```
!pip install scapy
!pip install seaborn

from scapy.all import *
import pandas as pd
import numpy as np
import binascii
import seaborn as sns
sns.set(color_codes=True)
%matplotlib inline

'''Use common fields in IP Packet to perform exploratory analysis on PCAP'''
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wh
Collecting scapy
  Downloading scapy-2.5.0.tar.gz (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 26.4 MB/s eta 0
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: scapy
  Building wheel for scapy (setup.py) ... done
  Created wheel for scapy: filename=scapy-2.5.0-py2.py3-none-any.whl size=14443
  Stored in directory: /root/.cache/pip/wheels/98/ea/08/164e840ab2c83b892bf8b19
Successfully built scapy
Installing collected packages: scapy
Successfully installed scapy-2.5.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wh
Requirement already satisfied: seaborn in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.8/dist-pack
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.8/dist
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/di
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packag
'Use common fields in IP Packet to perform exploratory analysis on PCAP'
```

```
num_of_packets_to_sniff = 100
pcap = sniff(count=num_of_packets_to_sniff)

# rdpcap returns packet list
## packetlist object can be enumerated
print(type(pcap))
print(len(pcap))
print(pcap)
pcap[0]
```

```
<class 'scapy.plist.PacketList'>
100
<Sniffed: TCP:100 UDP:0 ICMP:0 Other:0>
```

```
<Ether  dst=02:42:ac:1c:00:0c src=02:42:8b:97:1c:66 type=IPv4 |<IP  version=4
ihl=5 tos=0x0 len=52 id=18691 flags=DF frag=0 ttl=64 proto=6 chksum=0x997b
src=172.28.0.1 dst=172.28.0.12 |<TCP  sport=55372 dport=8080 seq=52080829
ack=971127689 dataofs=8 reserved=0 flags=A window=501 chksum=0x586c urgptr=0
options=[('NOP', None), ('NOP', None), ('Timestamp', (1563912288, 4279191969))]
|>>>
```

```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving suspicious.pcap to suspicious.pcap
```

```
# rdpcap used to Read Pcap
pcap = pcap + rdpcap("suspicious.pcap")
pcap
```

```
<Sniffed+suspicious.pcap: TCP:100 UDP:62 ICMP:0 Other:0>
```

```
# ETHERNET -> Internet Protocol -> Layer 4 Segments
# We're only interested in Layers 3 (IP) and 4 (TCP AND UDP)
## We'll parse those two layers and the layer 4 payload
## When capturing we capture layer 2 frames and beyond

# Retrieving a single item from packet list
ethernet_frame = pcap[101]
ip_packet = ethernet_frame.payload
segment = ip_packet.payload
data = segment.payload # Retrieve payload that comes after layer 4

# Observe that we just popped off previous layer header
print(ethernet_frame.summary())
print(ip_packet.summary())
print(segment.summary())
print(data.summary()) # If blank, empty object

# Complete depiction of paket
## Achieving understanding that these are the fields will enable the ability
## to ask the data more meaningful questions ie) type of layer 4 segment is defined in lay
ethernet_frame.show()
```

```
Ether / IP / UDP / DNS Ans "2607:f8b0:4005:807::200e"
IP / UDP / DNS Ans "2607:f8b0:4005:807::200e"
UDP / DNS Ans "2607:f8b0:4005:807::200e"
DNS Ans "2607:f8b0:4005:807::200e"
###[ Ethernet ]###
  dst       = 88:e9:fe:6a:92:52
  src       = 80:37:73:96:9b:db
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x20
     len       = 84
```

```
          id        = 58919
          flags     = 
          frag      = 0
          ttl       = 122
          proto     = 17
          chksum    = 0x360c
          src       = 84.54.22.33
          dst       = 10.1.10.53
          \options   \
     ###[ UDP ]###
             sport     = 53
             dport     = 53
             len       = 64
             chksum    = 0xfe25
     ###[ DNS ]###
                id        = 12
                qr        = 1
                opcode    = QUERY
                aa        = 0
                tc        = 0
                rd        = 1
                ra        = 1
                z         = 0
                ad        = 0
                cd        = 0
                rcode     = ok
                qdcount   = 1
                ancount   = 1
                nscount   = 0
                arcount   = 0
                \qd        \
                 |###[ DNS Question Record ]###
                 |  qname     = 'google.com.'
                 |  qtype     = AAAA
                 |  qclass    = IN
                \an        \
                 |###[ DNS Resource Record ]###
                 |  rrname    = 'google.com.'
                 |  type      = AAAA
                 |  rclass    = IN
                 |  ttl       = 299
                 |  rdlen     = 16
                 |  rdata     = 2607:f8b0:4005:807::200e
                ns        = None
                ar        = None
```

```
# Understanding the object types in scapy
print(type(ethernet_frame))
print(type(ip_packet))
print(type(segment))

# Packets can be filtered on layers ie) ethernet_frame[scapy.layers.l2.Ether]
ethernet_type = type(ethernet_frame)
ip_type = type(ip_packet)
tcp_type = type(segment)
print("Ethernet",pcap[ethernet_type])
print("IP", pcap[ip_type])
print("TCP", pcap[tcp_type])

# Scapy provides this via import statements
from scapy.layers.l2 import Ether
from scapy.layers.inet import IP
from scapy.layers.inet import TCP, UDP

print("UDP", pcap[UDP])
```

```
<class 'scapy.layers.l2.Ether'>
<class 'scapy.layers.inet.IP'>
<class 'scapy.layers.inet.UDP'>
Ethernet <Ether from Sniffed+suspicious.pcap: TCP:100 UDP:62 ICMP:0 Other:0>
IP <IP from Sniffed+suspicious.pcap: TCP:100 UDP:62 ICMP:0 Other:0>
TCP <UDP from Sniffed+suspicious.pcap: TCP:0 UDP:62 ICMP:0 Other:0>
UDP <UDP from Sniffed+suspicious.pcap: TCP:0 UDP:62 ICMP:0 Other:0>
```

```
# Collect field names from IP/TCP/UDP (These will be columns in DF)
ip_fields = [field.name for field in IP().fields_desc]
tcp_fields = [field.name for field in TCP().fields_desc]
udp_fields = [field.name for field in UDP().fields_desc]

dataframe_fields = ip_fields + ['time'] + tcp_fields + ['payload','payload_raw','payload_h

# Create blank DataFrame
df = pd.DataFrame(columns=dataframe_fields)
for packet in pcap[IP]:
    # Field array for each row of DataFrame
    field_values = []
    # Add all IP fields to dataframe
    for field in ip_fields:
        if field == 'options':
            # Retrieving number of options defined in IP Header
            field_values.append(len(packet[IP].fields[field]))
        else:
            field_values.append(packet[IP].fields[field])

    field_values.append(packet.time)

    layer_type = type(packet[IP].payload)
    for field in tcp_fields:
        try:
            if field == 'options':
                field_values.append(len(packet[layer_type].fields[field]))
            else:
                field_values.append(packet[layer_type].fields[field])
        except:
            field_values.append(None)

    # Append payload
    field_values.append(len(packet[layer_type].payload))
    field_values.append(packet[layer_type].payload.original)
    field_values.append(binascii.hexlify(packet[layer_type].payload.original))
    # Add row to DF
    df_append = pd.DataFrame([field_values], columns=dataframe_fields)
    df = pd.concat([df, df_append], axis=0)

# Reset Index
df = df.reset_index()
```

```
# Drop old index column
df = df.drop(columns="index")


# Retrieve first row from DataFrame
print(df.iloc[0])

print(df.shape)

# Return first 5 rows
df.head()

# Return last 5 rows
df.tail()

# Return the Source Address for all rows
df['src']

# Return Src Address, Dst Address, Src Port, Dst Port
df[['src','dst','sport','dport']]
```

```
version                        4
ihl                            5
tos                            0
len                           52
id                         18691
flags                         DF
frag                           0
ttl                           64
proto                          6
chksum                     39291
src                   172.28.0.1
dst                  172.28.0.12
options                        0
time           1674158586.494406
sport                      55372
dport                       8080
seq                     52080829
```

```
# Top Source Adddress
print("# Top Source Address")
print(df['src'].describe(),'\n\n')

# Top Destination Address
print("# Top Destination Address")
print(df['dst'].describe(),"\n\n")

frequent_address = df['src'].describe()['top']

# Who is the top address speaking to
print("# Who is Top Address Speaking to?")
print(df[df['src'] == frequent_address]['dst'].unique(),"\n\n")

# Who is the top address speaking to (dst ports)
print("# Who is the top address speaking to (Destination Ports)")
print(df[df['src'] == frequent_address]['dport'].unique(),"\n\n")

# Who is the top address speaking to (src ports)
print("# Who is the top address speaking to (Source Ports)")
print(df[df['src'] == frequent_address]['sport'].unique(),"\n\n")
```

```
# Top Source Address
count            162
unique             5
top       172.28.0.1
freq              55
Name: src, dtype: object


# Top Destination Address
count             162
unique              5
top       172.28.0.12
freq               55
Name: dst, dtype: object


# Who is Top Address Speaking to?
```

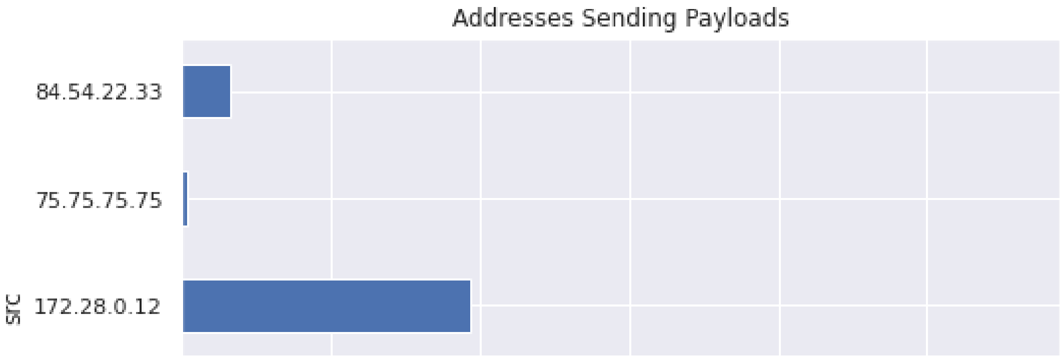
```
['172.28.0.12']


# Who is the top address speaking to (Destination Ports)
[8080 6000]


# Who is the top address speaking to (Source Ports)
[55372 43518 43530 59126 58766]
```

```
# Unique Source Addresses
print("Unique Source Addresses")
print(df['src'].unique())

print()

# Unique Destination Addresses
print("Unique Destination Addresses")
print(df['dst'].unique())
```

```
Unique Source Addresses
['172.28.0.1' '172.28.0.12' '10.1.10.53' '84.54.22.33' '75.75.75.75']

Unique Destination Addresses
['172.28.0.12' '172.28.0.1' '84.54.22.33' '10.1.10.53' '75.75.75.75']
```

```
# Group by Source Address and Payload Sum
source_addresses = df.groupby("src")['payload'].sum()
source_addresses.plot(kind='barh',title="Addresses Sending Payloads",figsize=(8,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93ef4e7430>
```


Addresses Sending Payloads
src