Vysoké Učení Technické v Brně
Fakulta informačních technologií

Documentation for IPK - Project 2

Packet sniffer Implementation

Marek Sarvaš (xsarva00) Brno, 03.05.2020

Contents

1 Preface

Documentation for packet sniffer implemented in C++ language with libraries for manipulating with packets, for necessary header structures such as ethernet header, ip header tcp header etc., namely **pcap.h, netinet/ip.h, netinet/ip6.h, netinet/tcp.h** etc.. Programme is sniffing packets using IPv4/IPv6 and UDP/TCP protocol on various ports.

2 Theory

Transport layer

Is 4th layer which transports application-layer messages between application endpoints using TCP and UDP protocols(in the internet). It breaks application messages into segments i.e. packets and sends them into internet layer where the recieving side reassembles them and passes to application layer.

Packet

Packet is an unit that carries data over network, it represents the smallest amount of data that can be transferred over a network at once. It contains control information(source destination addresses, error detection and correction etc.) and the data it is carrying. User data are encapsulated between header and trailer where control information are carried.

TCP

It is connection-oriented protocol in which the connection between client and server is established before any data is sent. TCP uses three way handshake for better error detection and reliability but adds on latency. The minimum size of header is 20 bytes and maximum 60 bytes where main segments used in this project where *source port* and *destination port*, each of them takes 16 bits.

UDP

UDP is another transport layer protocol but is unreliable and connectionless unlike TCP. It does not use three way handshake because there is no need to establish connection before data transfer. Using UDP performance is heigher it does not check for errors, drops delayed packets and has better latency than TCP. It is highly used in pc gaming or video communication. UDP header has fixed length to 8 bytes and contains necessary information for this project such as *source port* and *destination port* with same 16 bits length as TCP.

3 Implementation

Sniffer is implemented in one file ipk-sniffer.cpp, whole program is divided into few functions and main function. Compilation and how to run the sniffer is documented in readme.md .

3.1 Main()

First part of main function is for parsing given arguments using **check_args** function. If interface was not given as argument all interfaces are printed in loop.
If programme got interface (or other optional arguments such as port, tcp, etc.), opens given interface for sniffing using **pcap_open_live**, then check correct link-layer header type using **pcap_datalink** and failed when other than DLT_LINUX_SLL or DLT_EN10MB occures. On success compile given filter composed from given programme arguments - tcp, udp, port. If compiled successfuly filter is applied on interface handler.
For actual sniffing pcap _loop function is used with arguments such as interface handler, number of packets to be sniffed (stored in argument structure), callback function (documented below). There is no time limit in which packet has to be sniffed, because if user wants to sniffed eg. 2 packets programme will run until 2 packets are sniffed. After wanted number of packets is sniffed programme closes interface handler and frees allocated memory.
Otherwise if interface, where tcp/udp packets could not be sniffed was given, sniffer will run infinitely until interruption (eg.: with CTRL+C).

3.2 Create_filter

Creates a string filter using tcp, udp, port number from values given as programme arguments. Because programme is sniffing only tcp or udp packets default filter will be set to **"(proto tcp) or (proto udp)"**, this filter is set either if programme is run with "–tcp" and "–udp" arguments or without them. "Proto" in filter means that only IPv4 and IPv6 packets will be sniffed and default tcp/udp filter means only tcp and udp packets will be sniffed. Port number is added into filter with **"and"** e.g.: **"(proto tcp) and (port 443)"**.

3.3 Callback fucntion

Function passed into pcap_loop and is called for every packet sniffed. Is responsible for parsing packet to get necessary information such as: time, protocol of packet, source and destination ports and ip addresses, resolving ip addresses into names and printing these information and whole packet on standard output.

- **Ethernet type**

  Firstly gets ethernet header from packet data and chooses if sniffed packet uses IPv4 or IPv6. According to IP gets IPv4/IPv6 header in which are necessary information for future use such as IP source address and IP destination address, protocol(TCP/UDP) and length of IP header. For IP addresses **inet_ntop** function is used with **AF_INET** argument for IPv4 and **AF_INET6** for IPv6.

- **Source and destination names using cache**

  If the IP address is not in DNS cache, it is resolved to FQDN using **getaddrinfo** function to get rid of IPv4-IPv6 dependencies. Needed information are stored in **addrinfo** structure. This structure is next used in **getnameinfo** function to get FQDN from **ai_addr** variable. **Getnameinfo** function is used with **NI_NAMEREQD** flags which returns error if IP address cannot be resolved, this is causing small memory leak according to *valgrind* even though everything is freed same as when IP address is resolved into FQDN. In both cases(IP address can or cannot be resolved) IP address is stored in DNS cache. If IP is already in cache none of above functions is called and FQDN is got from there. DNS cache is implemented as unordered map where *key* is IP address and *value* is either resolved FQDN or same IP address. Cache is implemented because **getaddrinfo** and **getnameinfo** are producing another packets when called, then sniffer catches these new generated packets this can cause loops and influence output.

- **Print packet**

  Firstly is printed used filter on standard output and information if source and destination address is resolved or used from cache on standard error output. Then is printed time when packet was sniffed which is got from packet header and computed to hh:mm:ss.ffff format using **convert_time** function. Then IPv4/IPv6 source/destination IP addresses with corresponding ports. For port numbers **ntohs** function is used. Next is printed whole packet in bytes in hexadecimal and ascii format, 16 bytes per row, every row starting with hexadecimal number which represents how many bytes were printed before current row. Format of packet ouput is same as packet representation in **Wireshark**.

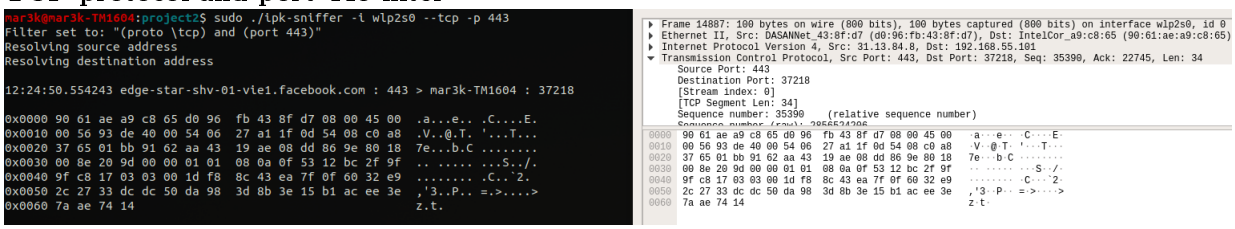
4 Testing

For testing purpose **Wireshark** application was used. Sniffer was tested only manually for filters number of packets etc., no automated testing was involved, for different configuration. Output of implemented ipk-sniffer was compared with the same packet in Wireshark for same time, source and destination ports and data of packet.

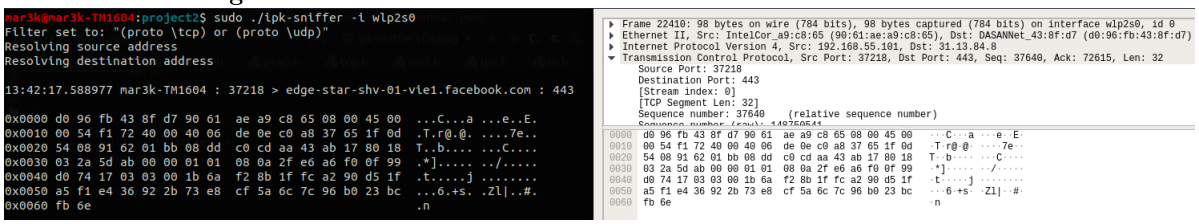
4.1 Basic filter testing

This involved testing different configuration of filter i.e. combination of **tcp, udp** and **port number**. Also manually added in code filter for IPv6, these packets was tested mainly by using ssh which produced IPv6 packets.

- **TCP protocol and port 443 filter**



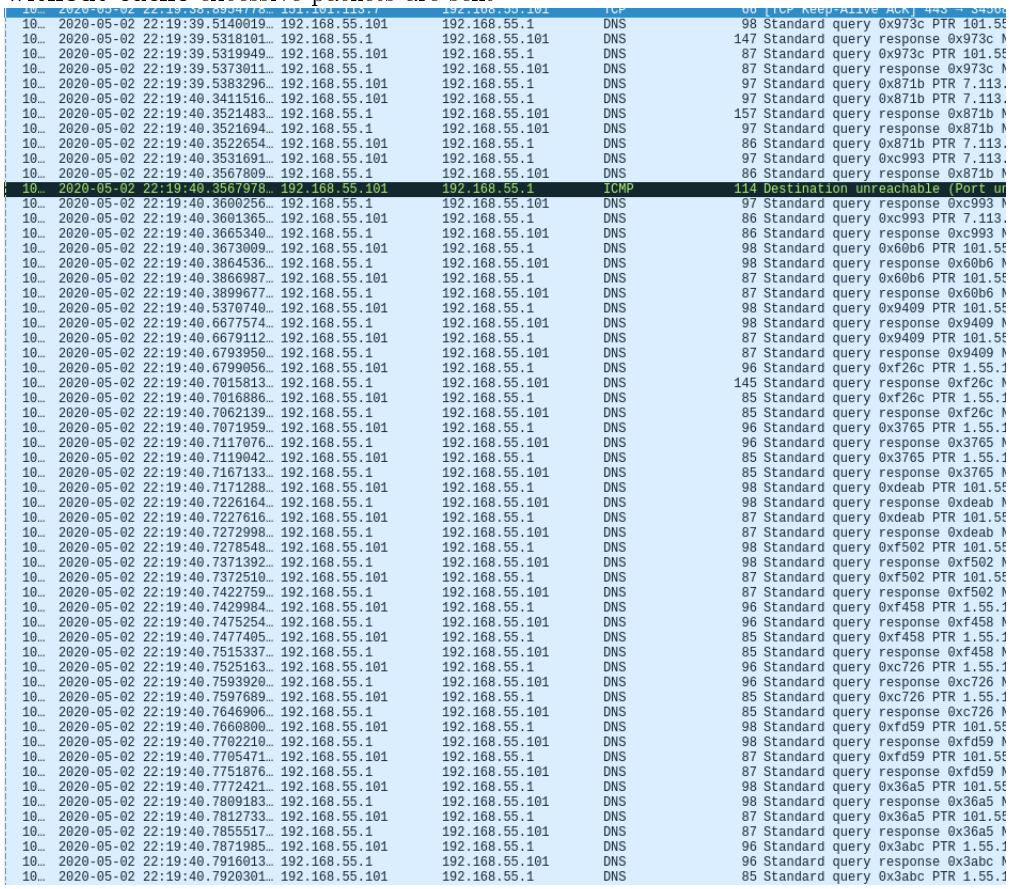
- **Without filter arguments** - TCP and UDP are default



4.2 Name resolving with cache

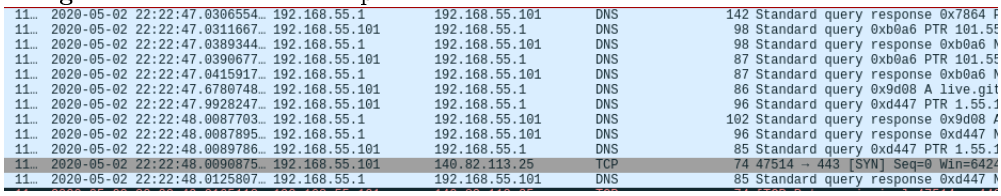
When resolving FQDN from ip address using *getaddrinfo()* and *getnameinfo()* additional packets are sent. This fact with combination of more packet sniffing e.g.: argument -n 10 can result into repeatedly sniffing only these packets for name resolving.
Examples are run with ***-n 20*** configuration and tested in **Wireshark** application.

- **without cache** excessive packets are sent



10…	2020-05-02 22:19:39.5140019…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0x973c PTR 101.55
10…	2020-05-02 22:19:39.5318101…	192.168.55.1	192.168.55.101	DNS	147 Standard query response 0x973c N
10…	2020-05-02 22:19:39.5319949…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0x973c PTR 101.55
10…	2020-05-02 22:19:39.5373011…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0x973c N
10…	2020-05-02 22:19:39.5383296…	192.168.55.101	192.168.55.1	DNS	97 Standard query 0x871b PTR 7.113.
10…	2020-05-02 22:19:40.3411516…	192.168.55.101	192.168.55.1	DNS	97 Standard query 0x871b PTR 7.113.
10…	2020-05-02 22:19:40.3521483…	192.168.55.1	192.168.55.101	DNS	157 Standard query response 0x871b N
10…	2020-05-02 22:19:40.3521694…	192.168.55.1	192.168.55.101	DNS	97 Standard query response 0x871b N
10…	2020-05-02 22:19:40.3522654…	192.168.55.101	192.168.55.1	DNS	86 Standard query 0x871b PTR 7.113.
10…	2020-05-02 22:19:40.3531691…	192.168.55.101	192.168.55.1	DNS	97 Standard query 0xc993 PTR 7.113.
10…	2020-05-02 22:19:40.3567809…	192.168.55.1	192.168.55.101	DNS	86 Standard query response 0x871b N
10…	2020-05-02 22:19:40.3567978…	192.168.55.101	192.168.55.1	ICMP	114 Destination unreachable (Port un
10…	2020-05-02 22:19:40.3600256…	192.168.55.1	192.168.55.101	DNS	97 Standard query response 0xc993 N
10…	2020-05-02 22:19:40.3601365…	192.168.55.101	192.168.55.1	DNS	86 Standard query 0xc993 PTR 7.113.
10…	2020-05-02 22:19:40.3665340…	192.168.55.1	192.168.55.101	DNS	86 Standard query response 0xc993 N
10…	2020-05-02 22:19:40.3673009…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0x60b6 PTR 101.55
10…	2020-05-02 22:19:40.3864536…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0x60b6 N
10…	2020-05-02 22:19:40.3866987…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0x60b6 PTR 101.55
10…	2020-05-02 22:19:40.3899677…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0x60b6 N
10…	2020-05-02 22:19:40.5370740…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0x9409 PTR 101.55
10…	2020-05-02 22:19:40.6677574…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0x9409 N
10…	2020-05-02 22:19:40.6679112…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0x9409 PTR 101.55
10…	2020-05-02 22:19:40.6793950…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0x9409 N
10…	2020-05-02 22:19:40.6799056…	192.168.55.101	192.168.55.1	DNS	96 Standard query 0xf26c PTR 1.55.1
10…	2020-05-02 22:19:40.7015813…	192.168.55.1	192.168.55.101	DNS	145 Standard query response 0xf26c N
10…	2020-05-02 22:19:40.7016886…	192.168.55.101	192.168.55.1	DNS	85 Standard query 0xf26c PTR 1.55.1
10…	2020-05-02 22:19:40.7062139…	192.168.55.1	192.168.55.101	DNS	85 Standard query response 0xf26c N
10…	2020-05-02 22:19:40.7071959…	192.168.55.101	192.168.55.1	DNS	96 Standard query 0x3765 PTR 1.55.1
10…	2020-05-02 22:19:40.7117076…	192.168.55.1	192.168.55.101	DNS	96 Standard query response 0x3765 N
10…	2020-05-02 22:19:40.7119042…	192.168.55.101	192.168.55.1	DNS	85 Standard query 0x3765 PTR 1.55.1
10…	2020-05-02 22:19:40.7167133…	192.168.55.1	192.168.55.101	DNS	85 Standard query response 0x3765 N
10…	2020-05-02 22:19:40.7171288…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0xdeab PTR 101.55
10…	2020-05-02 22:19:40.7226164…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0xdeab N
10…	2020-05-02 22:19:40.7227616…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0xdeab PTR 101.55
10…	2020-05-02 22:19:40.7272998…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0xdeab N
10…	2020-05-02 22:19:40.7278548…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0xf502 PTR 101.55
10…	2020-05-02 22:19:40.7371392…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0xf502 N
10…	2020-05-02 22:19:40.7372510…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0xf502 PTR 101.55
10…	2020-05-02 22:19:40.7422759…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0xf502 N
10…	2020-05-02 22:19:40.7429984…	192.168.55.101	192.168.55.1	DNS	96 Standard query 0xf458 PTR 1.55.1
10…	2020-05-02 22:19:40.7475254…	192.168.55.1	192.168.55.101	DNS	96 Standard query response 0xf458 N
10…	2020-05-02 22:19:40.7477405…	192.168.55.101	192.168.55.1	DNS	85 Standard query 0xf458 PTR 1.55.1
10…	2020-05-02 22:19:40.7515337…	192.168.55.1	192.168.55.101	DNS	85 Standard query response 0xf458 N
10…	2020-05-02 22:19:40.7525163…	192.168.55.101	192.168.55.1	DNS	96 Standard query 0xc726 PTR 1.55.1
10…	2020-05-02 22:19:40.7593920…	192.168.55.1	192.168.55.101	DNS	96 Standard query response 0xc726 N
10…	2020-05-02 22:19:40.7597689…	192.168.55.101	192.168.55.1	DNS	85 Standard query 0xc726 PTR 1.55.1
10…	2020-05-02 22:19:40.7646906…	192.168.55.1	192.168.55.101	DNS	85 Standard query response 0xc726 N
10…	2020-05-02 22:19:40.7660800…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0xfd59 PTR 101.55
10…	2020-05-02 22:19:40.7702210…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0xfd59 N
10…	2020-05-02 22:19:40.7705471…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0xfd59 PTR 101.55
10…	2020-05-02 22:19:40.7751876…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0xfd59 N
10…	2020-05-02 22:19:40.7772421…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0x36a5 PTR 101.55
10…	2020-05-02 22:19:40.7809183…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0x36a5 N
10…	2020-05-02 22:19:40.7812733…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0x36a5 PTR 101.55
10…	2020-05-02 22:19:40.7855517…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0x36a5 N
10…	2020-05-02 22:19:40.7871985…	192.168.55.101	192.168.55.1	DNS	96 Standard query 0x3abc PTR 1.55.1
10…	2020-05-02 22:19:40.7916013…	192.168.55.1	192.168.55.101	DNS	96 Standard query response 0x3abc N
10…	2020-05-02 22:19:40.7920301…	192.168.55.101	192.168.55.1	DNS	85 Standard query 0x3abc PTR 1.55.1

- **using cache** number of these packets is reduced to minimum



11…	2020-05-02 22:22:47.0306554…	192.168.55.1	192.168.55.101	DNS	142 Standard query response 0x7864 P
11…	2020-05-02 22:22:47.0311667…	192.168.55.101	192.168.55.1	DNS	98 Standard query 0xb0a6 PTR 101.55
11…	2020-05-02 22:22:47.0389344…	192.168.55.1	192.168.55.101	DNS	98 Standard query response 0xb0a6 N
11…	2020-05-02 22:22:47.0390677…	192.168.55.101	192.168.55.1	DNS	87 Standard query 0xb0a6 PTR 101.55
11…	2020-05-02 22:22:47.0415917…	192.168.55.1	192.168.55.101	DNS	87 Standard query response 0xb0a6 N
11…	2020-05-02 22:22:47.6780748…	192.168.55.101	192.168.55.1	DNS	86 Standard query 0x9d08 A live.git
11…	2020-05-02 22:22:47.9928247…	192.168.55.101	192.168.55.1	DNS	96 Standard query 0xd447 PTR 1.55.1
11…	2020-05-02 22:22:48.0087703…	192.168.55.1	192.168.55.101	DNS	102 Standard query response 0x9d08 A
11…	2020-05-02 22:22:48.0087895…	192.168.55.1	192.168.55.101	DNS	96 Standard query response 0xd447 N
11…	2020-05-02 22:22:48.0089786…	192.168.55.101	192.168.55.1	DNS	85 Standard query 0xd447 PTR 1.55.1
11…	2020-05-02 22:22:48.0090875…	192.168.55.101	140.82.113.25	TCP	74 47514 → 443 [SYN] Seq=0 Win=6424
11…	2020-05-02 22:22:48.0125807…	192.168.55.1	192.168.55.101	DNS	85 Standard query response 0xd447 N

5 References

[1] James F. Kurose, Keith W. Ross. *Computer networking : a top-down approach.* -6th edition

[2] Protocol Numbers,
`https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml`

[3] Linux manual,
`https://linux.die.net/man/`

[4] WinPcap Unix-compatible Functions,
`https://www.winpcap.org/docs/docs_40_2/html/group__wpcapfunc.html`

[5] LibPcap,
`https://www.tcpdump.org/pcap.html`

[6] tcpdump pcap_loop,
`https://www.tcpdump.org/manpages/pcap_loop.3pcap.html`

[7] Packet filter,
`https://www.tcpdump.org/manpages/pcap-filter.7.html`