Analyzing Packet Captures with Python

Source

> Analyzing Packet Captures with Python
> Why would you use Python to read a pcap?
> https://vnetman.github.io/pcap/python/pyshark/scapy/libpcap/2018/10/25/analyzing-packet-captures-with-python-part-1.html

Programmatically processing packet capture (pcap) files.

Why?

- Find the first slow connection to a specific server
- Identify prematurely terminated connections caused by client
- Investigate network related slowness
- Repeating the above regulary

Language and Modules

- python3
- scapy — allows you to craft, send, recieve, and manipulate network packets at different layers of the network protocol stack

Step 1: Program Skeleton

Make sure I can open pcap file

```
import argparse #get pcap file name from command line
import os
import sys

def pcapProcess(fileName):
    print('Opening {}...'.format(fileName))

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description = 'PCAP reader')
    parser.add_argument('--pcap', metavar='<pcap file name>',
                        help = 'pcap file to parse', required=True)
    args = parser.parse_args()

    fileName = args.pcap
    if not os.path.isfile(fileName):
        print('"{}" does not exist'.format(fileName), file=sys.stderr)
        sys.exit(-1)

    pcapProcess(fileName)
    sys.exit(0)
```

1. Takes in a PCAP file and prints out a message indictating it is opening that file
2. *argparse.ArgumentParser* creates a command line argument parser
3. The parser is configured with arguments *—pcap* and the pcap file ensuring that they are required
4. The file is extracted from the parsed arguments
5. If the file exists, pcapProcess is called and program exits.

```
(base) annalisaallan@Annalisas-MacBook-Pro pcap % python3 pcap_an
alyze.py --pcap example-01.pcap
Opening example-01.pcap...
```

Step 2: Basic PCAP Handling

Open pcap and count how many packets it has

```
from scapy.utils import RawPcapReader

def pcapProcess(fileName):
    print('Opening {}...'.format(fileName))

    count = 0
    for (pkt_data, pkt_metadata,) in RawPcapReader(fileName):
        count += 1

    print('{} contains {} packets'.format(fileName, count))
```

1. *count* keeps track of the number of packets in the PCAP file
2. *(pkt_data, pkt_metadata,) in RawPcapReader(fileName)* reads the packet's data and metadata. The loop processes each packet one by one

```
(base) annalisaallan@Annalisas-MacBook-Pro pcap % python3 pcap_analyze.py --pcap example-01.pcap
Opening example-01.pcap...
example-01.pcap contains 22639 packets
```

Step 3: Filter non IPv4/TCP packets

Filter out uninteresting packets

```
from scapy.utils import RawPcapReader
from scapy.layers.l2 import Ether
from scapy.layers.inet import IP, TCP

def pcapProcess(fileName):
    print('Opening {}...'.format(fileName))

    count = 0
    interestingPacketsCount = 0

    for (pkt_data, pkt_metadata,) in RawPcapReader(fileName):
        count += 1

        ether_pkt = Ether(pkt_data)
        if 'type' not in ether_pkt.fields:
            #LLC frames has 'len' instead of 'type' so those are disregarded
            continue

        if ether_pkt.type != 0x0800:
            #disregard non-IPv4 packets
            continue

        ip_pkt = ether_pkt[IP] #obtain IPv4 header
        if ip_pkt.proto != 6:
            #disregard non-TCP packets
            continue

        interestingPacketsCount += 1

    print('{} contains {} packets ({} interesting)'.format(fileName, count, interestingPacketsCount))
```

1. interestingPacketsCount is initialized and keeps track of the number of IPv4 and TCP packets
2. Create an ethernet packet using the packet data
3. LLC frames are disregarded as they don't have a *type* field
4. Extract the IPv4 header from the Ethernet packet if it is an IPv4 packet
5. Check is packet is a TCP packet

```
(base) annalisaallan@Annalisas-MacBook-Pro pcap % python3 pcap_analyze.py --pcap example-01.pcap
Opening example-01.pcap...
example-01.pcap contains 22639 packets (22639 interesting)
```

Step 4: Identifying interesting connection packets

Consider a connection between a specific client and a specific server

Note: The tutorial hardcoded the client and server IP addresses but later on I use argparse to gather the info from the command line in Step 5

```
def pcapProcess(fileName):
    print('Opening {}...'.format(fileName))

    client = '192.168.1.137:57080'
    server = '152.19.134.43:80'

    (client_ip, client_port) = client.split(':')
    (server_ip, server_port) = server.split(':')

    count = 0
    interestingPacketsCount = 0

    for (pkt_data, pkt_metadata,) in RawPcapReader(fileName):
        count += 1

        ether_pkt = Ether(pkt_data)
        if 'type' not in ether_pkt.fields:
            #LLC frames has 'len' instead of 'type' so those are disregarded
            continue

        if ether_pkt.type != 0x0800:
            #disregard non-IPv4 packets
            continue

        ip_pkt = ether_pkt[IP] #obtain IPv4 header
        if ip_pkt.proto != 6:
            #disregard non-TCP packets
            continue

        if (ip_pkt.src != server_ip) and (ip_pkt.src != client_ip):
            #uninteresting source IP address
            continue

        if(ip_pkt.dst != server_ip) and (ip_pkt.dst != client_ip):
            #uninteresting destination IP address
            continue

        tcp_pkt = ip_pkt[TCP]

        if(tcp_pkt.sport != int(server_port)) and (tcp_pkt.sport != int(client_port)):
            #uninteresting source TCP port
            continue

        if(tcp_pkt.dport != int(server_port)) and (tcp_pkt.dport != int(client_port)):
            #uninteresting destination TCP port
            continue

        interestingPacketsCount += 1

    print('{} contains {} packets ({} interesting)'.format(fileName, count, interestingPacketsCount))
```

1. Check if source IP address of the packet is either the client IP or the server IP
2. Check if the destination IP address of the packet is either the client IP or the server IP
3. Extract TCP header from IP packet
4. Check if the source TCP port of the packet is either the client port or the server port
5. Check if the destination TCP port of the packet is either the client port or the server port

```
(base) annalisaallan@Annalisas-MacBook-Pro pcap % python3 pcap_analyze.py --pcap example-01.pcap
Opening example-01.pcap...
example-01.pcap contains 22639 packets (14975 interesting)
```

Step 5: Packet metadata

Accessing the packet's metadata — timestamps and ordinal numbers of the first and last packets of the connection we're interested in.

1. Create a printable time stamp function that takes a timestamp and a resolution and converts the timestamp into a human readable string representation
2. The first interesting packet is looked at and the higher 32 and lower 32 bits of the timestamp from the metadata is used to calculate the timestamp
3. The last interesting packet is looked at and the same thing occurs as above for the first interesting packet

```
import time

def printable_timestamp(timestamp, resol):
    ts_sec = timestamp // resol
    ts_subsec = timestamp % resol
    ts_sec_str = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(ts_sec))
    return '{}.{}'.format(ts_sec_str, ts_subsec)

def pcapProcess(fileName):
    print('Opening {}...'.format(fileName))

    #client = '192.168.1.137:57080'
    #server = '152.19.134.43:80'

    (client_ip, client_port) = client.split(':')
    (server_ip, server_port) = server.split(':')

    count = 0
    interestingPacketsCount = 0

    for (pkt_data, pkt_metadata,) in RawPcapReader(fileName):
        count += 1

        ether_pkt = Ether(pkt_data)
        if 'type' not in ether_pkt.fields:
            #LLC frames has 'len' instead of 'type' so those are disregarded
            continue

        if ether_pkt.type != 0x0800:
            #disregard non-IPv4 packets
            continue

        ip_pkt = ether_pkt[IP] #obtain IPv4 header
        if ip_pkt.proto != 6:
            #disregard non-TCP packets
            continue

        if (ip_pkt.src != server_ip) and (ip_pkt.src != client_ip):
            #uninteresting source IP address
            continue

        if(ip_pkt.dst != server_ip) and (ip_pkt.dst != client_ip):
            #uninteresting destination IP address
            continue
```

```
        tcp_pkt = ip_pkt[TCP]

        if(tcp_pkt.sport != int(server_port)) and (tcp_pkt.sport != int(client_port)):
            #uninteresting source TCP port
            continue

        if(tcp_pkt.dport != int(server_port)) and (tcp_pkt.dport != int(client_port)):
            #uninteresting destination TCP port
            continue

        interestingPacketsCount += 1
        if interestingPacketsCount == 1:
            first_pkt_timestamp = (pkt_metadata.tshigh << 32) | pkt_metadata.tslow
            first_pkt_timestamp_resolution = pkt_metadata.tsresol
            first_pkt_ordinal = count

        last_pkt_timestamp = (pkt_metadata.tshigh << 32) | pkt_metadata.tslow
        last_pkt_timestamp_resolution = pkt_metadata.tsresol
        last_pkt_ordinal = count

    print('{} contains {} packets ({} interesting)'.format(fileName, count, interestingPacketsCount))
    print('First packet in connection: Packet #{} {}'.format(first_pkt_ordinal, printable_timestamp(first_pkt_timestamp, first_pkt_timestam
    print('Last packet in connection: Packet #{} {}'.format(last_pkt_ordinal, printable_timestamp(last_pkt_timestamp, last_pkt_timestamp_re

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description = 'PCAP reader')
    parser.add_argument('--pcap', metavar='<pcap file name>',
                        help = 'pcap file to parse', required=True)
    parser.add_argument('clientAddr', help='Specify the client pcap file name')
    parser.add_argument('serverAddr', help='Specify the server pcap file name')
    args = parser.parse_args()

    fileName = args.pcap
    if not os.path.isfile(fileName):
        print('"{}" does not exist'.format(fileName), file=sys.stderr)
        sys.exit(-1)

    client = args.clientAddr
    server = args.serverAddr

    pcapProcess(fileName)
    sys.exit(0)
```

```
(base) annalisaallan@Annalisas-MacBook-Pro pcap % python3 pcap_analyze.py --pcap example-01.pcap 192.168.1.137:57080 152.19.134.43:80
Opening example-01.pcap...
example-01.pcap contains 22639 packets (14975 interesting)
First packet in connection: Packet #2585 2018-09-26 11:51:02.883718124
Last packet in connection: Packet #22582 2018-09-26 11:52:04.324012912
```

Step 6: Relative timestamps, relative sequence numbers, TCP flags

—> indicates packets sent from client to server

<— indicates packets sent from server to client

[####] indicates the packet ordinals

0.######s is the timestamp

TCP flags are indicated

1. Define an enumeration class
2. Determine the packet direction whether client to server or server to client
3. Process the interesting packets of timestamps and offsets
4. Extract the TCP payload information
5. Format the packet information

```
from enum import Enum

class PktDirection(Enum):
    not_defined = 0
    clientServer = 1
    serverClient = 2

def pcapProcess(fileName):
    print('Opening {}...'.format(fileName))

    # client = '192.168.1.137:57080'
    # server = '152.19.134.43:80'


    (client_ip, client_port) = client.split(':')
    (server_ip, server_port) = server.split(':')

    count = 0
    interestingPacketsCount = 0

    server_seq_offset = None
    client_seq_offset = None

    for (pkt_data, pkt_metadata,) in RawPcapReader(fileName):
        count += 1

        ether_pkt = Ether(pkt_data)
        if 'type' not in ether_pkt.fields:
            #LLC frames has 'len' instead of 'type' so those are disregarded
            continue

        if ether_pkt.type != 0x0800:
            #disregard non-IPv4 packets
            continue

        ip_pkt = ether_pkt[IP] #obtain IPv4 header
        if ip_pkt.proto != 6:
            #disregard non-TCP packets
            continue

        if (ip_pkt.src != server_ip) and (ip_pkt.src != client_ip):
            #uninteresting source IP address
            continue

        if(ip_pkt.dst != server_ip) and (ip_pkt.dst != client_ip):
            #uninteresting destination IP address
            continue

        tcp_pkt = ip_pkt[TCP]

        direction = PktDirection.not_defined

        if ip_pkt.src == client_ip:
            if tcp_pkt.sport != int(client_port):
                continue
            if ip_pkt.dst != server_ip:
                continue
            if tcp_pkt.dport != int(server_port):
                continue
            direction = PktDirection.clientServer
        elif ip_pkt.src == server_ip:
            if tcp_pkt.sport != int(server_port):
                continue
            if ip_pkt.dst != client_ip:
                continue
            if tcp_pkt.dport != int(client_port):
                continue
            direction = PktDirection.serverClient
        else:
            continue

        interestingPacketsCount += 1
        if interestingPacketsCount == 1:
            first_pkt_timestamp = (pkt_metadata.tshigh << 32) | pkt_metadata.tslow
            first_pkt_timestamp_resolution = pkt_metadata.tsresol
            first_pkt_ordinal = count

        last_pkt_timestamp = (pkt_metadata.tshigh << 32) | pkt_metadata.tslow
```

```
        last_pkt_timestamp_resolution = pkt_metadata.tsresol
        last_pkt_ordinal = count

        this_pkt_relative_timestamp = last_pkt_timestamp - first_pkt_timestamp

        if direction == PktDirection.clientServer:
            if client_seq_offset is None:
                client_seq_offset = tcp_pkt.seq
            relative_offset_seq = tcp_pkt.seq - client_seq_offset

        else:
            assert direction == PktDirection.serverClient
            if server_seq_offset is None:
                server_seq_offset = tcp_pkt.seq
            relative_offset_seq = tcp_pkt.seq - server_seq_offset

        #if this TCP packet has the Ack bit set, then it must carry an ack number
        if 'A' not in str(tcp_pkt.flags):
            relative_offset_ack = 0
        else:
            if direction == PktDirection.clientServer:
                relative_offset_seq = tcp_pkt.ack - server_seq_offset
            else:
                relative_offset_seq = tcp_pkt.ack - client_seq_offset

        #determine the tcp payload length. IP fragmentation will mess up this logic so first check that the packet is unfragmented
        if(ip_pkt.flags == 'MF') or (ip_pkt.frag != 0):
            print('No support for fragmented IP packets')
            break

        tcp_payload_len = ip_pkt.len - (ip_pkt.ihl * 4) - (tcp_pkt.dataofs * 4)

        #print
        fmt = '[{ordnl:>5}]{ts:>10.6f}s flag={flag:<3s} seq={seq:<9d} ack={ack:<9d} len={len:<6d}'
        if direction == PktDirection.clientServer:
            fmt = '{arrow}' + fmt
            arr = '-->'
        else:
            fmt = '{arrow:>69}' + fmt
            arr = '<--'

        print(fmt.format(arrow = arr,
                         ordnl = last_pkt_ordinal,
                         ts = this_pkt_relative_timestamp/pkt_metadata.tsresol,
                         flag = str(tcp_pkt.flags),
                         seq = relative_offset_seq,
                         ack = relative_offset_ack,
                         len = tcp_payload_len))

    print('{} contains {} packets ({} interesting)'.format(fileName, count, interestingPacketsCount))
    print('First packet in connection: Packet #{} {}'.format(first_pkt_ordinal, printable_timestamp(first_pkt_timestamp, first_pkt_timestam
    print('Last packet in connection: Packet #{} {}'.format(last_pkt_ordinal, printable_timestamp(last_pkt_timestamp, last_pkt_timestamp_re
```

```
(base) annalisaallan@Annalisas-MacBook-Pro pcap % [K[?2004hcurl https://static-labs.tryhackme.cloud/sites/favicon/images/favicon.ico | md5s
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on en3 !
Opening example-01.pcap...
-->[ 2585]  0.000000s flag=S   seq=0         ack=0         len=0
                                                          <--[ 2586]  0.307193s flag=SA  seq=1         ack=0         len=0
-->[ 2587]  0.307242s flag=A   seq=1         ack=0         len=0
-->[ 2588]  0.307359s flag=PA  seq=1         ack=0         len=174
                                                          <--[ 2589]  0.620760s flag=A   seq=175       ack=0         len=0
                                                          <--[ 2590]  0.620798s flag=A   seq=175       ack=0         len=2880
-->[ 2591]  0.620823s flag=A   seq=2881      ack=0         len=0
                                                          <--[ 2592]  0.620843s flag=A   seq=175       ack=0         len=1440
-->[ 2593]  0.620849s flag=A   seq=4321      ack=0         len=0
                                                          <--[ 2594]  0.620870s flag=A   seq=175       ack=0         len=5760
-->[ 2595]  0.620877s flag=A   seq=10081     ack=0         len=0

...

-->[22579] 61.147676s flag=A   seq=52328684  ack=0         len=0
-->[22580] 61.148632s flag=FA  seq=52328684  ack=0         len=0
                                                          <--[22581] 61.440260s flag=FA  seq=176       ack=0         len=0
-->[22582] 61.440295s flag=A   seq=52328685  ack=0         len=0
example-01.pcap contains 22639 packets (14975 interesting)
```

```
First packet in connection: Packet #2585 2018-09-26 11:51:02.883718124
Last packet in connection: Packet #22582 2018-09-26 11:52:04.324012912
```