VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IPK – second project

Varianta ZETA: Sniffer paketů

May 3, 2020

Norbert Pócs

Contents

1 Introduction

The main purpose of the project was to create a program capable of catching TCP and UDP packets on the network and printing them out in a similar format to the one of wireshark's. Packet parsing was complicated because of different possible header types.

2 Design

2.1 Packet buffer timout

"If, when capturing, packets are delivered as soon as they arrive, the application capturing the packets will be woken up for each packet as it arrives, and might have to make one or more calls to the operating system to fetch each packet. If, instead, packets are not delivered as soon as they arrive, but are delivered after a short delay (called a "packet buffer timeout"), more than one packet can be accumulated before the packets are delivered, so that a single wakeup would be done for multiple packets, and each set of calls made to the operating system would supply multiple packets, rather than a single packet. This reduces the per-packet CPU overhead if packets are arriving at a high rate, increasing the number of packets per second that can be captured. The packet buffer timeout is required so that an application won't wait for the operating system's capture buffer to fill up before packets are delivered; if packets are arriving slowly, that wait could take an arbitrarily long period of time." [1]

2.2 inet_ntop

"This function converts the network address structure src in the af address family into a character string. The resulting string is copied to the buffer pointed to by dst, which must be a non-null pointer. The caller specifies the number of bytes available in this buffer in the argument size." [2]

2.3 ntohs

"The htons() function converts the unsigned short integer hostshort from host byte order to network byte order." [3]

3 Implementation

The program arguments are parsed using `getopt_long`. If no interface parameter was given, a list of the available interfaces are printed out. A pcap session is then opened using `pcap_open_live` on the given interface. A pocket buffer timeout[2.1] is set on the session with a value of 5 seconds. According to the set filter program parameters (as a specific port number or showing only TCP or UDP packets) a filter is set on the already opened pcap session. The filter first needs to be compiled using `pcap_compile`, then set using `pcap_setfilter`. `pcap_dispatch` is being used to read *n* number of packets.

3.1 Callback

The loop `pcap_dispatch` is calling a callback to process an incoming packet. The callback function should have 3 arguments: user, packet header and the pointer to the raw data.

Time and packet length can be read out from the packet header [4].
The raw data holds the headers of different layers and the payload of the packet if any. [5]
To obtain link layer header type, the program calls `pcap_datalink`. Return number [6] 113 means linux cooked header [7] and 1 means Ethernet header [8]. The type of the link layer header depends on which interface we are sniffing on. For this program, one important information is stored in the link layer header which is the version of the IP address. [9] The next header under the link layer is the IP header.

The IP header contains the source address, destination address and the protocol type. [10] The IPv6 header differ from version 4. [11]

The upcoming header is the protocol header, which in this case can be TCP or UDP.

3.2 Program flow

When a packet arrives, the callback is called (This is not actually correct, because the set pocket buffer timeout[2.1]). Information is gathered from the frame headers in the callback function `read_packet`. The information is used in the head line of the output. The format of the head line is:
`time IP|FQDN : port > IP|FQDN : port.`
The head line is printed out indicating the start of the packet and then the `print_data` function is called to print out the header parts of the packet, then the same function is called again to print out the payload if any is present in the packet. By the header parts of the packet should be understood the Link Layer header, IP header and the Protocol header. The headers and the payload are separated by a new line. The data line has a format of this:
`n: hexa-data readable-data` where n is a hexadecimal value representing the number of the first byte on the line, hexa-data is the data printed in hexadecimal value and readable-data is the same data as before, but in a readable character format (ASCII).

3.3 Printing data

A function is implemented for printing out data, called `print_data`. The function iterates through the given array of length `data_size`. The values are printed out by 16. If the array at the last iteration has less data than 16, then the remaining data is printed out without completing the line.

3.4 Getting data from headers

To get the source and destination address from the IP header `inet_ntop` is used[2.2].

To read the port number of the source and destination `ntohs` function is used[2.3].

To get domain name from the source and destination address `getaddrinfo` is used in the first place to get an addrinfo structure, what is then passed to the function `getnameinfo`, what returns the required domain name if found. If no domain name is found by `getnameinfo`, then the IP address is printed out, otherwise the domain name is printed out.

4 Inspiration

A few information sources were used as inspiration for the program. [12] [13] [14]

5 Testing

I used the KISS principle for testing. I started wireshark and the ipk-sniffer at the same time, then I compared the packets. Curl was used to generate IPv6 packets.

References

[1] The Tcpdump Group: Manpage of PCAP, https://www.tcpdump.org/manpages/pcap.3pcap.html

[2] Michael Kerrisk: INET_NTOP(3) Linux Programmer's Manual, http://man7.org/linux/man-pages/man3/inet_ntop.3.html

[3] Michael Kerrisk: BYTEORDER(3) Linux Programmer's Manual, http://man7.org/linux/man-pages/man3/htons.3.html

[4] The WinPcap Team: pcap_pkthdr Struct Reference, https://www.winpcap.org/docs/docs_412/html/structpcap__pkthdr.html

[5] Richie Slocum: PCAP format, https://github.com/hokiespurs/velodyne-copter/wiki/PCAP-format

[6] The Tcpdump Group: LINK-LAYER HEADER TYPES, https://www.tcpdump.org/linktypes.html

[7] Arthur Ketels and M.J.G. van den Molengraft: sll_header Struct Reference, http://docs.ros.org/lunar/api/soem/html/structsll__header.html

[8] Hector Martin and Andre Heider: ethhdr Struct Reference, https://docs.huihoo.com/doxygen/linux/kernel/3.7/structethhdr.html

[9] Fred N. van Kempen and Donald Becker and Alan Cox and Steve Whitehouse: f_ether.h, https://elixir.bootlin.com/linux/v4.9/source/include/uapi/linux/if_ether.h#L46

[10] Network Sorcery, Inc.: IP - Internet Protocol, http://www.networksorcery.com/enp/protocol/ip.htm#Protocol

[11] Network Sorcery, Inc.: IPv6 - Internet Protocol version 6 http://www.networksorcery.com/enp/protocol/ipv6.htm

[12] oananiculaescu: Analyse a tcpdump capture using libpcap in C, https://elf11.github.io/2017/01/22/libpcap-in-C.html

[13] Silver Moon: C Packet Sniffer Code with libpcap and linux sockets, https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/

[14] Mohan Raman: pcapstreamer – A packet dumper, https://mohan43u.wordpress.com/tag/linux-cooked-header/