

Programming using libpcap

Tan Wei Xuan (49003140)

`tanweixuan@postech.ac.kr`

May 25, 2019

1 Installing Dependencies

Ensure that all dependencies are installed on the operating system. The operating system that I am using is **Ubuntu 18.04.2 bit subsystem** and the list of required dependencies are as listed below:

1. **GCC**
2. **build-essential**
3. **libpcap**

1.1 Installing GCC

The following linux command will install the **GCC compiler** on on Ubuntu 18.04. Open up terminal and enter:

```
$ sudo apt install g++
```

1.2 Installing build-essential

The following linux command will install the **build-essential** package on on Ubuntu 18.04. Open up terminal and enter:

```
$ sudo apt install build-essential
```

1.3 Installing libpcap

The following linux command will install the **libpcap** package on on Ubuntu 18.04. Open up terminal and enter:

```
$ sudo apt install libpcap-dev
```

2 Capturing Packets using libpcap

The file "question1.cpp" contains the source code required to capture **TCP, UDP and ICMP traffic** on my Network Interface Card for a period of 40 seconds. The key functions that I have utilised to capture these traffic are as follow:

1. **pcap_lookupdev()**

This function finds a default network device on which to capture packets from

2. **pcap_openlive()**

This function is used to establish a connection with the network device that can bring packets

3. **pcap_compile()** and **pcap_setfilter()**

These functions are used to filter and capture only **TCP, UDP and ICMP** packets which go through the well-known ports (1 - 1024).

4. **pcap_dump_open()** and **pcap_dump** pcap_dump_open() opens a dump file to output and pcap_dump prints a packet to the dump file created.

In order to compile the source code (question1.cpp), we need to run the following command.

```
$ gcc question1.c -lpcap
```

In order to execute our program, we have to run the following command.

```
$sudo ./a.out <argument1> <argument2>
```

The program expects the following two arguments:

1. **<Argument 1>**

The first argument is the period to capture in seconds.

2. **<Argument 2>**

The second argument is the name of the output file to print the captured packets to.

The following command will generate the output file containing the captured packets.

```
jerm@jerm-VirtualBox:~/Desktop/Assignment 6/Assignment 6$ sudo ./a.out 40 packetCapture.pcap
[sudo] password for jerm:
Using device: enp0s3
```

Figure 1: *Executing question1.cpp*

My packet capture file is provided in **packetCapture.pcap**. The time period of capture is **40 seconds**.

3 Packet Analysis using libpcap

The file "question2.cpp" contains the source code required to capture **TCP, UDP and ICMP traffic** on my Network Interface Card and to print out the different traffic traces analysis results for the captured packets. The key steps and functionality of my code are as listed below:

Firstly, we will open our previously captured file for offline processing with the command **pcap_open_offline()**

```
if((file = pcap_open_offline(argv[1], error_buffer)) == NULL) {  
    printf("pcap_open_offline() failed.\n%s\n", error_buffer);  
    exit(1);  
}
```

Next, we parse through each packet till the end with the function **pcap_next**.

```
while((packet = pcap_next(file, &header)) != NULL) {  
    //Code not included for this portion  
}
```

Within the loop, for each packet, we perform traffic analysis through the following steps:

1. **Record timestamp of packet**

The timestamp for each packet is recorded. The timestamp and length of the packet can be found in the libpcap header.

```
if(total_packets == 1) {  
    time_first = header.ts.tv_sec * 1000000 + header.ts.tv_usec;  
}  
time_last = header.ts.tv_sec * 1000000 + header.ts.tv_usec;
```

2. **Analyse which Transport Layer Protocol is used**

We perform an analysis on which transport layer protocol is being used (TCP, UDP or ICMP) by adding an offset of 0x17 to the packet.

```
switch(*(packet + 0x17)) {  
    case 0x01: cnt_icmp++; break;  
    case 0x06: cnt_tcp++; break;  
    case 0x11: cnt_udp++; break;  
    default: break;  
}
```

3. **Analyse number of Packets of FTP, SSH, DNS and HTTP** We perform an analysis on the number of packets of FTP, SSH, DNS and HTTP by adding an offset of **0x22 - 0x23** and **0x24 - 0x25** to the packet to get the source and destination port respectively. As FTP uses port 21, SSH uses port 22, DNS uses port 53 and HTTP uses port 80, depending on the value of the source or destination port, we can categorise the traffic into either FTP, SSH, DNS or HTTP.

```
src_port = *(packet + 0x22) * 256 + *(packet + 0x23);
dest_port = *(packet + 0x24) * 256 + *(packet + 0x25);
if(src_port == 21 || dest_port == 21) cnt_ftp++;
else if(src_port == 22 || dest_port == 22) cnt_ssh++;
else if(src_port == 53 || dest_port == 53) cnt_dns++;
else if(src_port == 80 || dest_port == 80) cnt_http++;
```

4. Update Endhost List

When we reach an end's IP address, we update the endhost list. The endhost list is implemented as a linked list such that if an IP address has being used previously, it will update the packet count and total bytes to that node. Otherwise, it will create a new node at the end of the list with the new IP address. We use an offset of 0xb and 0xe to add the destination and source IPs to the endhost list respectively.

```
update_endhost(packet + 0xb, &hosts, header.len);
update_endhost(packet + 0xe, &hosts, header.len);
```

We can compile the source code (question2.cpp) the same way as we did previously.

```
$ gcc question2.c -lpcap
```

In order to execute our program, we have to run the following command.

```
$sudo ./a.out <argument1> <argument2>
```

The program expects the following two arguments:

1. **<Argument 1>**

The first argument is name of the packet capture file to perform analysis on. For my assignment, the name of the file is **packetCapture.pcap**.

2. **<Argument 2>**

The second argument is the name of the output file to print the endpoint analysis to. This file will contain the total number of packets and total number of bytes for each endpoint IP Address. For my assignment, the name of the output file is **endpointAnalysis.txt**.

The following command will generate the output file containing the endpoint analysis of the provided packet capture file.

```

jerm@jerm-VirtualBox:~/Desktop/Assignment 6/Assignment 6$ ./a.out packetCapture.pcap endpointAnalysis.txt
Total packets: 5212
Total bytes: 12944026

Time difference between first/last packet: 63.534s
Packets per protocol
TCP: 5064, UDP: 148, ICMP: 0

IP end host analysis written in "endpointAnalysis.txt"
In (IP address / # of packets / # of bytes order)
Packets per application
FTP: 0, SSH: 0, DNS: 148, HTTP: 107

Average packet size: 2483B
Average packet inter-arrival time: 12.190ms

```

Figure 2: *Executing question2.cpp*

The endpoint analysis (the number of packet and total bytes of each end host) is provided in **endpointAnalysis.txt**. Other required categories of analyzed data are shown in Figure 2 above. The categories of analyzed data will be printed out on the terminal upon execution of the program.

4 Packet Decapsulation using libpcap

The file **"question3.cpp"** contains the source code required to perform packet decapsulation of the given trace file. The implementation of the packet decapsulation are similar to **"question2.cpp"**, however, in order to know what is inside the IP packet that is encapsulated by the GTP protocol, a few minor changes (listed below) have been made to the code:

1. Obtaining Inner Packet's Length

Every packet in the provided trace file has two IP headers: The original one (outer packet) and another one within the GTP protocol (inner packet). The data outside the Inner Packet is 40 bytes, with 36 bytes being on the front and 4 bytes of Ethernet protocol footer at the back. Therefore, we can simply subtract 40 from the Outer Packet's length to obtain the Inner Packet's length.

```
total_bytes += (header.len - 40);
```

2. Accessing Inner Packet

For each packet, the offset difference between the Inner and Outer Packet is constant and thus we can simply add 40 to the packet pointer to access the Inner Packet.

```

while((packet = pcap_next(file, &header)) != NULL) {
    packet += 0x28;
}

```

We can compile the source code (question3.cpp) the same way as we did previously.

```
$ gcc question3.c -lpcap
```

In order to execute our program, we have to run the following command.

```
$sudo ./a.out <argument1> <argument2>
```

The program expects the following two arguments:

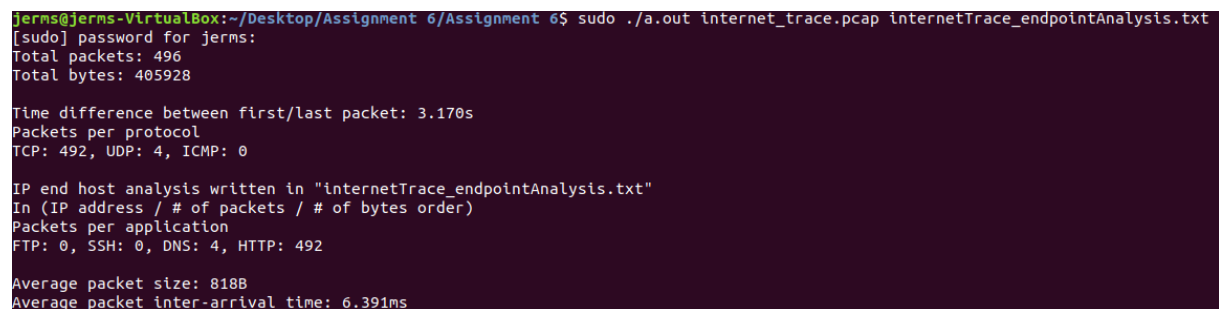
1. **⟨Argument 1⟩**

The first argument is name of the packet capture file to perform analysis on. We will be utilising the provided packet capture file, **internet_trace.pcap**.

2. **⟨Argument 2⟩**

The second argument is the name of the output file to print the endpoint analysis to. This file will contain the total number of packets and total number of bytes for each endpoint IP Address. For my assignment, the name of the output file is **internet-Trace_endpointAnalysis.txt**.

The following command will generate the output file containing the endpoint analysis of the provided packet capture file.



```
jerm@jerm-VirtualBox:~/Desktop/Assignment 6/Assignment 6$ sudo ./a.out internet_trace.pcap internetTrace_endpointAnalysis.txt
[sudo] password for jerm:
Total packets: 496
Total bytes: 405928

Time difference between first/last packet: 3.170s
Packets per protocol
TCP: 492, UDP: 4, ICMP: 0

IP end host analysis written in "internetTrace_endpointAnalysis.txt"
In (IP address / # of packets / # of bytes order)
Packets per application
FTP: 0, SSH: 0, DNS: 4, HTTP: 492

Average packet size: 8188
Average packet inter-arrival time: 6.391ms
```

Figure 3: *Executing question3.cpp*

The endpoint analysis (the number of packet and total bytes of each end host) is provided in **internetTrace_endpointAnalysis.txt**. Other required categories of analyzed data are shown in Figure 3 above. The categories of analyzed data will be printed out on the terminal upon execution of the program.