Politecnico di Torino

HTTP Processing with libpcap

Fulvio Risso

November 13, 2023

Acknowledgments

The author would like to thank all the people who contributed to this document.

1 Description

Given a computer equipped with a packet capture library (e.g., `libpcap` or `WinPcap`), write a program in C/C++ language that:

- Captures all the packets generated and received by the host;
- Writes, per each packet, a single line on screen reporting the following information (in case some information are not available, such as the PORT in case of a packet that is neither UDP nor TCP, please leave the field blank):

```
timestamp  MAC_src -> MAC_dst  IP_src -> IP_dst  Protocol  PORT_src -> PORT_dst
```

- Check if the TCP the destination port of the packet is equal to '80'; in this case:
  - Check if the packet contains an HTTP request (e.g., a `POST/GET` command)
  - In this case, extract the URL contained in the packet (e.g., `www.cnn.com`) and print it on screen, after the data mentioned before.

Please note that the URI of an HTTP request message can be specified in different forms (e.g. absolute URI, absolute path, etc.) and some of them may refer to the `Host` header field. For the purpose of this exercise the student is asked to print on screen a single line that concatenates the text present in the `Host` field with the URL contained on the `Request` line, such as:

```
www.cnn.com/weather
```

1.1 Optional: parse the message across multiple HTTP packets

The student is suggested to complete the same exercise by considering that the HTTP payload can be split across multiple packets.

2 Documentation and additional suggestions

2.1 Capture library

In order to capture traffic on your host, your operating system needs to include a packet capture library, such as `libpcap` on Unix or `npcap` (a fork of `WinPcap`, which is no longer maintained) on Windows. Focusing on Linux, remember that you need to install also the development libraries such as `libpcap-dev` if you want to create a new capture program.

2.2 Documentation for the libpcap API

A rather complete documentation of the `libpcap` capture library is available on the WinPcap website, which includes also programming samples, at the following address: `https://www.winpcap.org/docs/docs_412/html/main.html`. Although some new APIs have been added recently (e.g., `pcap_create()`, `pcap_activate()` and more, replacing existing functions such as `pcap_open_live()`), this documentation still uses the 'old' API, which looks more compact and it still preferred nowadays.

2.3 Reading the packet data

The packet capture library exports a set of primitives that allow the user software to receive the entire packet as it is received by the network interface card (i.e., full packet dump). This data is formatted as a plain buffer; you need to know the format of each protocol in order to parse the packet and check what is contained inside. Please refer to the proper documentation (e.g., RFCs) for the protocol you need; a brief summary is available at the following website: `http://www.networksorcery.com/`.

2.4 Byte ordering

Please note that the information contained in the packet buffer is written in *network byte order* (which is *big-endian*), while Intel machines work the opposite way (*little-endian*). Therefore, all the fields that need to be read as numbers (e.g., TCP/UDP ports) need to be translated in the proper byte order before being able to operate on them (e.g., checking their value). In this case, it is strongly suggested to use the functions `ntohX()` (available in the C standard library) in order to convert numbers to the right format.

More information on byte ordering is available at `http://en.wikipedia.org/wiki/Endiannes`.

2.5 Packet size and GRO

Currently, Linux usually has features such as Generic Receive Offloading, which performs TCP reassembly as one of the early network processing steps, often even in the NIC hardware. Therefore it is likely that your program will receive packets that exceed 1500B MTU, and remember that packets on the wire are usually compliant with the 1500B MTU rule.

2.6 Packet size and captured data

Remember that your code needs always to check if the given offset in the packet if valid before loading data from the packet buffer. This can be done by checking that the actual offset is less than the packet length. However, remember that in some cases only a snapshot of the packet is captured and brought to the user-level program, which requires the software to perform the previous check against the *captured length* variable, instead of the *packet length* one[1].

3 Code skeleton and compile instructions

The following code snippet can be used to begin the lab starting with a minimal, running example in `libpcap`.

```
#include <stdio.h>            /* Standard C include file for I/O functions */
#include <time.h>             /* Include file for time manipulation functions */
#include <linux/if_ether.h>   /* Definition of the 'ethhdr' structure */
#include <pcap.h>             /* Include files for libpcap functions */

/* Prototype of the packet handler */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *
    pkt_data);
```

[1]Both variables are available as part of the `struct pcap_pkthdr` defined by the libpcap/WinPcap library.

```
int main(int argc, char **argv)
{
    pcap_t *adhandle;
    char errbuf[PCAP_ERRBUF_SIZE];

    if (argc != 2)
    {
        printf("Usage: %s network_device_name (e.g., eth0)\n\n", argv[0]);
        return -1;
    }

    /* Open the capture device */
    if ( (adhandle= pcap_open_live(argv[1],     // name of the device
                          65536,            // portion of the packet to capture
                                            // 65536 guarantees that the whole
                                               packet is captured
                          PCAP_OPENFLAG_PROMISCUOUS,    // promiscuous mode
                          1000,             // read timeout, 1 second
                          errbuf            // error buffer
                          ) ) == NULL)
    {
        fprintf(stderr,"Unable to open the adapter: either %s is not supported by
            libpcap, ", argv[1]);
        fprintf(stderr, "or you do not have 'superuser' privileges\n\n");
        return -1;
    }

    printf("\nlistening on %s...\n", argv[1]);

    /* start the capture */
    pcap_loop(adhandle, 0, packet_handler, NULL);

    pcap_close(adhandle);
    return 0;
}


/* Callback function invoked by libpcap for every incoming packet */
void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *
    pkt_data)
{
    struct tm *tm;
    char timestr[32];
    time_t local_tv_sec;
    struct ethhdr *ethptr;

    /* Cast the packet buffer into a pointer to an Ethernet frame */
    ethptr = (struct ethhdr *) pkt_data;

    /* Convert the timestamp to a readable format */
    local_tv_sec = header->ts.tv_sec;
    tm= localtime(&local_tv_sec);
    strftime(timestr, sizeof timestr, "%Y-%m-%d %H:%M:%S", tm);

    printf("%s.%.6ld len:%d - ", timestr, header->ts.tv_usec, header->len);

    if (ntohs(ethptr->h_proto) == 0x800)
        printf("IP packet\n");
    else
```

```
        printf("Non IP packet\n");
}
```

This code snipped is attached to the PDF file as "`snippets/pcap_skeleton.c`"

In order to compile the previous example, you can use this command:

```
crownlabs@netlab:~$ gcc pcap_skeleton.c -lpcap -o pcap_skeleton
```

where '`-o pcap_skeleton`' creates an executable named `pcap_skeleton` in your local folder.

The program can be started with the name of a network interface as first parameter, e.g., `enp1s0`, and remembering that packet capture programs require administrative privileges (i.e., `sudo`), as in the following example:

```
crownlabs@netlab:~$ sudo ./pcap_skeleton enp1s0
```