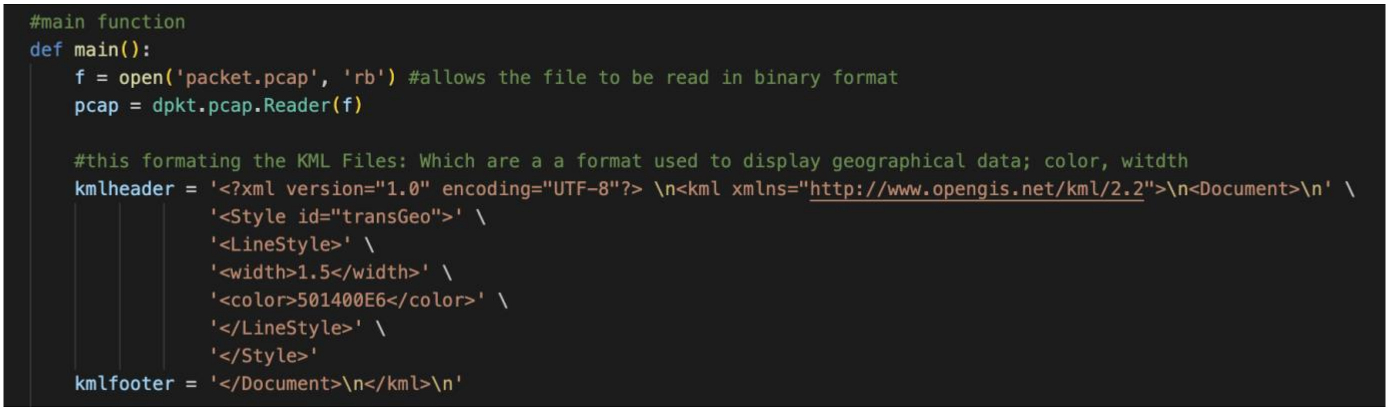


```
No.  Time       Source           Destination      Protocol Length Info
 288 22.098740  192.168.1.88     142.250.80.110   QUIC     706 Protected Payload (KP0), DCID=e24455603be
 289 22.098742  192.168.1.88     142.250.80.110   QUIC     576 Protected Payload (KP0), DCID=e24455603be
 290 22.116545  142.250.80.110   192.168.1.88     QUIC      67 Protected Payload (KP0)
 291 22.120949  142.250.80.110   192.168.1.88     QUIC     162 Protected Payload (KP0)
 292 22.121050  192.168.1.88     142.250.80.110   QUIC      74 Protected Payload (KP0), DCID=e24455603be
 293 22.121162  142.250.80.110   192.168.1.88     QUIC      67 Protected Payload (KP0)
 294 22.121820  142.250.80.110   192.168.1.88     QUIC      67 Protected Payload (KP0)
 295 22.121820  142.250.80.110   192.168.1.88     QUIC      67 Protected Payload (KP0)
 296 22.121821  142.250.80.110   192.168.1.88     QUIC      67 Protected Payload (KP0)
 297 22.121821  142.250.80.110   192.168.1.88     QUIC      71 Protected Payload (KP0)
 298 22.146912  142.250.80.110   192.168.1.88     QUIC     291 Protected Payload (KP0)
 299 22.146914  142.250.80.110   192.168.1.88     QUIC      65 Protected Payload (KP0)
 300 22.147129  192.168.1.88     142.250.80.110   QUIC      77 Protected Payload (KP0), DCID=e24455603be
 301 22.175658  192.168.1.88     142.250.80.110   QUIC      74 Protected Payload (KP0), DCID=e24455603be
 302 22.196422  142.250.80.110   192.168.1.88     QUIC      66 Protected Payload (KP0)
 303 28.287540  104.19.173.68    192.168.1.88     TLSv1…   205 Application Data
 304 28.287748  192.168.1.88     104.19.173.68    TCP       66 59280 → 443 [ACK] Seq=1 Ack=140 Win=2048
 305 29.306224  Verizon_c7:ef:10 Apple_53:82:2f   ARP       42 Who has 192.168.1.88? Tell 192.168.1.1
 306 29.306291  Apple_53:82:2f   Verizon_c7:ef:10 ARP       42 192.168.1.88 is at a8:8f:d9:53:82:2f
```

```
> Frame 291: 162 bytes on wire (1296 bits), 162 bytes capture
> Ethernet II, Src: Verizon_c7:ef:10 (48:5d:36:c7:ef:10), Dst
> Internet Protocol Version 4, Src: 142.250.80.110, Dst: 192.
> User Datagram Protocol, Src Port: 443, Dst Port: 55346
> QUIC IETF
```

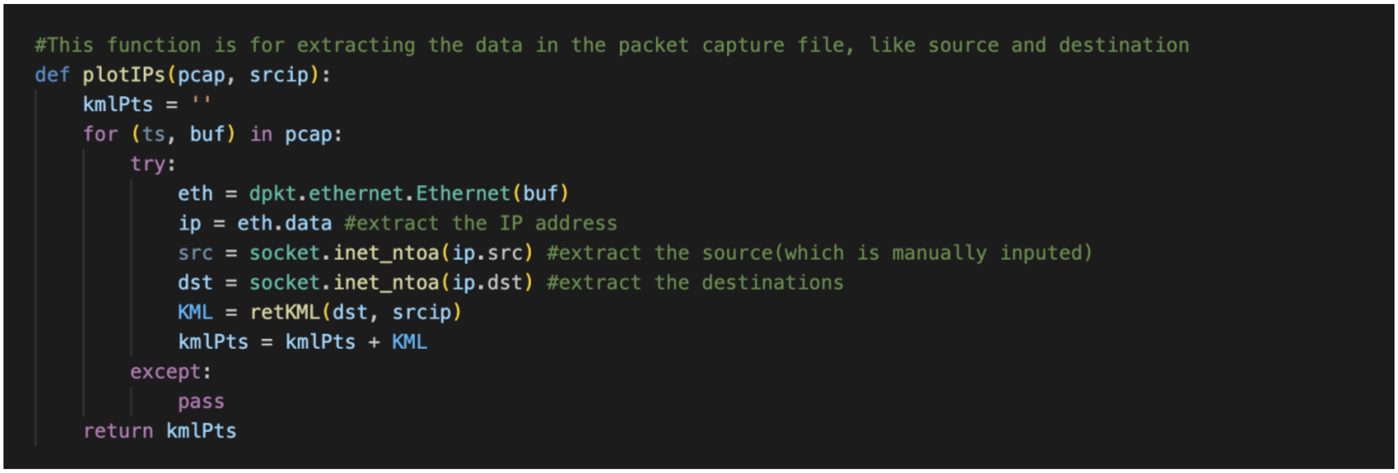
```
0000  a8 8f d9 53 82 2f 48 5d  36 c7 ef 10 08 00 45 00   ···S
0010  00 94 00 00 40 00 3c 11  9c f0 8e fa 50 6e c0 a8   ····
0020  01 58 01 bb d8 32 00 80  f8 b3 55 6b 59 0c 01 b8   ·X··
0030  91 f4 b9 7f 19 ab 53 d1  c1 cd f1 37 b3 41 47 cc   ····
0040  43 f7 65 b0 ab ec 71 97  c7 9e c4 17 96 b0 bb 6f   C·e·
0050  cd 99 bd e3 57 42 c2 7b  76 b9 d6 b6 c3 f2 7f fc   ····
0060  0f 5c 81 e6 88 9e 62 00  b6 e8 c9 94 7a 97 5d 13   ·\··
0070  4e 8f 07 0d a1 cd 1e b3  e0 df f7 f3 a9 c4 2b 87   N···
0080  e6 ec 21 3b 55 f8 26 27  1d 87 dc 2e 33 c5 54 f7   ··!;
0090  38 45 f2 bd 6f 86 0c 6b  4a 56 a2 e2 5c 64 73 d0   8E··
00a0  94 ae                                              ··
```

It orchestrates the execution of the script by opening the packet capture file, generating KML header, calling `plotIPs` function to process the data.



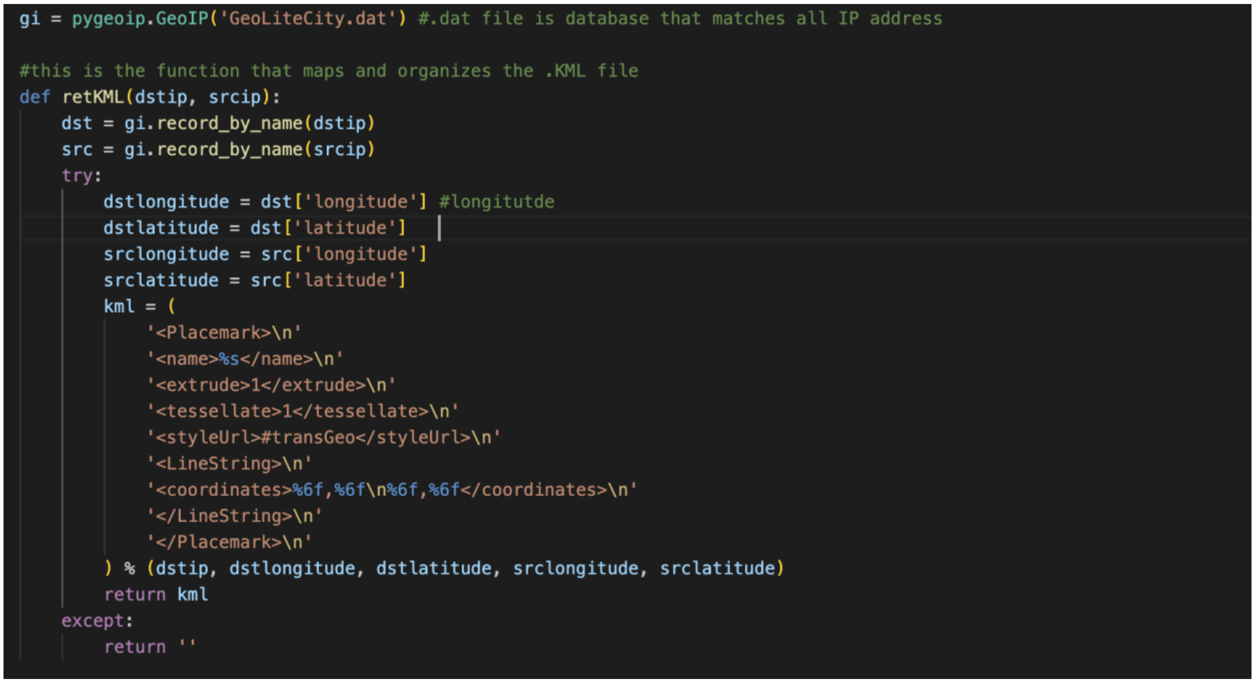
```python
#main function
def main():
    f = open('packet.pcap', 'rb') #allows the file to be read in binary format
    pcap = dpkt.pcap.Reader(f)

    #this formating the KML Files: Which are a a format used to display geographical data; color, witdth
    kmlheader = '<?xml version="1.0" encoding="UTF-8"?> \n<kml xmlns="http://www.opengis.net/kml/2.2">\n<Document>\n' \
                '<Style id="transGeo">' \
                '<LineStyle>' \
                '<width>1.5</width>' \
                '<color>501400E6</color>' \
                '</LineStyle>' \
                '</Style>'
    kmlfooter = '</Document>\n</kml>\n'
```

It enables the visualization of geolocated IPs by processing packet capture data and converting IP addresses to geographical coordinates.



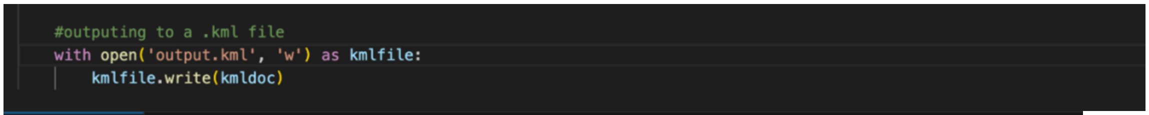
```python
#This function is for extracting the data in the packet capture file, like source and destination
def plotIPs(pcap, srcip):
    kmlPts = ''
    for (ts, buf) in pcap:
        try:
            eth = dpkt.ethernet.Ethernet(buf)
            ip = eth.data #extract the IP address
            src = socket.inet_ntoa(ip.src) #extract the source(which is manually inputed)
            dst = socket.inet_ntoa(ip.dst) #extract the destinations
            KML = retKML(dst, srcip)
            kmlPts = kmlPts + KML
        except:
            pass
    return kmlPts
```

It retrieves latitude and longitude coordinates for the given source and destination IP addresses and generates a KML string that represents a Placemark in the KML format, which is used for geospatial data visualization.

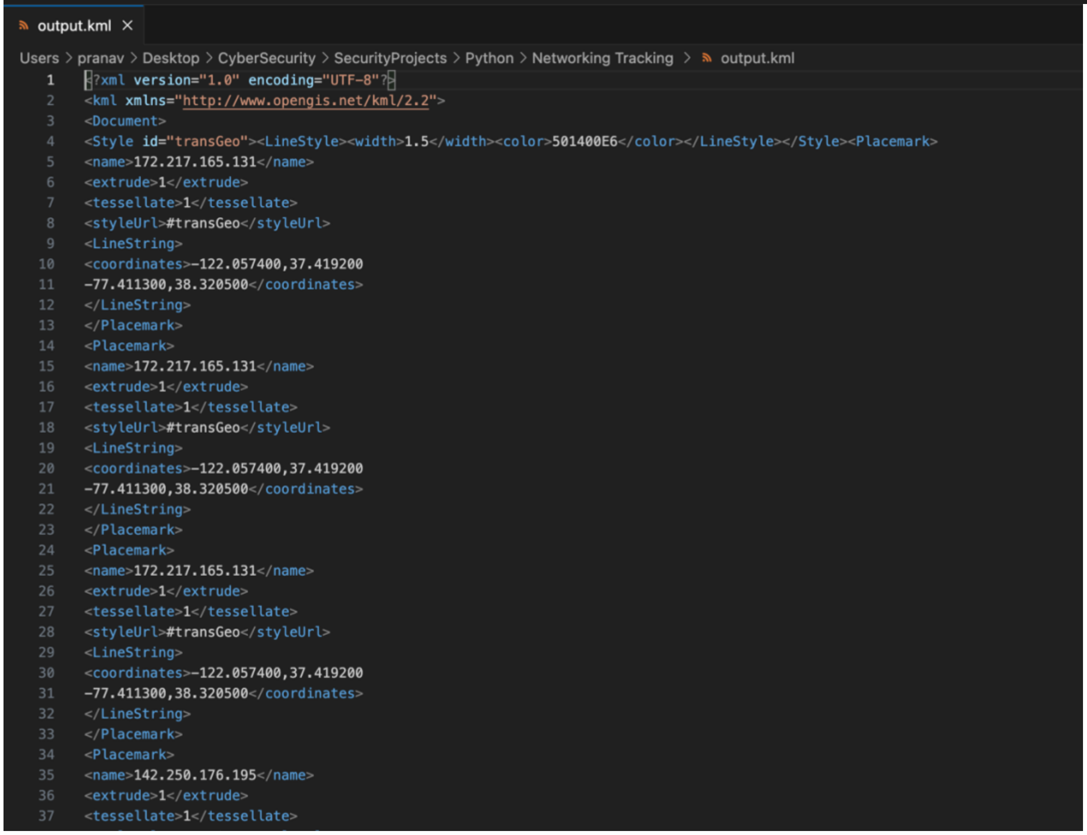


```python
gi = pygeoip.GeoIP('GeoLiteCity.dat') #.dat file is database that matches all IP address

#this is the function that maps and organizes the .KML file
def retKML(dstip, srcip):
    dst = gi.record_by_name(dstip)
    src = gi.record_by_name(srcip)
    try:
        dstlongitude = dst['longitude'] #longitutde
        dstlatitude = dst['latitude']
        srclongitude = src['longitude']
        srclatitude = src['latitude']
        kml = (
            '<Placemark>\n'
            '<name>%s</name>\n'
            '<extrude>1</extrude>\n'
            '<tessellate>1</tessellate>\n'
            '<styleUrl>#transGeo</styleUrl>\n'
            '<LineString>\n'
            '<coordinates>%6f,%6f\n%6f,%6f</coordinates>\n'
            '</LineString>\n'
            '</Placemark>\n'
        ) % (dstip, dstlongitude, dstlatitude, srclongitude, srclatitude)
        return kml
    except:
        return ''
```

After the conducting the function the last line in the main function creates an output.KML file



```
#outputing to a .kml file
with open('output.kml', 'w') as kmlfile:
    kmlfile.write(kmldoc)
```

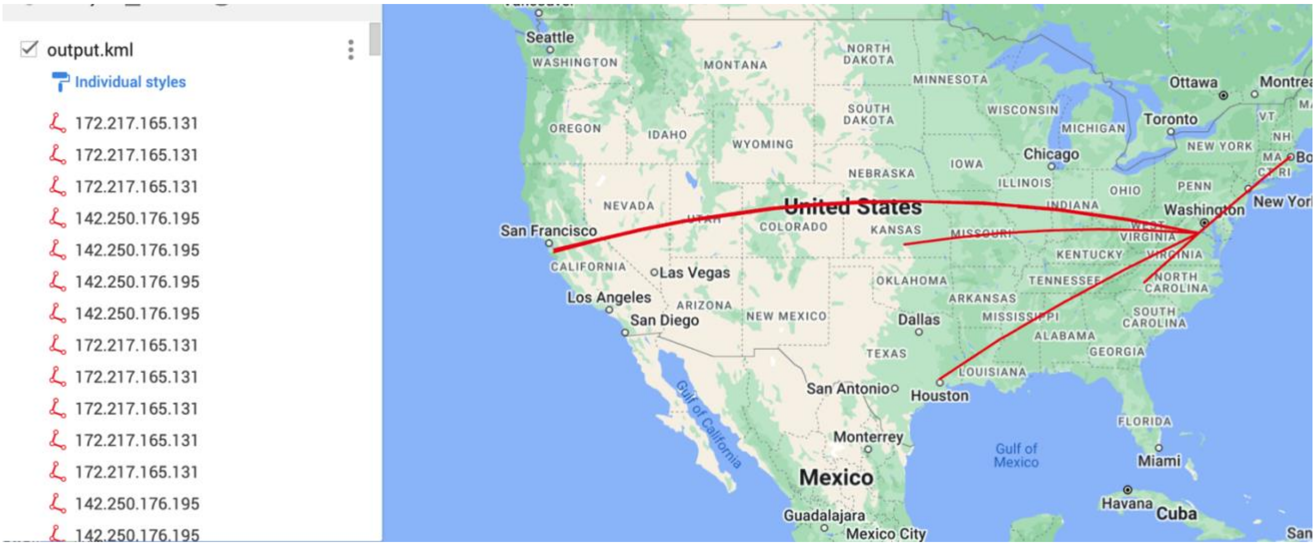


output.kml ×

Users > pranav > Desktop > CyberSecurity > SecurityProjects > Python > Networking Tracking > output.kml

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Style id="transGeo"><LineStyle><width>1.5</width><color>501400E6</color></LineStyle></Style><Placemark>
<name>172.217.165.131</name>
<extrude>1</extrude>
<tessellate>1</tessellate>
<styleUrl>#transGeo</styleUrl>
<LineString>
<coordinates>-122.057400,37.419200
-77.411300,38.320500</coordinates>
</LineString>
</Placemark>
<Placemark>
<name>172.217.165.131</name>
<extrude>1</extrude>
<tessellate>1</tessellate>
<styleUrl>#transGeo</styleUrl>
<LineString>
<coordinates>-122.057400,37.419200
-77.411300,38.320500</coordinates>
</LineString>
</Placemark>
<Placemark>
<name>172.217.165.131</name>
<extrude>1</extrude>
<tessellate>1</tessellate>
<styleUrl>#transGeo</styleUrl>
<LineString>
<coordinates>-122.057400,37.419200
-77.411300,38.320500</coordinates>
</LineString>
</Placemark>
<Placemark>
<name>142.250.176.195</name>
<extrude>1</extrude>
<tessellate>1</tessellate>
```

Insert the .KML file to Google Maps to get a visual representation of



☑ output.kml

Individual styles

- 172.217.165.131
- 172.217.165.131
- 172.217.165.131
- 142.250.176.195
- 142.250.176.195
- 142.250.176.195
- 142.250.176.195
- 172.217.165.131
- 172.217.165.131
- 172.217.165.131
- 172.217.165.131
- 172.217.165.131
- 142.250.176.195
- 142.250.176.195