Vysoké učení technické v Brně
Fakulta informačních technologií

Počítačové komunikace a sítě

2021/2022

Dokumentácia projektu

Varianta ZETA: Sniffer paketov

Peter Ďurica (xduric05)

Brno,24.apríla.2022

1 Obsah

2 Úvod do problematiky

Cieľom projektu je navrhnúť a vytvoriť funkčný sieťový analyzátor IPv4, IPv6 a ARP paketov na určitom sieťovom rozhraní zadaným užívateľom. Tento sieťový analyzátor môže byť napísaný v jazykoch C, C++ alebo C#. Analyzátor bude mať za úlohu odchytávať pakety na sieťovom rozhraní a ich obsah prepisovať do formy zrozumiteľnej pre užívateľa. Užívateľ si môže pri štarte programu zvoliť aký typ paketov má analyzátor spracovávať. Takisto si môže zvoliť aj port, na ktorý alebo z ktorého je paket odoslaný a počet paketov ktorý má analyzátor spracovať. Analyzátor po spracovaní paketu ešte vytlačí na obrazovku jeho obsah bajt po bajte vo forme hexadecimálneho zápisu.

3 Implementácia

3.1 Návrh aplikácie

Aplikáciu som sa rozhodol písať v jazyku C++, pretože už mám skúsenosti z jazyka C a zároveň som mohol použiť rozšírené knižnice pre jazyk C++. Pri návrhu som použil hlavne knižnice `pcap` a `netinet`. Na spracovanie argumentov som využil funkciu `getopt` aj s rozšírením pre dlhé možnosti argumentov. Sieťové rozhrania otváram pomocou `pcap` funkcií. Na čítanie hlavičiek používam štruktúry v knižniciach `netinet`, ktoré mi umožňujú čítať informácie z hlavičiek rýchlo a jednoducho. K transformácii UNIX time do vyžadovaného timestampu využívam knižnicu `time` a jej funkcie.

3.2 Funkcia `main`

Na začiatku funkcie `main` kontrolujem argumenty programu pri spustení, ktoré boli zadané užívateľom. Po vytvorení pomocných premenných pre `getopt` som si napísal štruktúru `option` pre dlhé argumenty.

```
static struct option long_options[] =
{
    {"interface",   required_argument,  NULL,  'i'},
    {"tcp",         no_argument,        NULL,  't'},
    {"udp",         no_argument,        NULL,  'u'},
    {"help",        no_argument,        NULL,  'h'},
    {"arp",         no_argument,        &f_arp,  1},
    {"icmp",        no_argument,        &f_icmp, 1},
    {NULL,          0,                  NULL,    0}
};
```

Obrázok 1: Štruktúra option pre getopt

Aplikácia podporuje argumenty:

1. --interface | -i (názov) : Vyžaduje argument, ktorý označuje sieťové rozhranie. Ak argument nie je zadaný alebo kompletne chýba tak sa vypíšu všetky dostupné rozhrania. Ak rozhranie nie je v zozname dostupných rozhraní tak aplikácia končí.
2. -p (číslo) : Vyžaduje argument, ktorý označuje na ktorom porte môže aplikácia analyzuje pakety. (default všetky)

3. -n (číslo): Vyžaduje argument, ktorý označuje koľko paketov analyzovať. (default 1)
4. --tcp | -t : Aplikácia bude analyzovať TCP pakety
5. --udp | -u : Aplikácia bude analyzovať UDP pakety
6. --arp : Aplikácia bude analyzovať UDP pakety
7. --icmp : Aplikácia bude analyzovať ICMP pakety

Po spracovaní argumentov si aplikácia otvorí zadané sieťové rozhranie pomocou knižnice `pcap`. Po kontrole že sa dané rozhranie správne otvorilo sa kontroluje či vôbec dané rozhranie podporuje ethernet pakety. Ak podporuje tak sa začína kompilácia a aplikácia filtra na rozhranie. Filter sa skladá podľa argumentov zadaných od užívatela. Po správnej aplikácii filtra sa môže začať analýza packetov.

```
//Open device using pcap used code from https://www.tcpdump.org/pcap.html

/*
This document is Copyright 2002 Tim Carstens. All rights reserved.
Redistribution and use, with or without modification, are permitted provided that the following conditions are met:
    Redistribution must retain the above copyright notice and this list of conditions.
    The name of Tim Carstens may not be used to endorse or promote products derived from this document without specific prior written permission.
*/
handle = pcap_open_live(device.c_str(), BUFSIZ, 1, 100, errbuf);
//If device wasnt opened
if (handle == NULL) {
    fprintf(stderr, "Couldn't open device %s: %s\n", device.c_str(), errbuf);
    return(2);
}
//If device cant use ethernet frames
if (pcap_datalink(handle) != DLT_EN10MB) {
    fprintf(stderr, "Device %s doesn't provide Ethernet headers - not supported\n", device.c_str());
    pcap_close(handle);
    return(2);
}
struct bpf_program fp;
//Compiling filter
if (pcap_compile(handle, &fp, filterGen().c_str(), 0, 0) == -1) {
    fprintf(stderr, "Couldn't parse filter %s: %s\n", filterGen().c_str(), pcap_geterr(handle));
    pcap_close(handle);
    return(2);
}
//Installing filter
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s\n", filterGen().c_str(), pcap_geterr(handle));
    pcap_close(handle);
    return(2);
}
```

Obrázok 2: Otvorenie sieťového rozhrania a aplikácia filtra

Po otvorení spojenia s rozhraním sa spúšťa funkcia pre bezpečné ukončenie programu ak užívateľ zadá interrupčný signál `Ctrl+C`. Analýza paketov prebieha vo `for` slučke, ktorá sa vykoná toľkokrát ako zadá užívateľ. Ak je filter správne aplikovaný tak nám funkcia `pcap_next` bude posielať len užívateľom želané pakety. Po prijatí paketu ho aplikácia analyzuje a čaká na prípadný ďalší paket. Po analýze želaného počtu paketov sa rozhranie zatvorí a aplikácia sa ukončí.

3.3 Funkcia printInterfaces

Funkcia si otvorí všetky rozhrania do predpripravenej štruktúry. Štruktúra sa chová ako `linked list` takže názvy rozhraní sa tlačia cez `while` slučku. Po vytlačení sa rozhrania uvoľnia.

```
void printInterfaces(){ //Prints available interfaces
    pcap_if_t *devices;
    if(pcap_findalldevs(&devices, errbuf) != 0){ //if pcap function fails
        fprintf(stderr,"%s\n", errbuf);
        return;
    }
    printf("Interfaces:\n");
    while (devices != NULL){//print device name trough linked list
        printf("%s\n", devices->name);
        devices = devices->next;
    }
    pcap_freealldevs(devices);//free devices after use
}
```

Obrázok 3: Funkcia printInterfaces

3.4 Funkcia filterGen

Funkcia do predpripraveného `stringu` postupne pridáva `substringy` podľa `flagov`, ktoré sa nastavovali pri analýze argumentov programu. Pri TCP a UDP sa kontrolujú aj implementácia konkrétneho portu. Ak sa vypisuje viacej protokolov do `stringu` kontroluje sa aj výpis “or“.

```
string filterGen(){ //Generates filter based on program arguments
    string filter;
    if (f_tcp){ //if tcp is selected
        if (port == -1){ //if no port is set
            filter = filter + "tcp";
        }else{ //if port is set
            filter = filter + "(tcp and port " + to_string(port) + ")";
        } //if more protocols are selected
        if (f_arp || f_udp || f_icmp){
            filter = filter + " or ";
        }
    }
    if (f_udp){ //if udp is selected
        if (port == -1){//if no port is set
            filter = filter + "udp";
        }else{ //if port is set
            filter = filter + "(udp and port " + to_string(port) + ")";
        }
        if (f_arp || f_icmp){ //if more protocols are selected
            filter = filter + " or ";
        }
    }
    if (f_arp){ //if arp is selected
        filter = filter + "arp";
        if (f_icmp){ //if more protocols are selected
            filter = filter + " or ";
        }
    }
    if (f_icmp){ //if icmp is selected
        filter = filter + "icmp or icmp6";
    }
    return filter;
}
```

Obrázok 4: Funkcia filterGen

3.5 Funkcia printPacket

Funkcia najprv zavolá funkciu na spracovanie a výpis `timestampu` z `pcap header`. Potom si uloží hlavičku paketu do štruktúry z knižnice `netinet`. Vytlačí MAC adresy odosielateľa a príjemcu, ktoré sú uložené v štruktúre ethernet hlavičky. MAC adresy sú tlačené postupne po bytoch. Potom sa vytlačí veľkosť ethernet framu v bytoch, ktoré sú uložené v `pcap header`. Následne sa skontroluje typ paketu a podľa toho sa spustí príslušná funkcia do ktorej sa pošle paket bez ethernet hlavičky. Po ďalšom spracovaní sa vytlačí hexadecimálne obsah ethernet framu.

```
void printPacket(const u_char *packet, pcap_pkthdr header){ //Inspired by https://www.tcpdump.org/pcap.html
    printTimeStamp(header);                                 //Parse and prints information from packet
    //Save ethernet header to struct
    struct ethhdr *eth = (struct ethhdr *)packet;
    //Write MAC adresses inspired by https://www.binarytides.com/packet-sniffer-code-in-c-using-linux-sockets-bsd-part-2/
    printf("src MAC: %.2X:%.2X:%.2X:%.2X:%.2X:%.2X\n", eth->h_source[0], eth->h_source[1], eth->h_source[2], eth->h_source[3], eth->h_source[4], eth->h_source[
    printf("dst MAC: %.2X:%.2X:%.2X:%.2X:%.2X:%.2X\n", eth->h_dest[0], eth->h_dest[1], eth->h_dest[2], eth->h_dest[3], eth->h_dest[4], eth->h_dest[5]);
    //Frame lenght saved in ethernet header
    printf("frame length: %i bytes\n", header.caplen);
    //Check protocol in header
    if (ntohs(eth->h_proto) == IPv4_p){
        ipv4_parse((packet + sizeof(struct ethhdr)));
    }else if (ntohs(eth->h_proto) == IPv6_p){
        ipv6_parse((packet + sizeof(struct ethhdr)));
    }else if (ntohs(eth->h_proto) == ARP_p){
        arp_parse((packet + sizeof(struct ethhdr)));
    }
    //Print hex values of packet
    hexDump(&packet, header.caplen);
    printf("\n");
}
```

Obrázok 5: funkcia printPacket

3.6 Funkcia printTimestamp

Funkcia si uloží do predpripravenej štruktúry čas podľa počtu sekúnd z pcap header. Potom si uloží do stringu pomocou funkcie strftime hodnoty od roku po sekundy podľa vzoru. Následne pridá mikrosekundy a časové pásmo podľa údajov z pcap header a hodnoty z štruktúry time ktorá obsahuje počet sekúnd mimo GMT, ktoré vydelí počtom sekúnd v hodine. Finálny string vytlačí.

```
void printTimeStamp(pcap_pkthdr header){ //used and adjusted code from https://stackoverflow.com/questions/2408976/struct-timeval-to-printable-format
    struct tm *time;                     //Converts and prints timestamp from timeval
    char timestampBuff[20], timestamp[33];
    //Set up tm struct for use in later
    time = localtime(&(header.ts.tv_sec));

    //Insert Years, Months, Days, Hours, Minutes and Seconds
    strftime(timestampBuff, sizeof(timestampBuff), "%FT%H:%M:%S", time);
    //Add microseconds and timezone
    snprintf(timestamp, sizeof(timestamp), "%s.%06ld+%.2ld:00", timestampBuff, header.ts.tv_usec, time->tm_gmtoff/3600);
    printf("timestamp: %s\n", timestamp);
}
```

Obrázok 6: Funkcia printTimestamp

3.7 Funkcia ipv4_parse

Funkcia si do štruktúry IPv4 hlavičky uloží hlavičku z paketu. Následne si vypočíta veľkosť hlavičky podľa hodnoty v ihl v hlavičke, ktorú vynásobí 4 aby dostal hodnotu v bytoch. Pomocou funkcie `inet_ntop` vypíše IPv4 adresy v požadovanom formáte. Následne skontroluje protokol vnútri paketu podľa ktorého spustí príslušnú funkciu do ktorej pošle paket bez IPv4 hlavičky.

```
                                //Parse IPv4 packet
void ipv4_parse (const u_char *packet){ //Used and adjusted code from https://www.binarytides.com/packet-sniffer-code-in-c-using-linux-sockets-bsd-part-2/
    //header lenght
    unsigned short iphdrlen;
    //address strings
    char saddr[16], daddr[16];
    //ipv4 header to struct
    struct iphdr *iph = (struct iphdr *) packet ;
    //size of header in double words to bytes
    iphdrlen = iph->ihl*4;
    //Write IP adresses to strings
    inet_ntop(AF_INET, &(iph->saddr), saddr, sizeof(saddr));
    inet_ntop(AF_INET, &(iph->daddr), daddr, sizeof(daddr));

    printf("src IP: %s\n", saddr);
    printf("dst IP: %s\n", daddr);
    //Check protocol
    if (iph->protocol == IPPROTO_TCP){
        tcp_parse(packet + iphdrlen);
    }else if (iph->protocol == IPPROTO_UDP){
        udp_parse(packet + iphdrlen);
    }else if (iph->protocol == IPPROTO_ICMP){
        printf("\n");
    }
}
```

Obrázok 7: Funkcia ipv4_parse

3.8 Funkcia ipv6_parse

Funkcia si do štruktúry IPv6 hlavičky uloží hlavičku z paketu. Pomocou funkcie `inet_ntop` vypíše IPv6 adresy v požadovanom formáte. Následne skontroluje protokol vnútri paketu podľa ktorého spustí príslušnú funkciu do ktorej pošle paket bez IPv6 hlavičky.

```
void ipv6_parse (const u_char *packet){ //Parse IPv6 packet
    //adress strings
    char saddr[40], daddr[40];
    //IPv6 header to struct
    struct ip6_hdr *ip6h = (struct ip6_hdr *) packet;
    //Write IPv6 adresses to strings
    inet_ntop(AF_INET6, &(ip6h->ip6_src), saddr, sizeof(saddr));
    inet_ntop(AF_INET6, &(ip6h->ip6_dst), daddr, sizeof(daddr));

    printf("src IP: %s\n", saddr);
    printf("dst IP: %s\n", daddr);
    //Check protocol
    if (ip6h->ip6_ctlun.ip6_un1.ip6_un1_nxt == IPPROTO_TCP){
        tcp_parse(packet + IPv6_hlen);
    }else if (ip6h->ip6_ctlun.ip6_un1.ip6_un1_nxt == IPPROTO_UDP){
        udp_parse(packet + IPv6_hlen);
    }else if (ip6h->ip6_ctlun.ip6_un1.ip6_un1_nxt == IPPROTO_ICMPV6){
        printf("\n");
    }
}
```

Obrázok 8: Funkcia ipv6_parse

3.9 Funkcia arp_parse

Funkcia si vytiahne z paketu ARP hlavičku do prepripravenej štruktúry. Potom pomocou funkcie `inet_ntop` vypíše IPv4 adresy v správnom formáte.

```
void arp_parse (const u_char *packet){ //Parse arp_packet
    //arp header to struct
    struct ether_arp *arp = (struct ether_arp*) packet;
    char saddr[16], daddr[16];
    //Write IPv4 adresses to string
    inet_ntop(AF_INET, &(arp->arp_spa), saddr, sizeof(saddr));
    inet_ntop(AF_INET, &(arp->arp_tpa), daddr, sizeof(daddr));

    printf("src IP: %s\n", saddr);
    printf("dst IP: %s\n\n", daddr);
}
```

Obrázok 9:Funkcia arp_parse

3.10 Funkcie tcp_parse, udp_parse

Funkcia si vytiahne do štruktúry príslušnú hlavičku a z nej vytlačí porty odosielateľa a príjemcu.

```
void tcp_parse (const u_char *packet){ //Parse tcp
    //TCP header to struct
    struct tcphdr *tcp=(struct tcphdr*)packet;
    //Print source and destination ports
    printf("src port: %i\n", ntohs(tcp->source));
    printf("dst port: %i\n\n", ntohs(tcp->dest));
}
```

Obrázok 11: Funkcia tcp_parse

```
void udp_parse (const u_char *packet){ //Parse udp
    //UDP header to struct
    struct udphdr *udp=(struct udphdr*)packet;
    //print source and destination port
    printf("src port: %i\n", ntohs(udp->source));
    printf("dst port: %i\n\n", ntohs(udp->len));
}
```

Obrázok 10:Funkcia udp_parse

3.11 Funkcia hexDump

Funkcia vytlačí obsah ethernet framu. Funkcia tlačí pokiaľ sa index bytov nachádza vnútri framu. Nazačiatku vytlačí offset bytov v stvormiestnej hexadecimálnej hodnote, potom 16 bytov v dvojmiestnej hexadecimálnej hodnote a potom rovnakých 16 bytov ako písmená z ASCII tabuľky. V polovici hexadecimálnych hodnôt a ASCII znakov je 2 miestna medzera. Vypisuje sa 16 znakov v riadku až po koniec framu. Ak frame skončí skorej ako 16 znak v riadku tak sa dotlačia zvyšné medzery aby každý riadok vyzeral rovnako.

```
                                        // Prints out hex contents of the packet
void hexDump(const u_char *packet, size_t size) { //inspired by https://www.programcreek.com/cpp/?CodeExample=hex+dump
        size_t i = 0; //number value used in loops
        uint32_t bytesNum = 0; // number value used in loops
        while (bytesNum < size) { //while to not read out of frame
            printf("0x%04x:", bytesNum); //print first part of hexdump (0x0000:)
            while (bytesNum + i < size && i < 16){ //while for writing bytes in hex
                printf(" %02x", packet[bytesNum+i]);
                if (i == 7){ //extra space in middle
                    printf(" ");
                }
                i++; //iterate trough i so bytesNum can be used again in next print
            }
            for(;i<16;i++){ //Padding if hex finishes before 16 values in line
                printf("   ");
                if (i == 7){
                    printf(" ");
                }
            }
            printf("  ");
            i = 0; //reset i for next cycle
            while (bytesNum + i < size && i < 16){ //while for writing bytes as char
                if (packet[bytesNum+i] >= 32 && packet[bytesNum+i] <= 0x7e){ //if char is printable
                    printf("%c", packet[bytesNum+i]);
                }else{ //if char is not printable
                    printf(".");
                }
                if (i == 7){ //extra space in middle
                    printf(" ");
                }
                i++;
            }
            printf("\n");
            i = 0; //reset i
            bytesNum += 16; //add 16 to bytesNum
        }
    }
```

Obrázok 12: Funkcia hexDump

4 Testovanie

4.1 Priebeh testovania

Testovanie som robil pomocou programu Wireshark. Porovnával som pakety chytené mojím analyzátorom a Wiresharkom a ak sa niekde údaje nezhodovali, tak som prepisoval kód v analyzátore. IPv6 som netestoval pretože na internáte nemám IPv6 pripojenie.

4.2 TCP IPv4 paket test

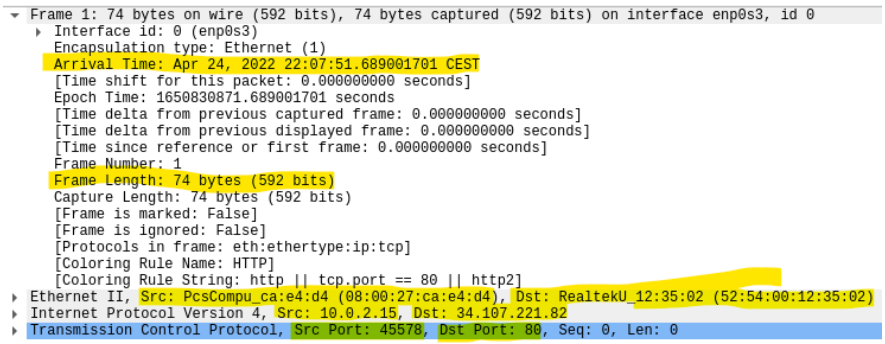
Môj nájdený paket spolu s spúšťacou kombináciou parametrov. Test prebiehal spustením internetového prehliadača:

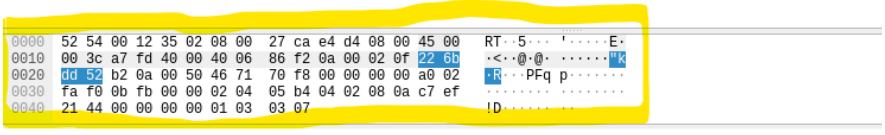
```
student@student-vm:~/Documents/IPKSniffer$ sudo ./ipk-sniffer -i enp0s3 --tcp -p 80
timestamp: 2022-04-24T22:07:51.689001+02:00
src MAC: 08:00:27:CA:E4:D4
dst MAC: 52:54:00:12:35:02
frame length: 74 bytes
src IP: 10.0.2.15
dst IP: 34.107.221.82
src port: 45578
dst port: 80

0x0000: 52 54 00 12 35 02 08 00  27 ca e4 d4 08 00 45 00  RT..5... '.....E.
0x0010: 00 3c a7 fd 40 00 40 06  86 f2 0a 00 02 0f 22 6b  .<..@.@. ......"k
0x0020: dd 52 b2 0a 00 50 46 71  70 f8 00 00 00 00 a0 02  .R...PFq p.......
0x0030: fa f0 0b fb 00 00 02 04  05 b4 04 02 08 0a c7 ef  ........ ........
0x0040: 21 44 00 00 00 00 01 03  03 07                    !D...... ..
```

Obrázok 13:Môj nájdený TCP packet

Paket nájdený programom Wireshark s vyznačenými kontrolnými hodnotami:





Obrázok 14: TCP paket nájdený Wiresharkom

4.3 UDP IPv4 paket test

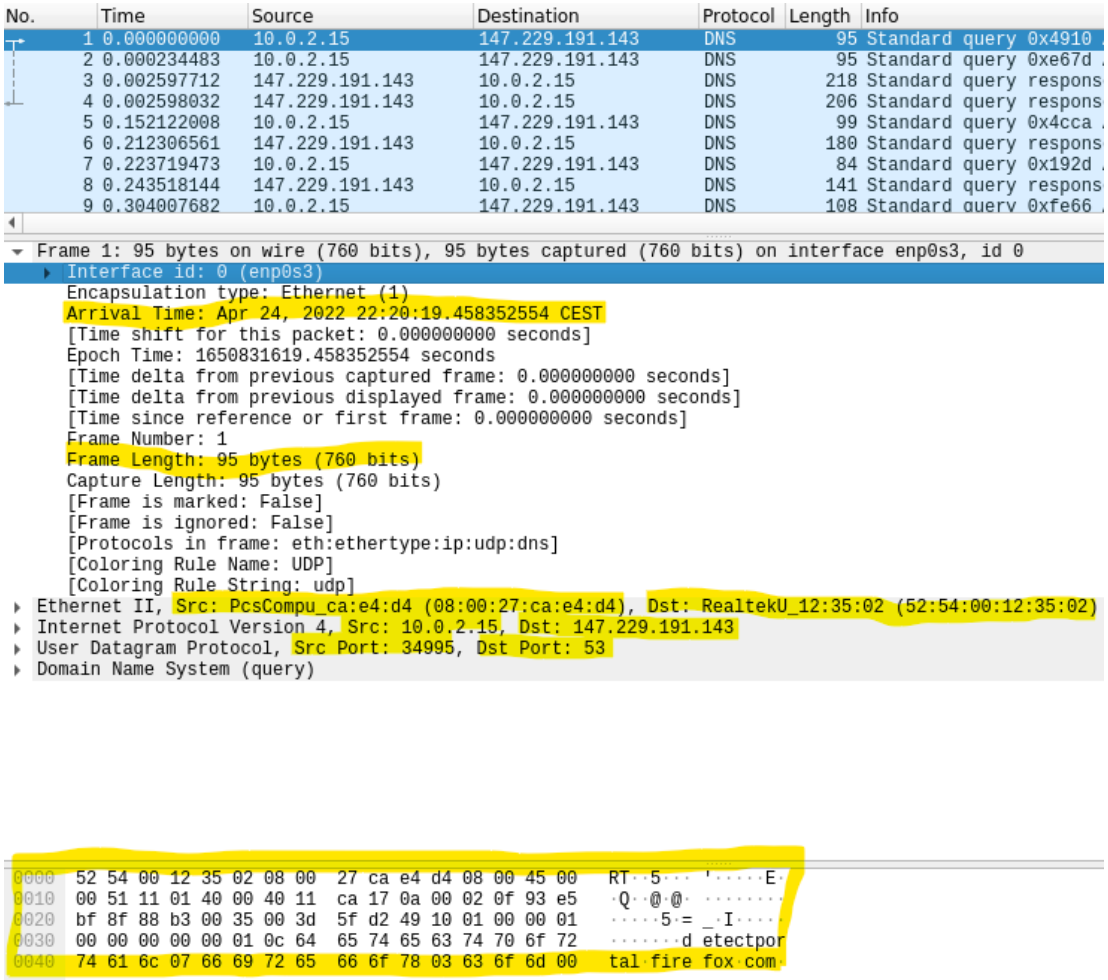
Môj nájdený paket spolu s spúšťacou kombináciou parametrov. Test prebiehal spustením internetového prehliadača:

```
student@student-vm:~/Documents/IPKSniffer$ sudo ./ipk-sniffer -i enp0s3 --udp
timestamp: 2022-04-24T22:20:19.458352+02:00
src MAC: 08:00:27:CA:E4:D4
dst MAC: 52:54:00:12:35:02
frame length: 95 bytes
src IP: 10.0.2.15
dst IP: 147.229.191.143
src port: 34995
dst port: 61

0x0000: 52 54 00 12 35 02 08 00  27 ca e4 d4 08 00 45 00  RT..5... '.....E.
0x0010: 00 51 11 01 40 00 40 11  ca 17 0a 00 02 0f 93 e5  .Q..@.@. ........
0x0020: bf 8f 88 b3 00 35 00 3d  5f d2 49 10 01 00 00 01  .....5.= _.I.....
0x0030: 00 00 00 00 00 01 0c 64  65 74 65 63 74 70 6f 72  .......d etectpor
0x0040: 74 61 6c 07 66 69 72 65  66 6f 78 03 63 6f 6d 00  tal.fire fox.com.
0x0050: 00 01 00 01 00 00 29 02  00 00 00 00 00 00 00     ......). .......
```

Obrázok 15: Môj nájdený UDP paket

Paket nájdený programom Wireshark s vyznačenými kontrolnými hodnotami:



Obrázok 16: UDP paket nájdený Wiresharkom

4.4 ICMP IPv4 paket test

Môj nájdený paket spolu s spúšťacou kombináciou parametrov. Test prebiehal pingom loopback adresy:

```
student@student-vm:~/Documents/IPKSniffer$ sudo ./ipk-sniffer -i lo --icmp
timestamp: 2022-04-24T22:28:14.462056+02:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 98 bytes
src IP: 127.0.0.1
dst IP: 127.0.0.1

0x0000: 00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00  ........ ......E.
0x0010: 00 54 47 15 40 00 40 01  f5 91 7f 00 00 01 7f 00  .TG.@.@. ........
0x0020: 00 01 08 00 22 09 00 01  00 01 de b2 65 62 00 00  ....".... ....eb..
0x0030: 00 00 cc 0c 07 00 00 00  00 00 10 11 12 13 14 15  ........ ........
0x0040: 16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25  ........ .. !"#$%
0x0050: 26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35  &'()*+,- ./012345
0x0060: 36 37                                            67
```

Obrázok 17: Môj nájdený ICMP paket

Paket nájdený programom Wireshark s vyznačenými kontrolnými hodnotami:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id
2	0.000014647	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id
3	1.026615712	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id
4	1.026627795	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id

```
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface lo, id 0
    Interface id: 0 (lo)
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 24, 2022 22:28:14.462056054 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1650832094.462056054 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Internet Control Message Protocol
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 54 47 15 40 00 40 01  f5 91 7f 00 00 01 7f 00   ·TG·@·@· ........
0020  00 01 08 00 22 09 00 01  00 01 de b2 65 62 00 00   ····"··· ····eb··
0030  00 00 cc 0c 07 00 00 00  00 00 10 11 12 13 14 15   ········ ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

Obrázok 18: ICMP paket nájdený Wiresharkom

4.5 ARP paket test

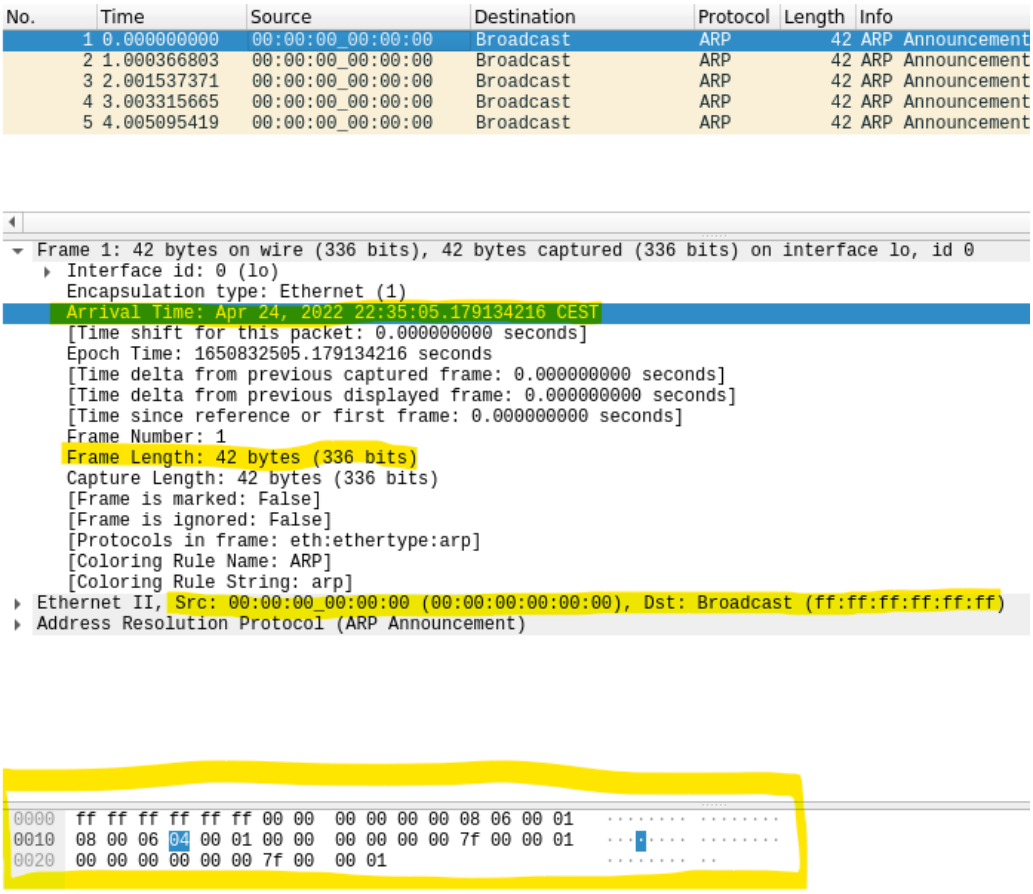
Môj nájdený paket spolu s spúšťacou kombináciou parametrov. Test pomocou programu nping na loopback adresu:

```
student@student-vm:~/Documents/IPKSniffer$ sudo ./ipk-sniffer -i lo --arp
timestamp: 2022-04-24T22:35:05.179134+02:00
src MAC: 00:00:00:00:00:00
dst MAC: FF:FF:FF:FF:FF:FF
frame length: 42 bytes
src IP: 127.0.0.1
dst IP: 127.0.0.1

0x0000: ff ff ff ff ff ff 00 00  00 00 00 00 08 06 00 01  ........ ........
0x0010: 08 00 06 04 00 01 00 00  00 00 00 00 7f 00 00 01  ........ ........
0x0020: 00 00 00 00 00 00 7f 00  00 01                    ........ ..
```

Obrázok 19: Môj nájdený ARP paket

Paket nájdený programom Wireshark s vyznačenými kontrolnými hodnotami:



Obrázok 20: ARP paket nájdený Wiresharkom

5 Použité zdroje pri písaní

Getopt a analýza argumentov programu:
https://azrael.digipen.edu/~mmead/www/Courses/CS180/getopt.html

Otvorenie sieťového rozhrania a práca s ethernet hlavičkou:
https://www.tcpdump.org/pcap.html

Práca s protokolovými hlavičkami:
https://www.binarytides.com/packet-sniffer-code-in-c-using-linux-sockets-bsd-part-2/

Práca s timestampom:
https://stackoverflow.com/questions/2408976/struct-timeval-to-printable-format

HexDump inšpirácia:
https://www.programcreek.com/cpp/?CodeExample=hex+dump