POČÍTAČOVÉ KOMUNIKACE A SÍTĚ

2021/2022

PROTOKOL K PROJEKTU

Varianta ZETA: Sniffer paketů

Václav Valenta (xvalen29)

Brno, 24.04. 2022

1. Obsah

Obsah

2. Základní funkcionalita

2.1. Překlad a spuštění

Překlad projektu se provádí příkazem make, který vytvoří spustitelný soubor s názvem *ipk-sniffer*. Program se spouští následovně:

```
./ipk-sniffer [-i rozhraní | --interface rozhraní] {-p --port} {[--tcp|-t] [--udp|-u] [--arp] [--icmp] } {-n num}
```

2.2. Přepínače

Pomocí přepínače *-i* nebo *--interface* definujeme rozhraní. V případě, že rozhraní není zadáno, program vypíše seznam všech rozhraní.

Přepínač *-p* nebo *--port* slouží ke specifikaci určitého portu, na kterým se má komunikace zachytávat. Pokud port není specifikován, program bude zachytávat komunikaci na všech portech.

Další přepínače určují typ paketů, které se mají vypisovat *(--tcp* | *-t, --udp* | *-u, --arp, --icmp*). Pokud tyto přepínače nejsou uvedeny, nebo jejich kombinace je neplatná, uvažují se všechny zmíněné typy. Zadaný port se nemění.

Poslední přepínač -n určuje kolik paketů program vypíše.

3. Implementace

3.1. Zpracování argumentů

Program po spuštění začne zpacovávat zadané vstupní argumenty pomocí funkce *getopt_long()*. Pokud zadané argumenty neodpovídají formátu popsanému výše, program vypíše chybové hlášení a ukončí činnost.

Pro lepší přehlednost jsou všechny chybová hlášení v kodě zadefinované v enum struktuře a volané funkcí *print_error(error_t err)*:

```
void print_error(error_t err) {
    switch (err) {
        case PCAP_DEVS_ERR:
            fprintf(stderr, "pcap_findalldevs error");
            break;
        ...

typedef enum error_enum {
    PCAP_DEVS_ERR,
    INVALID_INTERFACE_ERR,
    ...
} error_t;
```

3.2. Připojení rozhraní

V případě, že rozhraní není definované, program vypíše seznam všech aktivních rozhraní a ukončí činnost:

```
student@student-vm:~/Documents$ sudo ./ipk-sniffer
enp0s3
lo
any
bluetooth-monitor
nflog
nfqueue
student@student-vm:~/Documents$
```

V opačném případě se pokusí na rozhraní připojit – v případě úspěchu program pokračuje dále.

3.3. Filtrace

Filtry jsou parsovány ze vstupu – konkrétně se jedná o bool hodnoty *tcp, udp, icmp* a *arp*. Pokud se jedná o validní kombinaci, jsou filtry logicky spojeny a zapsány do řetězce filter_string. V opačném případě je tento řetězec prázdný. Nakonec je připojen i filtr, který udává číslo zadaného portu.

```
// Add filter for port
if (port != -1) {
    // Check invalid filter combination
    if (arp || icmp) {
        strcpy(filter_string, "");
        sprintf(port_filter, "port %d", port);
    } else {
        sprintf(port_filter, " and port %d", port);
    }
}
strcat(filter_string, port_filter);
```

3.4. Zpracování paketů – čas

Po úspěšném vytvoření a aplikování filtru je zavolána funkce *pcap_loop*, která volá funkci *sniffing*. Tato funkce ihned vypíše čas ve formátu *yyyy-MM-dd'T'HH:mm:ss.SSSZ*. Pro dosažení požadovaného výsledku byly použity následující struktury:

```
time_t rawtime;
struct tm *info;
struct timeval tv;
```

Milisekundová část je vypočítána následujícím způsobem:

```
int ms = round(tv.tv_usec / 1000);
if (ms >= 1000) {
    ms -= 1000;
    tv.tv_sec++;
}
```

3.5. Zpracování paketů – typ

Kromě času se také ihned vypíše zdrojová, cílová MAC adresa a také délka rámce.

Dalším úkolem programu je zjištění typu paketu. K tomu využívá funkci *ntohs*.

```
 void sniffing(u_char *user, const struct pcap_pkthdr *header, const u_char
*packet) {
    struct ether_header *eth_header;
    eth_header = (struct ether_header *)packet;

    ...
    if (ntohs(eth_header->ether_type) == ETHERTYPE_ARP) {
        ...
    } else if (ntohs(eth_header->ether_type) == ETHERTYPE_IP) {
        ...
    }
    ...
 }
```

Následné zpracování se odvíjí právě podle tohoto typu. V případě typu ETHERTYPE_IP je třeba ještě zjistit ještě příslušný typ IPv4 a obdobně v případě ETHERTYPE_IPV6.

Na zjištění portů se používá funkce *ntohs,* na IPv4 adresy funkce *inet_ntoa*.

3.6. Zpracování paketů – výpis dat

Poslední věc, kterou programu ještě zbývá udělat, je vypsat data – a to jak v hexadecimální podobě, tak i v ASCII podobě. Program zavolá funkci *print_data*.

Funkce počítá aktuální offset a podle velikosti zbývajících dat rozhoduje, zda bude pokračovat dále, nebo budou aktuálně zpracovávaná data poslední řádkou výpisu.

```
    // Print whole lines, until the size is smaller than one line
    while (size_left > 16) {
        line_length = 16 % size_left;
        print_hex_ascii(char_data, line_length, offset);
        size_left -= line_length;
        char_data += line_length;
        offset += 16;
    }

    // Print the last line
    print_hex_ascii(char_data, size_left, offset);
```

Funkce na každý zpracovaný řádek volá funkci *print_hex_ascii*. Která data vypíše v obou formátech s potřebnými mezerami.

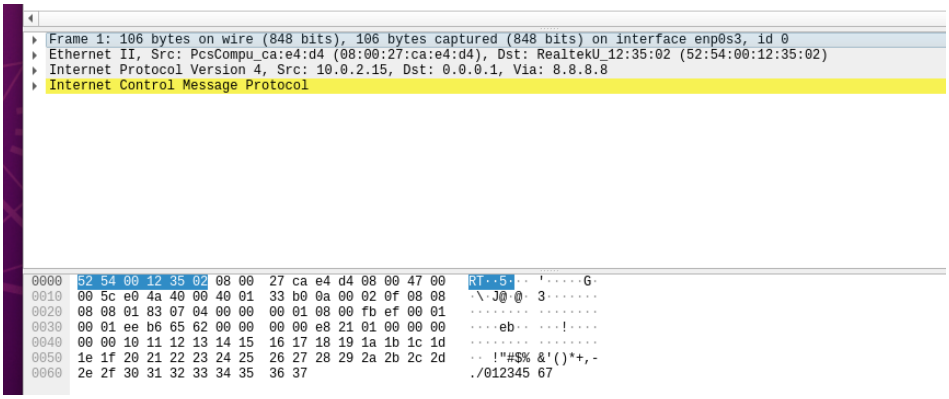
Po zpracování posledního paketu funkce zavolá funcki *pcap_close* a ukončí běh programu.

4. Testování

K testování byl použit virtuální stroj a program Wireshark

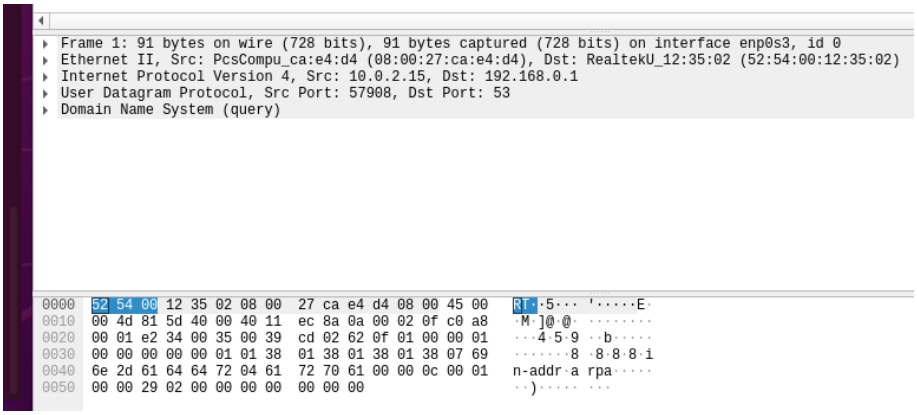
4.1. ICMP

```
student@student-vm:~/Documents$ sudo ./ipk-sniffer -i enp0s3
timestamp: 2022-04-24T22:45:34.397+02:00
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 106 bytes
src IP: 10.0.2.15
dst IP: 8.8.8.8
0x0000: 52 54 00 12 35 02 08 00  27 ca e4 d4 08 00 47 00  RT..5... '.....G.
0x0010: 00 5c e0 4a 40 00 40 01  33 b0 0a 00 02 0f 08 08  .\.J@.@. 3.......
0x0020: 08 08 01 83 07 04 00 00  00 01 08 00 fb ef 00 01  ........ ........
0x0030: 00 01 ee b6 65 62 00 00  00 00 e8 21 01 00 00 00  ....eb.. ...!....
0x0040: 00 00 10 11 12 13 14 15  16 17 18 19 1a 1b 1c 1d  ........ ........
0x0050: 1e 1f 20 21 22 23 24 25  26 27 28 29 2a 2b 2c 2d  .. !"#$% &'()*+,-
0x0060: 2e 2f 30 31 32 33 34 35  36 37                    ./012345 67
```



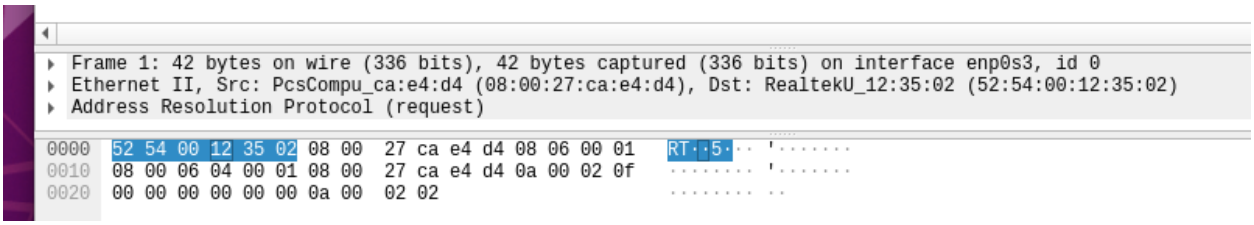
4.2. UDP

```
student@student-vm:~/Documents$ sudo ./ipk-sniffer -i enp0s3 -u
timestamp: 2022-04-24T22:48:47.677+02:00
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 91 bytes
src IP: 10.0.2.15
dst IP: 192.168.0.1
src port: 57908
dst port: 53
0x0000: 52 54 00 12 35 02 08 00  27 ca e4 d4 08 00 45 00  RT..5... '.....E.
0x0010: 00 4d 81 5d 40 00 40 11  ec 8a 0a 00 02 0f c0 a8  .M.]@.@. ........
0x0020: 00 01 e2 34 00 35 00 39  cd 02 62 0f 01 00 00 01  ...4.5.9 ..b.....
0x0030: 00 00 00 00 00 01 01 38  01 38 01 38 01 38 07 69  .......8 .8.8.8.i
0x0040: 6e 2d 61 64 64 72 04 61  72 70 61 00 00 0c 00 01  n-addr.a rpa.....
0x0050: 00 00 29 02 00 00 00 00  00 00 00                 ..)..... ...
```



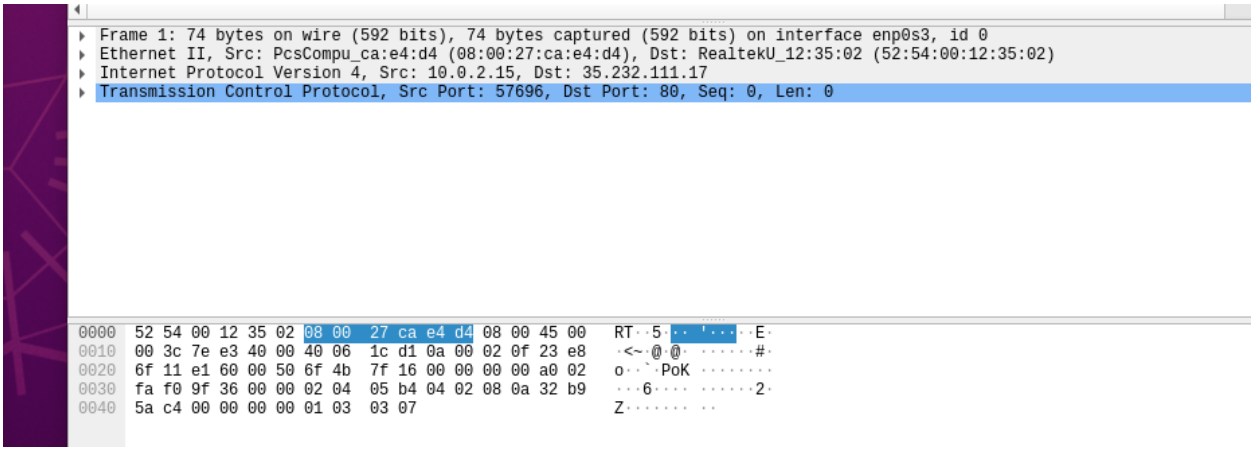
4.3. ARP

```
student@student-vm:~/Documents$ sudo ./ipk-sniffer -i enp0s3 --arp
timestamp: 2022-04-24T22:51:26.883+02:00
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 42 bytes
src IP: 0.1.8.0
dst IP: 39.202.228.212
0x0000: 52 54 00 12 35 02 08 00  27 ca e4 d4 08 06 00 01  RT..5... '.......
0x0010: 08 00 06 04 00 01 08 00  27 ca e4 d4 0a 00 02 0f  ........ '.......
0x0020: 00 00 00 00 00 00 0a 00  02 02                    ........ ..
```



4.4. TCP

```
student@student-vm:~/Documents$ sudo ./ipk-sniffer -i enp0s3 --tcp
timestamp: 2022-04-24T22:54:48.931+02:00
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 74 bytes
src IP: 10.0.2.15
dst IP: 35.232.111.17
src port: 57696
dst port: 80
0x0000: 52 54 00 12 35 02 08 00  27 ca e4 d4 08 00 45 00  RT..5... '.....E.
0x0010: 00 3c 7e e3 40 00 40 06  1c d1 0a 00 02 0f 23 e8  .<~.@.@. ......#.
0x0020: 6f 11 e1 60 00 50 6f 4b  7f 16 00 00 00 00 a0 02  o..`.PoK ........
0x0030: fa f0 9f 36 00 00 02 04  05 b4 04 02 08 0a 32 b9  ...6.... ......2.
0x0040: 5a c4 00 00 00 00 01 03  03 07                    Z....... ..
```



5. Zdroje

5.1. Výpis aktivních rozhraní

Tcpdump [online] <https://www.tcpdump.org/manpages/pcap_findalldevs.3pcap.html> [viewed: 23.04.2022]

Linux Documentation [online] <https://linux.die.net/man/3/pcap_freealldevs> [viewed: 23.04.2022]

5.2. Timestamp

WinPcap documentation [online] <https://www.winpcap.org/docs/docs_40_2/html/group__wpcap__tut4.html> [viewed: 23.04.2022]

Stack Overflow [online] <https://stackoverflow.com/questions/3673226/how-to-print-time-in-format-2009-08-10-181754-811> [viewed: 23.04.2022]

Sumo Logic Doc Hub [online] <https://help.sumologic.com/03Send-Data/Sources/04Reference-Information-for-Sources/Timestamps%2C-Time-Zones%2C-Time-Ranges%2C-and-Date-Formats> [viewed: 23.04.2022]

The Open Group Library [online] <https://pubs.opengroup.org/onlinepubs/9699919799/functions/strftime.html> [viewed: 23.04.2022]

C Standard Library Reference Tutorial [online] <https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm> [viewed: 23.04.2022]

5.3. Připojení rohzraní

Tcpdump [online] <https://www.tcpdump.org/manpages/pcap_open_live.3pcap.html> [viewed: 23.04.2022]

5.4. Filtrace

Tcpdump [online] <https://www.tcpdump.org/manpages/pcap_setfilter.3pcap.html> [viewed: 23.04.2022]

Tcpdump [online] <https://www.tcpdump.org/manpages/pcap_compile.3pcap.html> [viewed: 23.04.2022]

Wireshark documentation [online] <https://www.wireshark.org/docs/wsug_html_chunked/ChCapCaptureFilterSection.html> [viewed: 23.04.2022]

Linux Documentation [online] <https://linux.die.net/man/3/pcap_freecode> [viewed: 23.04.2022]

5.5. MAC adresa

Capturing Our First Packet [online]
<http://yuba.stanford.edu/~casado/pcap/section2.html> [viewed: 24.04.2022]

5.6. Výpis dat

Tcpdump [online] <https://www.tcpdump.org/other/sniffex.c> [viewed: 24.04.2022]

How to code a Packet Sniffer in C with Libpcap on Linux [online]
<https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets>
[viewed: 24.04.2022]

5.7. Zpracování paketů

Using libpcap in C [online] <https://www.devdungeon.com/content/using-libpcap-c#pcap-loop> [viewed: 24.04.2022]

WinPcap documentation [online]
<https://www.winpcap.org/docs/docs_412/html/structpcap__pkthdr.html> [viewed: 24.04.2022]

Wikipedia [online] <https://en.wikipedia.org/wiki/Ethernet_frame> [viewed: 24.04.2022]

netinet/if_ether.h documentation [online]
<https://unix.superglobalmegacorp.com/Net2/newsrc/netinet/if_ether.h.html>
[viewed: 24.04.2022]

Tcpdump [online] <https://samy.pl/packet/MISC/tcpdump-3.7.1/ethertype.h>
[viewed: 24.04.2022]

netinet/in.h documentation [online]
<https://pubs.opengroup.org/onlinepubs/009695399/basedefs/netinet/in.h.html>
[viewed: 24.04.2022]

ether_arp Struct Reference [online]
<http://www.ethernut.de/api/structether__arp.html> [viewed: 24.04.2022]

Jeremiah Mahler - socket-examples [online] <https://github.com/jmahler/socket-examples/blob/master/packets/packets.c> [viewed: 24.04.2022]

Assigned Internet Protocol Numbers [online]
<https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>
[viewed: 24.04.2022]

Sniffer example of TCP/IP packet capture using libpcap [online]
<https://www.tcpdump.org/other/sniffex.c> [viewed: 24.04.2022]

Understanding the IPv6 Header [online]
<https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=3>
[viewed: 24.04.2022]

5.8. Zpracování argumentů

Linux Documentation [online] <https://linux.die.net/man/3/getopt_long> [viewed: 23.04.2022]

Stack Overflow [online] <https://stackoverflow.com/questions/19604413/getopt-optional-arguments> [viewed: 23.04.2022]

Stack Overflow [online] <https://stackoverflow.com/questions/7489093/getopt-long-proper-way-to-use-it> [viewed: 23.04.2022]

Stack Overflow [online] <https://stackoverflow.com/questions/9642732/parsing-command-line-arguments-in-c> [viewed: 23.04.2022]

getopt() function in C to parse command line arguments [online] <https://www.tutorialspoint.com/getopt-function-in-c-to-parse-command-line-arguments> [viewed: 23.04.2022]

Robbins, A. (2004). Linux programming by example (1st ed.). Prentice Hall. ISBN 9780131429642