Lance Wetzel

Network Traffic Analysis Lab 2

Basic traffic stats

1. [5 pts] How many packets does the trace contain?

25864733

2. [5 pts] What is the average traffic rate (in bits per second) of the trace?

9355.12 bits/sec

3. [5 pts] How many IPv4 packets does the trace contain?

25864733

4. [5 pts] Estimate the size of the telescope (i.e., how many addresses does it monitor). Provide your estimate in "stroke" notation, e.g., "/8."

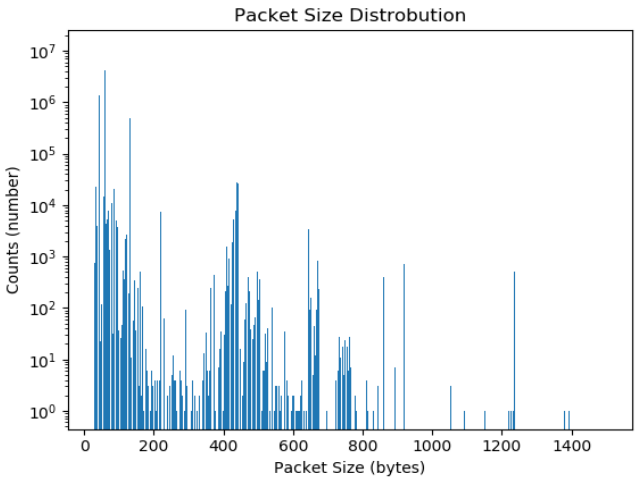
/24

The smallest address is 7.81.152.0 and the largest is 7.81.152.255

5. [10 pts] What is the packet IP protocol distribution (e.g., TCP, UDP, etc)? (A table showing the 5 top protocols and their respective contributions is fine.)

Protocol	Counts
TCP	21,892,184
UDP	3,505,521
ICMP	465,878
SCTP	1,097
GRE	47

6. [10 pts] Plot a histogram of the packet size distribution.



7. [5 pts] Find the documentation for PcapNg online. Briefly (no more than 2 or 3 sentences) describe the differences between pcap and PcapNg.

pcapNg adds several features and organizes the protocol is such a way to allow for easier expansion of new features to be added without invalidating tools written before those features were implemented. Some of the new features are the ability to distinguish between capture interfaces in the pcapng header, allowing captures from multiple interfaces to be stored in the same file. It also gives the ability to change the timestamp resolution to be able to record times faster than microseconds.

Backstatter

8. [5 pts] Explain why the TCP SYN/ACK and TCP RST packets in the trace are likely the result of traffic with spoofed source IP addresses

A legitimate TCP SYN/ACK packet should only be sent in response to a TCP SYN. Because we know there are no active users on the network telescope, the only reason a SYN/ACK would to be sent to a telescope address is the original sender spoofed their IP address and somehow picked an address in the telescope. Similarly, RST packets are only sent in response to a unexpected TCP packet therefore the instigating packet must have come from somewhere not in the telescope address space.

9. [5 pts] Besides TCP SYN/ACK and TCP RST packets, what other traffic in the trace is likely the result of spoofed traffic?

Any traffic that is a response to an event that had to be initiated by some party. This would include DNS responses and ICMP error messages.

10. [10 pts] Identify the host in the trace that experienced the highest-rate DoS attack. Briefly explain how you arrived at your answer.

Highest rate Ddos target: ('185.180.14.91', 6962)

I filtered for all TCP traffic that had only the SYN/ACK or RST flags set, and then kept a dictionary of IP address that sent those packets with the number of times a packet was received by that IP address. After all packets were inspected, I sorted the dictionary based on the value of the counts.

This method will not see any ddos attacks based other protocols (ICMP or DNS).

11. [5 pts] For the host in Q 10, estimate the rate of the DoS attack in packets per second.

1st packet from 185.180.14.91: 1566908507.862047

Last packet from 185.180.14.91: 1567698277.419868: 789,769.557821 seconds long

Observed attack rate = 6962 packets/789,769.557821 seconds = 0.00881529 packets/second

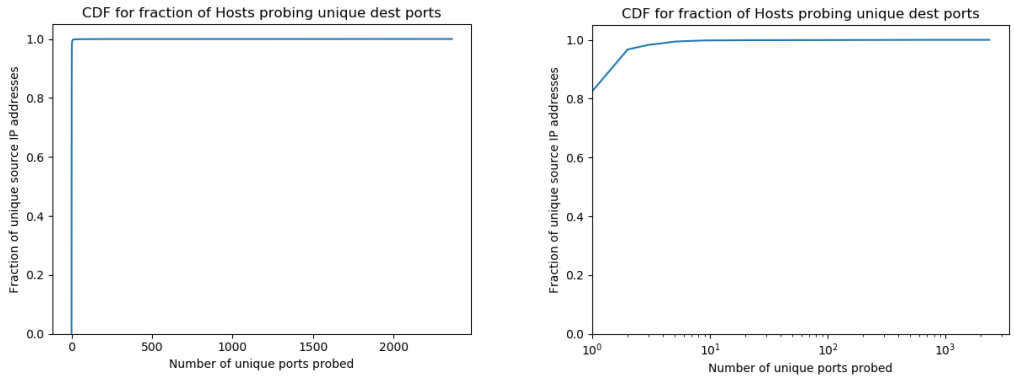
Actual attack rate >= Observed attack rate * (total IP space / observed IP space)

>= 0.0881529 * ($2^{32}/2^{8}$) >= 147,895.011 packets per second

12. [5 pts] What are the five (5) most frequently probed TCP ports?

Port	# of times probed
3010	4403124
3002	4383416
23	1497956
445	1096344
22	303579

13. [15 pts] Create a cumulative distribution plot of fraction of hosts as a function of the number of unique destination ports. The x-axis is the number of unique TCP ports probed per source, and the y-axis is the cumulative fraction of sources.



To collect this data I made a dictionary mapping source IP address to a list containing each unique port that address attempted to connect to. To be considered 'probing' a source had send a TCP SYN packet. The data shows that ~80% of probing sources only attempted 1 unique port, this is indicative of horizontal network scans. However, the tail of the CDF is very long, indicating that a small number of sources are probing ~2000 ports, indicative of vertical scans.

The right-most graph shows the same data, but the x-axis is shown in log scale, allowing better inspection of the lower end of the values. The log scale shows that just over 80% of sources only probe one port, and ~99% of sources probe less than 10 ports.

14. [5 pts] How many IP sources initiated connections to both TCP port 23 and 2323, but no other ports?

15589 unique IP sources attempted connection with only ports 23 and 2323

15. [5 pts] Perform some Internet research to determine what is the most likely cause of the TCP port 23 and 2323 traffic in Q 14. Please answer with no more than 2-3 sentences.

As of Nov 2017 there is a know exploit on IoT devices made by ZyXel. Attackers are able to use default passwords of admin/CentryL1nk and admin/QwestM0dem to gain control of IoT devices and infect them with botnet malware. This has been attributed to the Mirai botnet/malware.

Code:

```
import dpkt
import socket
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
import operator

fd = open("telescope.2019.anon.pcap", "rb")
pcap = dpkt.pcap.Reader(fd)

fd = open("first100.pcap", "rb")
short_pcap = dpkt.pcap.Reader(fd)

num_pkts = 0
ipv4_count = 0
ip_val_max = 0
ip_val_min = 255
ip_val_expect = '152'
unexpect_ip_val = []

tport_protcol = {}

packet_size_distro = {}

ddos_targets = {}

port_count = {}

unique_port_map = {}
unique_port_count = {}
end_ddos_ts = 0


#-----------------------------------------------packet analysis loop--------------
-----------------------------------
#-------------------------------------------------------------------------------
-----------------------------------

pkt_count = 0
for ts, data in pcap:
    if pkt_count == 0:
        start_ts = ts
    else:
```

```
        end_ts = ts
    #print("timestamp: " , ts)              #print the time stamp
    pkt_count += 1
    '''
    if pkt_count > 50000:
        break
    '''
    eth = dpkt.ethernet.Ethernet(data)      #make eth object
    #if eth.type == 2048:                   #used for counting ipv4 addresses
    #    ipv4_count +=1
    #print(eth.type)
    ip = eth.data                           #make ip object, dpkt already knows t
his is an dpkt.ip type,
    tport_num = ip.p
    ip_src = socket.inet_ntoa(ip.src)
    #print('transport protocol: ' ,tport_num)                                #print
protocol number of the ip object (this will be the transport layer protocol numbe
r)
    if tport_num == 6:
        try:
            tcp = ip.data
            if tcp.flags == 18 or tcp.flags == 4:                           #if
 tcp flags == SYN/ACK then the ip source is a ddos target
                if ip_src not in ddos_targets:          #store counts of ddos res
ponses in a dict. key is scr ip addr, val is counts of ddos responses
                    ddos_targets[ip_src] = 1
                    if ip_src == '185.180.14.91':
                        begin_ddos_ts = ts
                else:
                    count = ddos_targets.get(ip_src)
                    ddos_targets[ip_src] = count + 1
                    if ip_src == '185.180.14.91' and ts > end_ddos_ts:
                        end_ddos_ts = ts



            dest_port = tcp.dport
            if tcp.flags == 2:
                #print('IP source: ' , ip_src , ' probed port ' , dest_port)
                if dest_port not in port_count:          #counts number of times
each port is probed. key is port #, val is count
                    port_count[dest_port] = 1
                else:
                    count = port_count.get(dest_port)
                    port_count[dest_port] = count + 1
```

```
                if ip_src not in unique_port_map:       #keeps track of what port
s each ip src has probed. key is ip addr, val is list of all ports that addr has
probed
                    unique_port_map[ip_src] = [dest_port]
                    unique_port_count[ip_src] = 1
                else:
                    if dest_port not in unique_port_map[ip_src]:
                        unique_port_map[ip_src].append(dest_port)
                        count = unique_port_count.get(ip_src)
                        unique_port_count[ip_src] = count + 1
        except:
            pkt_count -= 1
            print('bad packet')

#-------------------------------------------------end packet analysis loop------
-----------------------------------------------
#-------------------------------------------------------------------------------
-----------------------------------------------
    '''
    packet_size = ip.len                        #for recording the counts of all
packet sizes
    if packet_size not in packet_size_distro:
        packet_size_distro[packet_size] = 1
    else:
        count = packet_size_distro.get(packet_size)
        packet_size_distro[packet_size] = count + 1
    '''

    '''
    if tport_num not in tport_protcol:          #adds transport protocol to dict
to count instance of each protocol use
        tport_protcol[tport_num] = 1
    else:
        count = tport_protcol.get(tport_num)
        tport_protcol[tport_num] = count + 1
    '''
    #print("unformatted IP address " , ip.dst)                          #print i
p desination but will be in an odd format
    #ip_dest = socket.inet_ntoa(ip.dst)
    #print("formatted IP address " , ip_dest)        #must use socket.inet.ntoa
to convert to dottect decimal
    #print("IP length: " , ip.len)
    #num_pkts += 1
```

```
#print('number of packets: ' , num_pkts)
#print('number of IPv4 packets: ' , ipv4_count)
#print('min IP last octet: ' , ip_val_min)
#print('max IP last octet: ' , ip_val_max)
#print('IP addrs not in x.x.152.y ' , unexpect_ip_val)
#print("protocol distribution: " , tport_protcol)
#print('packet size distrobution: ' , final_packet_size)
#print('ddos targets: ' , ddos_targets)
#print('scanned port distro: ' , port_count)
#print('port map: ' , unique_port_map)
print('Number of packets: ' , count)
print('Start time: ' , start_ts)
print('End time: ' , end_ts)

#-----------------------------------------------ddos targets--------------------
-----------------------------------
#print(ddos_targets)
try:
    sorted_ddos_targets = sorted(ddos_targets.items(),key=operator.itemgetter(1),
reverse = True)
    print('Highest rate Ddos targets: ')
    for i in range(5):
        print(sorted_ddos_targets[i])
except:
    print('no ddos targets')

try:
    print('ddos began at ' , begin_ddos_ts)
    print('ddos ended at ' , end_ddos_ts)
except:
    print('highest rate ddos not found')
#-----------------------------------------------------end ddos targets----------
-------------------------------------------

#-----------------------------------------------------most probed ports---------
-------------------------------------------

sorted_port_count = sorted(port_count.items(),key=operator.itemgetter(1),reverse
= True)
print('\n')
print('Most probed ports:')
for i in range(5):
```

```
    print(sorted_port_count[i])
#-------------------------------------------------------end most probed ports-----
-------------------------------------------------

#-------------------------------------------------------telnet scanner------------
-------------------------------------------------
print('\n')
telnet_try = 0
for key in unique_port_map:
    if len(unique_port_map[key]) == 2:
        if unique_port_map[key][0] == 23 and unique_port_map[key][1] == 2323:
            telnet_try += 1
            #print(key , ' attempted connection with port 23 and 2323')
        elif unique_port_map[key][0] == 2323 and unique_port_map[key][1] == 23:
            telnet_try += 1
            #print(key , ' attempted connection with port 23 and 2323')


print(telnet_try , ' unique IP sources attempted connection with port 23 and 2323
')
print('\n')

#-------------------------------------------------------end telnet scanner--------
-----------------------------------------------
#print('port count: ' , port_count)
#print('port map: ' , unique_port_map)
#print('unique port count: ' , unique_port_count)


#-------------------------------------------------------unique probed port CDF-----
-------------------------------------------------
sorted_unique_port_count = sorted(unique_port_count.items(),key=operator.itemgett
er(1),reverse = True)
print('highest number of unique ports probe by an IP address (according to unique
_port_count): ')
for i in range(5):
    print(sorted_unique_port_count[i])

most_probing_source = sorted_unique_port_count[0][0]
most_probes = sorted_unique_port_count[0][1]
print('probe number check')
if len(unique_port_map[most_probing_source]) == most_probes:
    print('check passed')
else:
    print('check failed')
```

```
    print(most_probing_source , ':' , most_probes , 'does not match' , len(unique
_port_map[most_probing_source]))


cdf_builder = {}
for key in unique_port_count:
    if unique_port_count[key] not in cdf_builder:
        cdf_builder[unique_port_count[key]] = 1
    else:
        count = cdf_builder.get(unique_port_count[key])
        cdf_builder[unique_port_count[key]] = count + 1

reverse_cdf_builder = sorted(cdf_builder.items(),key=operator.itemgetter(0),rever
se = True)
print('highest number of unique ports probe by an IP address (according to cdf_bu
ilder): ')
try:
    for i in range(5):
        print(reverse_cdf_builder[i])
except:
    print('cdf builder: ' , cdf_builder)
    print('reverse cdf builder: ' , reverse_cdf_builder)

#print('cdf builder: ' , cdf_builder)
total = 0
for key in cdf_builder:
    total = total + cdf_builder[key]
print('total probe sources: ' , total)

sorted_cdf_builder = sorted(cdf_builder.items(),key=operator.itemgetter(0))
#print('soreted cdf builder: ' , sorted_cdf_builder)
x_val = []
y_val = []
for item in sorted_cdf_builder:
    x_val.append(item[0])
    y_val.append(item[1])

running_total = 0
for i in range(len(y_val)):
    fraction = (y_val[i] / total) + running_total
    y_val[i] = fraction
    running_total =  y_val[i]

#print(x_val)
#print(y_val)
```

```
adjusted_x_val = [0]
adjusted_x_val.extend(x_val)
#print(adjusted_x_val)

adjusted_y_val = [0]
adjusted_y_val.extend(y_val)
#print(adjusted_y_val)

plt.plot(adjusted_x_val,adjusted_y_val)
plt.xlabel('Number of unique ports probed')
plt.ylabel('Fraction of unique source IP addresses')
plt.title('CDF for fraction of Hosts probing unique dest ports')
plt.ylim(bottom = 0)
plt.show()

plt.plot(adjusted_x_val,adjusted_y_val)
plt.xlabel('Number of unique ports probed')
plt.ylabel('Fraction of unique source IP addresses')
plt.title('CDF for fraction of Hosts probing unique dest ports')
plt.ylim(bottom = 0)
plt.xscale('log')
plt.xlim(left = 0)
plt.show()


'''
orig_stdout = sys.stdout
ddos = open("ddos_targets.txt","w+")
sys.stdout = ddos
print(ddos_targets)
sys.stdout = orig_stdout
ddos.close()

port_file = open("scanned_port_distro.txt","w+")
sys.stdout = port_file
print(port_count)
sys.stdout = orig_stdout
port_file.close()

port_map = open('scanned_port_map.txt' , "w+")
sys.stdout = port_map
print(unique_port_map)
sys.stdout = orig_stdout
port_map.close()
```

```
'''
```