CS 176B – Network Computing

Project Deliverable #C

Packet analysis and filtering using Python

Submitted by:
Venkat Raman
Vishal Hosakere

CONTENTS

Abstract:

Goal of the project is to implement filters on the packets captured by Wireshark and perform analysis on the packets. Packet analysis is useful for understanding, debugging, and verifying packet flow properties of the network. Information is exchanged in the form of packets over the internet and by analyzing the packets we can get an insight on how the internet works. Packets give information on the type of protocols used by different nodes in the network, IP address of the nodes involved in transmitting the packets, encryption used to send secured data and much more. So, it is important to understand information carried by the packets to know the working of internet better.

Since the packet carries much more information that needed for a particular understanding, we tend to get lost while searching for a particular information. This is where packet filtering comes into play. Filtering allows us to eliminate information that might not be need for the particular analysis and give the packet information related to the specific information needed. Filtered data is again present in a readable format so that the end user can use them to understand them easily. Filters can be applied based on the protocols used, links between two nodes, data transferred between any two IPs and so on.

Introduction:

Packet analysis, also referred to as packet sniffing or protocol analysis, is the process of capturing and interpreting data as it flows across a network in order to better understand what is happening on that network. Packet analysis is typically performed by a packet sniffer, a tool used to capture raw network data going across the wire.

Packet analysis can help with the following:

- Understanding network characteristics
- Learning who is on a network
- Determining who or what is utilizing available bandwidth
- Identifying peak network usage times
- Identifying possible attacks or malicious activity
- Finding unsecured and bloated applications

There are various types of packet-sniffing programs, including both free and commercial ones. Each program is designed with different goals in mind. A few popular packet-analysis programs are tcpdump, and Wireshark. tcpdump is a command-line program. Wireshark have graphical user interfaces (GUIs). Command line program equivalent of Wireshark is tshark. We used tshark to fetch the packet information.

Factors to keep in mind when designing a good packet sniffer are:

- Supported protocols – Filter is designed to support TCP, UDP and DNS protocols for now. Protocols can be added later.
- User-friendliness – Information can be obtained by giving necessary input to the program.
- Operating system support – Filter can be run on both windows and linux distributions.

Literature Survey:

Packet analysis on the data being transmitted over the internet is very important in understanding how and what type of data is exchanged over the internet. But the data is in a complex form and cannot be easily understood by a common individual. So, it is necessary to provide filters to ease the understanding of the data to end users. This is the main motivation of the project.

Methodology and Results:

Packers were captured in Wireshark for about 10 mins. The output pcap file is converted to CSV file with all the default fields. The pcap file was taken as input to the python script and various filters were implemented on this data.

1. Filter: get_src

The csv file is read, and the data is stored in a list containing all packets. The list is scanned through to find out all the different IP addresses present under the source IP column. Every new source IP is appended to a new list and it is returned when all the packets are checked.

Inputs: csv filename

Output: list of different source IP addresses

Result:

```
64.233.180.156
74.119.119.70
188.92.41.202
74.119.119.68
74.119.119.65
172.217.2.230
74.119.119.78
178.250.0.78
74.119.119.76
104.28.23.93
172.217.9.170
199.96.57.6
216.58.218.173
217.20.147.1
199.59.148.85
217.20.156.158
217.20.156.72
151.101.1.63
216.58.218.170
104.16.102.102
104.25.119.108
104.16.101.102
104.16.89.50
172.217.2.227
34.192.178.135
205.251.203.41
64.95.32.18
172.217.12.34
64.95.32.51
216.58.218.195
54.243.128.120
```

Only a portion of the output is displayed here

2. Filter: get_src_dest

The csv file is read, and the data is stored in a list containing all packets. The list is scanned through to find out all the different IP addresses present under the source IP column. The corresponding destination IPs for every source IP is appended to a dictionary for that source IP and it is returned when all the packets are checked.

Inputs: csv filename

Output: Dictionary mapping source IPs to its corresponding destination IPs

Result:

```
151.101.65.140  has sent packet to ['192.168.1.9']
13.57.91.89  has sent packet to ['192.168.1.9']
188.92.40.78  has sent packet to ['192.168.1.9']
54.88.164.103  has sent packet to ['192.168.1.9']
89.206.219.187  has sent packet to ['192.168.1.9']
34.208.151.55  has sent packet to ['192.168.1.9']
159.127.41.108  has sent packet to ['192.168.1.9']
216.58.218.202  has sent packet to ['192.168.1.9']
216.58.218.206  has sent packet to ['192.168.1.9']
157.240.22.20  has sent packet to ['192.168.1.9']
151.101.1.63  has sent packet to ['192.168.1.9']
157.240.22.25  has sent packet to ['192.168.1.9']
35.196.185.31  has sent packet to ['192.168.1.9']
104.25.119.108  has sent packet to ['192.168.1.9']
192.168.1.5  has sent packet to ['239.255.255.250', '192.168.1.255', '224.0.0.252']
104.28.23.93  has sent packet to ['192.168.1.9']
63.251.210.243  has sent packet to ['192.168.1.9']
34.203.79.136  has sent packet to ['192.168.1.9']
64.233.180.156  has sent packet to ['192.168.1.9']
fe80::8f5:52e6:4fba:f32c  has sent packet to ['ff02::fb']
52.8.201.211  has sent packet to ['192.168.1.9']
104.16.91.60  has sent packet to ['192.168.1.9']
192.0.73.2  has sent packet to ['192.168.1.9']
64.95.32.60  has sent packet to ['192.168.1.9']
104.16.164.50  has sent packet to ['192.168.1.9']
151.101.26.109  has sent packet to ['192.168.1.9']
64.95.32.65  has sent packet to ['192.168.1.9']
172.217.9.8  has sent packet to ['192.168.1.9']
64.94.107.12  has sent packet to ['192.168.1.9']
148.251.77.209  has sent packet to ['192.168.1.9']
209.133.193.138  has sent packet to ['192.168.1.9']
54.213.234.155  has sent packet to ['192.168.1.9']
35.160.108.211  has sent packet to ['192.168.1.9']
23.57.55.152  has sent packet to ['192.168.1.9']
216.155.194.56  has sent packet to ['192.168.1.9']
107.23.74.241  has sent packet to ['192.168.1.9']
```

Packets are sent from the source to list of destination IPs

3. Filter: get_payload_1

Data is read from the csv file to fetch the list of all the source IPs. For all the IPs in the list of source IPs, the amount of data sent by each IP is fetched from the csv file and stored in the dictionary corresponding to the source IP. Once done, the total data sent by every source IP is returned.

Inputs: csv filename

Output: Dictionary mapping source IP to data sent by them

Result:

```
216.58.218.173 has sent  2688 bytes
172.217.2.238 has sent  29376 bytes
216.58.218.170 has sent  5733 bytes
172.217.2.234 has sent  2985 bytes
172.217.9.131 has sent  1925 bytes
172.217.2.232 has sent  50641 bytes
172.217.2.230 has sent  54378 bytes
172.217.6.130 has sent  4634 bytes
172.217.2.226 has sent  77244 bytes
172.217.9.170 has sent  49795 bytes
64.233.171.154 has sent  5572 bytes
172.217.9.8 has sent  1697 bytes
172.217.14.162 has sent  62156 bytes
172.217.6.161 has sent  49016 bytes
172.217.1.234 has sent  5252 bytes
108.177.104.95 has sent  4591 bytes
172.217.6.174 has sent  17983 bytes
216.58.218.206 has sent  8832 bytes
74.125.199.104 has sent  1301 bytes
216.58.194.34 has sent  5150 bytes
172.217.2.225 has sent  466555 bytes
172.217.2.227 has sent  93767 bytes
216.58.218.163 has sent  99317 bytes
192.168.1.9 has sent  485475 bytes
172.217.9.142 has sent  16430 bytes
216.58.194.98 has sent  2333 bytes
172.217.9.162 has sent  4919 bytes
64.233.171.155 has sent  35801 bytes
216.58.218.198 has sent  419074 bytes
216.58.218.174 has sent  82364 bytes
172.217.6.162 has sent  20485 bytes
172.217.6.163 has sent  16794 bytes
216.58.218.193 has sent  7203 bytes
64.233.180.156 has sent  20732 bytes
216.58.218.195 has sent  9916 bytes
```

4. Filter: get_payload_2

Data is read from the csv file to fetch the list of all the source IPs. For all the IPs in the list of source IPs, the corresponding destination IPs list is determined. Now with this list, the data sent from source IPs to the corresponding destination IPs is calculated from the csv file in bytes.

Inputs: csv file

Output: Dictionary mapping source IP and the destination IP to the data sent between them

Results:

```
64.233.171.154 has sent 192.168.1.9   5572 bytes
172.217.9.8 has sent 192.168.1.9   1697 bytes
172.217.14.162 has sent 192.168.1.9   62156 bytes
172.217.6.161 has sent 192.168.1.9   49016 bytes
172.217.1.234 has sent 192.168.1.9   5252 bytes
108.177.104.95 has sent 192.168.1.9   4591 bytes
172.217.6.174 has sent 192.168.1.9   17983 bytes
216.58.218.206 has sent 192.168.1.9   8832 bytes
74.125.199.104 has sent 192.168.1.9   1301 bytes
216.58.194.34 has sent 192.168.1.9   5150 bytes
172.217.2.225 has sent 192.168.1.9   466555 bytes
172.217.2.227 has sent 192.168.1.9   93767 bytes
216.58.218.163 has sent 192.168.1.9   99317 bytes
192.168.1.9 has sent 216.58.194.138   875 bytes
192.168.1.9 has sent 74.125.28.105   14257 bytes
192.168.1.9 has sent 74.125.28.147   24261 bytes
192.168.1.9 has sent 216.58.194.66   22672 bytes
192.168.1.9 has sent 216.58.218.202   8995 bytes
192.168.1.9 has sent 172.217.12.34   59984 bytes
192.168.1.9 has sent 172.217.9.138   9650 bytes
192.168.1.9 has sent 64.233.180.189   4930 bytes
192.168.1.9 has sent 216.58.218.173   2826 bytes
192.168.1.9 has sent 172.217.2.238   13826 bytes
192.168.1.9 has sent 216.58.218.170   1485 bytes
192.168.1.9 has sent 172.217.2.234   2447 bytes
192.168.1.9 has sent 172.217.9.131   620 bytes
192.168.1.9 has sent 172.217.2.232   2471 bytes
192.168.1.9 has sent 172.217.2.230   11443 bytes
192.168.1.9 has sent 172.217.6.130   2397 bytes
192.168.1.9 has sent 172.217.2.226   21149 bytes
192.168.1.9 has sent 172.217.9.170   1952 bytes
192.168.1.9 has sent 64.233.171.154   2866 bytes
```

5. Filter: dns_get

Data is read from the csv file to fetch the list of all the packets being exchanged over the network. All the DNS packets are filtered out from the packet list. Now standard queries are fetched from the DNS packet information using which all the websites being resolved is split from the data.

Inputs: csv file

Output: List of all the website being resolved

Results:

```
pix04.revsci.net
rs.gwallet.com
s.opendsp.com
atp.mxptint.net
imrk.net
atv.sync.yume.com
finder.cox.net
usersync.videoamp.com
adaptvbidder-east.extend.tv
aolbidder-east.extend.tv
match.prod.bidr.io
icv6.imrk.net
adapj.rtb.adxl.com
bh.contextweb.com
a.tribalfusion.com
sync.ipredictive.com
rtb.adentifi.com
static.sparcvideo.com
u.acuityplatform.com
px.owneriq.net
cookie.brealtime.com
pixel.tapad.com
secure.adnxs.com
odr.mookiel.com
su.addthis.com
ads.scorecardresearch.com
ads.adap.tv
player.vimeo.com
fpdl.vimeocdn.com
play.google.com
www.gsmarena.com
cdn.gsmarena.com
cdn2.gsmarena.com
```

6. Filter: avg_time_by_IP

Data is read from the csv file to fetch the list of all the source IPs. For every source IP in the list, the time taken to send every packet by the IPs is determined and summed up over the session. Finally, the average time taken by every source IP to send the packets is determined and returned in a dictionary.

Inputs: csv file

Output: Dictionary mapping the average time to each IP

Results:

```
192.168.1.1 took an average time of 0.622 s
64.233.180.189 took an average time of 11.201 s
192.168.1.9 took an average time of 0.012 s
192.168.1.2 took an average time of 2.613 s
192.168.1.11 took an average time of 7.399 s
172.217.6.163 took an average time of 13.937 s
74.125.199.104 took an average time of 2.394 s
157.240.22.25 took an average time of 0.031 s
157.240.22.35 took an average time of 0.107 s
34.231.119.41 took an average time of 6.150 s
54.230.86.225 took an average time of 2.791 s
216.58.218.174 took an average time of 3.135 s
157.240.22.20 took an average time of 0.037 s
130.211.16.53 took an average time of 19.199 s
157.240.22.19 took an average time of 1.547 s
172.217.6.162 took an average time of 9.036 s
172.217.2.238 took an average time of 10.622 s
172.217.2.234 took an average time of 6.089 s
52.21.219.50 took an average time of 14.116 s
172.217.9.138 took an average time of 1.437 s
192.168.1.5 took an average time of 10.460 s
64.233.169.188 took an average time of 43.649 s
172.217.9.131 took an average time of 6.619 s
192.168.1.14 took an average time of 18.198 s
```

7. Filter: retransmission

Data is read from the csv file to fetch all the source IPs. TCP retransmissions by every IP is looked up and mapped into a dictionary against the source and destination IPs.

Input: csv file

Output: Dictionary mapping retransmissions between the source and destination IPs.

Results:

```
104.254.150.77 has sent retransmission message to ['192.168.1.9']
52.22.91.181 has sent retransmission message to ['192.168.1.9']
157.240.22.35 has sent retransmission message to ['192.168.1.9']
23.34.142.93 has sent retransmission message to ['192.168.1.9']
34.192.178.135 has sent retransmission message to ['192.168.1.9']
188.92.40.78 has sent retransmission message to ['192.168.1.9']
104.16.101.102 has sent retransmission message to ['192.168.1.9']
146.20.128.107 has sent retransmission message to ['192.168.1.9']
69.89.74.101 has sent retransmission message to ['192.168.1.9']
54.230.87.25 has sent retransmission message to ['192.168.1.9']
209.73.190.12 has sent retransmission message to ['192.168.1.9']
130.211.16.53 has sent retransmission message to ['192.168.1.9']
151.101.26.2 has sent retransmission message to ['192.168.1.9']
151.101.26.109 has sent retransmission message to ['192.168.1.9']
104.28.23.93 has sent retransmission message to ['192.168.1.9']
148.251.77.209 has sent retransmission message to ['192.168.1.9']
52.10.217.21 has sent retransmission message to ['192.168.1.9']
54.192.117.71 has sent retransmission message to ['192.168.1.9']
157.240.22.20 has sent retransmission message to ['192.168.1.9']
52.8.201.211 has sent retransmission message to ['192.168.1.9']
209.177.149.138 has sent retransmission message to ['192.168.1.9']
151.101.24.166 has sent retransmission message to ['192.168.1.9']
```

8. dns_get2

Data is read from the csv file to fetch the list of all the packets being exchanged over the network. All the DNS packets are filtered out from the packet list. Now standard queries are fetched from the DNS packet information using which all the websites being resolved is split from the data. Now the blacklisted site is searched against the list of websites and a dictionary with the IP and the corresponding blacklisted website is mapped,

Input: csv file

Output: List of IPs contacting the blacklisted sites.

```
192.168.1.9  visited  syndication.twitter.com
192.168.1.9  visited  platform.twitter.com
192.168.1.9  visited  twitter.com
192.168.1.9  visited  analytics.twitter.com
```

In this example twitter was the keyword that was blacklisted.

Demo:

Code is written to provide the filters that are mentioned in the report. Individual filters will be run to display the desired outputs.

Comments:

Difficulties faced:

• Converting the pcap output from Wireshark to the csv file using python. There was a problem in redirecting the output to a csv file using subprocess function call. Finally using os.system provided the desired results.

Conclusion:

Filters were implemented on the packet data captured us wireshark application. Filters give us an insight on the packets without spending much effort on analyzing them.

References:

- https://www.lifewire.com/wireshark-tutorial-4143298
- https://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/
- http://repository.root-me.org/R%C3%A9seau/EN%20-%20Practical%20packet%20analysis%20-%20Wireshark.pdf