# Analyze Network Packets with Python

**Bin Xiang**

May 16, 2019

## 1 Introduction

### 1.1 Network packet - PCAP

PCAP (packet capture) is a type of file containing packet data of a network.

*Check this link:* PCAP Next Generation Dump File Format - PCAP-DumpFileFormat2

### 1.2 Analyzing tool

- **Scapy** > is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark).
- **Wireshark** > is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998.

## 2 Obtain network packets

There are many ways to obtain packets. Here are several nice options:

- **Wireshark**
- **Tcpdump**
- **Scapy**

where **wireshark** has GUI support. In the following, since we adopt **Scapy** to parse pcap file, we also use it to capture network packets.

### 2.1 Capture packets

- To sniff packets, you need **root privileges**, otherwise, you will get the "*Operation not permitted*" errors.

- To skip this problem, you can uncomment the following commands and run them in a **scapy's interactive shell** with root privileges.

[1]:
```
from scapy.all import * # Packet manipulation
```

[2]:
```
#pkts = sniff(count=10)       # sniff 10 packets
#if pkts:                     # if pkts exist
#    wrpcap("temp.pcap",pkts) # save captured packets to a pcap file
```

### 2.2 Read packets

[3]:
```
pkts = rdpcap("temp.pcap") # read and parse packets into memory
```

- **OR**

[4]:
```
pkts = sniff(offline="temp.pcap", count=10) # sniff in a offline way from a pcap␣
↪file
```

Notice that, this can be very useful when you try to load a large size pcap file.

### 2.3 Filter packets

sniff() uses **Berkeley Packet Filter (BPF)** syntax (the same one as tcpdump).

Check this link for more details about the syntax.

[5]:
```
pkts_filtered = sniff(offline="temp.pcap", filter="tcp", prn=lambda x:x.
↪summary()) # filter all TCP packets
```

```
Ether / IP / TCP 10.169.226.59:36112 > 216.58.205.132:https A
Ether / IP / TCP 216.58.205.132:https > 10.169.226.59:36112 A
Ether / IP / TCP 10.169.226.59:47924 > 172.217.23.99:https PA / Raw
Ether / IP / TCP 172.217.23.99:https > 10.169.226.59:47924 A
Ether / IP / TCP 172.217.23.99:https > 10.169.226.59:47924 PA / Raw
Ether / IP / TCP 172.217.23.99:https > 10.169.226.59:47924 PA / Raw
```

**prn**: function to apply to each packet. If something is returned, it is displayed.

[6]:
```
print(pkts) # print a overview of pkts
print(pkts_filtered)
```

```
<Sniffed: TCP:6 UDP:1 ICMP:0 Other:3>
<Sniffed: TCP:6 UDP:0 ICMP:0 Other:0>
```

### 2.4 Show packets details

[7]:
```
print(pkts[0].summary()) # summary of the first packet
```

```
Ether / IP / TCP 10.169.226.59:36112 > 216.58.205.132:https A
```

[8]:
```
pkts[0].payload.show() # show the payload details of the first packet
```

```
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 52
  id        = 7371
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = 0x8b55
  src       = 10.169.226.59
  dst       = 216.58.205.132
  \options   \
###[ TCP ]###
     sport     = 36112
     dport     = https
     seq       = 3292936675
     ack       = 1115597772
     dataofs   = 8
     reserved  = 0
     flags     = A
     window    = 501
     chksum    = 0xc6cd
     urgptr    = 0
     options   = [('NOP', None), ('NOP', None), ('Timestamp', (1426347376,
3775730836))]
```

# 3 Transform pcap to DataFrame

## 3.1 Retrieve layers in packet

[9]:
```
from scapy.layers.l2 import Ether
from scapy.layers.inet import IP
from scapy.layers.inet import TCP, UDP

print(pkts[IP]) # a overview of all IP packets
```

```
<IP from Sniffed: TCP:6 UDP:1 ICMP:0 Other:0>
```

[10]:
```
# Store the pre-defined fields name in IP, TCP layers
f_ip = [field.name for field in IP().fields_desc]
f_tcp = [field.name for field in TCP().fields_desc]
print(f_ip)  # field name of IP Layer
print(f_tcp) # field name of TCP Layer

f_all = f_ip + ['time'] + f_tcp + ['payload']
```

```
['version', 'ihl', 'tos', 'len', 'id', 'flags', 'frag', 'ttl', 'proto',
'chksum', 'src', 'dst', 'options']
['sport', 'dport', 'seq', 'ack', 'dataofs', 'reserved', 'flags', 'window',
'chksum', 'urgptr', 'options']
```

[11]:
```
# Data structures and data analysis
import pandas as pd

# Blank DataFrame
df = pd.DataFrame(columns=f_all)
for packet in pkts[IP]:
     # store data for each row of DataFrame
    field_values = []

    # Read values of IP fields
    for field in f_ip:
        if field == 'options':
            # we only store the number of options defined in IP Header
            field_values.append(len(packet[IP].fields[field]))
        else:
            field_values.append(packet[IP].fields[field])

    # Read values of Time
    field_values.append(packet.time)

    # Read values of TCP fields
    layer_type = type(packet[IP].payload)
    for field in f_tcp:
        try:
            if field == 'options':
                field_values.append(len(packet[layer_type].fields[field]))
            else:
                field_values.append(packet[layer_type].fields[field])
        except:
            # the field value may not exist
            field_values.append(None)

    # Read values of Payload
    field_values.append(len(packet[layer_type].payload))
```

```
    # Fill the data of one row
    df_append = pd.DataFrame([field_values], columns=f_all)
    # Append row in df
    df = pd.concat([df, df_append], axis=0)

# Reset index
df = df.reset_index()
df = df.drop(columns="index")

# shape
print("Shape: ", df.shape, '\n')
# first row
print(df.iloc[0], '\n')
# table with specified fields
df[['time', 'src', 'dst', 'sport', 'dport']]
```

```
Shape:  (7, 26)

version                 4
ihl                     5
tos                     0
len                    52
id                   7371
flags                  DF
frag                    0
ttl                    64
proto                   6
chksum              35669
src          10.169.226.59
dst         216.58.205.132
options                 0
time          1.55743e+09
sport               36112
dport                 443
seq            3292936675
ack            1115597772
dataofs                 8
reserved                0
flags                   A
window                501
chksum              50893
urgptr                  0
options                 3
payload                 0
Name: 0, dtype: object
```

[11]:

| | time | src | dst | sport | dport |
|---|---|---|---|---|---|
| 0 | 1.557433e+09 | 10.169.226.59 | 216.58.205.132 | 36112 | 443 |
| 1 | 1.557433e+09 | 216.58.205.132 | 10.169.226.59 | 443 | 36112 |
| 2 | 1.557433e+09 | 10.169.227.252 | 224.0.0.2 | 1985 | 1985 |
| 3 | 1.557433e+09 | 10.169.226.59 | 172.217.23.99 | 47924 | 443 |
| 4 | 1.557433e+09 | 172.217.23.99 | 10.169.226.59 | 443 | 47924 |
| 5 | 1.557433e+09 | 172.217.23.99 | 10.169.226.59 | 443 | 47924 |
| 6 | 1.557433e+09 | 172.217.23.99 | 10.169.226.59 | 443 | 47924 |

## 4 Statistics

[12]:
```
print(df['src'].describe(), '\n')  # show description of the source addresses
print(df['src'].describe()['top']) # top ip address
print(df['src'].unique())          # unique address
```

```
count                 7
unique                4
top       172.217.23.99
freq                  3
Name: src, dtype: object

172.217.23.99
['10.169.226.59' '216.58.205.132' '10.169.227.252' '172.217.23.99']
```

[13]:
```
src_addr = df.groupby("src")['payload'].sum() # show the sum of payload for each␣
 ↪src ip
src_addr.plot(kind='barh', figsize=(8,2))     # plot figure
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f311580b8>
```