

**VYSOKÉ UCENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMACNÍCH TECHNOLOGIÍ**



Počítačové komunikace a sítě

Projektová dokumentace

Packet sniffer - varianta ZETA

3. dubna 2022

Martin Baláž (xbalaz15)

Obsah

1. Úvod	3
2. Spuštění programu	3
3. Implementace.....	4
3.1. použité knihovny.....	4
3.2. funkce main()	5
3.3. funkce process_packet()	6
3.4. funkce pro vypisování informací o paketu/rámci.....	6
3.5. funkce print_data().....	6
4. Příklady výpisu	7
5. Testování	10
6. Použité zdroje	11

1. Úvod

Výsledkem vyřešeného projektu je síťový analyzátor, napsaný v jazyce C, který je schopný na určitém síťovém rozhraní zachytávat a filtrovat pakety.

Program umí zachytávat a filtrovat pakety na těchto protokolech: TCP, UDP, ICMPv4, ICMPv6 a také ARP. U každého packetu se zjistí typ IP adres příjemce a odesílatele. Program vypíše na STDOUT přibližný čas zachycení packetu, zdrojovou a cílovou MAC adresu, délku packetu v bytech, zdrojovou a cílovou IP adresu (typu IPv4, nebo IPv6), zdrojový a cílový port a obsah packetu v hexadecimálním, ale i ASCII tvaru.

2. Spuštění programu

Program byl vytvořený a testovaný na operační systém Ubuntu 20.04, pro jiné operační systémy funkčnost nezaručuji. Součástí projektu je Makefile, který umožňuje sestavit projekt příkazem: *make*, ale i odstranit vytvořený program (*ipk-sniffer*) pomocí příkazu: *make clean*. Při překladu dochází k vypsání warningu o nepoužívání parametru *args*, ale programu to ničím neškodí. Pro spuštění projektu je potřeba mít administrátorská práva, toho dosáhneme použitím příkazu *sudo*, nebo přepnutím do rootu.

Spouštění programu:

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

- *-i eth0* (právě jedno rozhraní, na kterém se bude poslouchat. Nebude-li tento parametr uveden, či bude-li uvedené jen *-i* bez hodnoty, vypíše se seznam aktivních rozhraní)
- *-p 23* (bude filtrování paketů na daném rozhraní podle portu; nebude-li tento parametr uveden, uvažují se všechny porty; pokud je parametr uveden, může se daný port vyskytnout jak v source, tak v destination části)
- *-t* nebo *--tcp* (bude zobrazovat pouze TCP pakety)
- *-u* nebo *--udp* (bude zobrazovat pouze UDP pakety)
- *--icmp* (bude zobrazovat pouze ICMPv4 a ICMPv6 pakety)
- *--arp* (bude zobrazovat pouze ARP rámce)
- pokud nebudou konkrétní protokoly specifikovány, uvažují se k tisknutí všechny (tj. veškerý obsah, nehledě na protokol)
- *-n 10* (určuje počet paketů, které se mají zobrazit, tj. i "dobu" běhu programu; pokud není uvedeno, uvažujte zobrazení pouze jednoho packetu, tedy jakoby *-n 1*)

U syntaxe vstupních voleb jednotlivým programům složené závorky *{}* znamenají, že volba je nepovinná (pokud není přítomna, tak se použije implicitní hodnota), oproti tomu *[]* znamená povinnou volbu. Přičemž pořadí jednotlivých argumentů a jejich parametrů může být libovolné. Musí se zadat pouze takový port, přes který se můžou dané pakety posílat.

V případě zadání neplatného/neexistujícího argumentu je vypsána nápověda, jak program používat a v případě zadání nadbytečného parametru (tj. nepatří k žádnému argumentu) budete vyzváni k jejich odstranění. Pokud budou zadána dva a více stejných argumentů, dojde k chybě a budete na to upozorněni.

3. Implementace

Pro implementaci jsem použil knihovnu *Libpcap*. Jedná se o open source knihovnu jazyka C, která poskytuje API pro zachycení paketů z datových linek. Samotný kód napsaný v jazyce C je uložen v jednom souboru *ipk-sniffer.c*.

```
// Global variables
int frame_length = 0;

char src_ipv4[INET_ADDRSTRLEN]; // IPv4 source address
char dest_ipv4[INET_ADDRSTRLEN]; // IPv4 destination address

char src_ipv6[INET_ADDRSTRLEN]; // IPv6 source address
char dest_ipv6[INET_ADDRSTRLEN]; // IPv6 destination address
```

Obrázek 1 - Globální proměnné

Dále jsou popsány knihovny, které jsem použil a některé funkce, které jsem naprogramoval.

```
// Functions declaration
void process_packet(u_char *, const struct pcap_pkthdr *, const u_char *);
void print_tcp_packet(const u_char * Buffer, int Size, bool Ipv6);
void print_udp_packet(const u_char * Buffer, int Size, bool Ipv6);
void print_icmp_packet(const u_char * Buffer, bool Ipv6);
void print_arp_frame(const u_char * Buffer, bool Ipv6);
void print_ethernet_header(const u_char * Buffer);
void print_data(const u_char * Buffer, int Size);
void print_ip(char * source, char * dest);
void print_timestamp();
```

Obrázek 2 – Deklarace všech funkcí

3.1. použité knihovny

Pro základní práci jsem používal funkce z knihoven *<stdio.h>*, *<stdbool.h>*, *<stdlib.h>* a *<string.h>* pro práci se stringy. Pro zpracování argumentů a parametrů příkazové řádky jsem použil *<getopt.h>*, *<ctype.h>* kvůli *isprint()* funkci a *<time.h>* pro práci s aktuálním časem. *<pcap/pcap.h>* pro zpřístupnění Libpcap knihovny pro práci s pakety. *<arpa/inet.h>* kvůli funkci *inet_ntop()* pro získání IPv4 a IPv6 adres a *<net/ethernet.h>* kvůli důležitým ethernetovým konstantám. A v poslední řadě jsem použil funkce pro přístup k hlavičkám jednotlivých protokolů a ethernetové hlavičky a to jsou: *<netinet/if_ether.h>*, *<netinet/udp.h>*, *<netinet/tcp.h>*, *<netinet/ip.h>* a *<netinet/ip6.h>*

3.2. funkce main()

Nadefinoval jsem si, které argumenty mají dlouhou a krátkou verzi, či pouze krátkou verzi, včetně volitelných parametrů, viz. obrázek 3.

```
char *short_options = "i::p::tun::";
```

Obrázek 3 - Šablona pro argumenty

Pro zpracování argumentů a parametrů jsem použil funkci `getopt_long()`. Před začátkem zpracovávání jsem si však nadefinoval vlajky argumentů na false, abych při zpracovávání odhalil, zda se jednotlivý argument objevil víckrát a vyhodil chybu. Protože `getopt` funkce nepovoluje argumentům s volitelnými parametry psát parametry za mezeru, využil jsem následující kus kódu, abych to umožnil.

```
// source: https://cfengine.com/blog/2021/optional-arguments-with-getopt-long/  
// author: Lars Erik Wik  
// date: 13th August, 2021  
if (optarg == NULL && optind < argc && argv[optind][0] != '-') optarg = argv[optind++];
```

Obrázek 4 - Psaní parametrů za mezeru po argumentu

V případě, že zadáme neexistující argument, bude vypsána nápověda na standartní výstup. Pokud by se stalo, že by někdo zadal navíc parametr, který k žádnému argumentu nepatří, je tato možnost ošetřena a bude vyzván k jejich odstranění. Pokud uživatel nezadá žádné rozhraní, bude vypsán seznam aktivních rozhraní a to díky ukazateli na strukturu `pcap_if_t`, do které uložíme seznam funkcí `pcap_findalldevs()`, a pomocí cyklu se vypíše.

Pokud bude zadané nějaké rozhraní, prvně získáme masku sítě našeho připojení pomocí funkce `pcap_lookupnet()` a otevřu rozhraní pro síťové přenosy za pomocí funkce `pcap_open_live()` v promiskuitním režimu, která vrátí manipulátor. Dále je sestaven filtr pomocí vlajek jednotlivých protokolů a výskytu portu, viz. obrázek 5.

```
if (icmp_flag)  
{  
    strcat(filter, "icmp or icmp6");
```

Obrázek 5 - V případě true, bude zkopírováno na konec filtru icmp or icmp6

Pomocí funkce `pcap_compile()` je nyní možné zkompilovat filtrový výraz a aplikovat jej do manipulátoru funkcí `pcap_setfilter()`. Poslední věc, kterou funkce `main` udělá je, že zavolá

```
// Compile the filter expression  
// source: https://www.tcpdump.org/manpages/pcap\_compile.3pcap.html  
if(pcap_compile(handle, &fp, filter, 0, pNet))  
{  
    fprintf(stderr, "\npcap_compile() failed\n");  
    printf("pcap_compile(): %s\n", pcap_geterr(handle));  
    return 1;  
}  
// Apply the compiled filter  
if(pcap_setfilter(handle, &fp) == -1)  
{  
    fprintf(stderr, "pcap_setfilter() failed\n");  
    return 1;  
}  
  
// Put the device in sniff loop  
pcap_loop(handle, num, process_packet, NULL);
```

Obrázek 6 - Ukázka pcap funkcí

funkci `pcap_loop()`, které slouží k samotnému „sniffování“ paketů ve smyčce, dokud nenarazí na zadanou hodnotu parametru `num`, která udává počet získaných paketů. Za pomoci funkce `process_packet()`, která je volána ve smyčce, se jednotlivý paket vyhodnotí.

3.3. funkce `process_packet()`

Prvně se zavolá funkce `print_timestamp()` pro získání co nejvíc přesného času chytnutí paketu. Do proměnné `frame_length` uložím délku paketu v bytech. Dále určím zda se paket přijímá a odesílá v IPv4 nebo IPv6 adrese tak, že porovná ethernetový typ paketu s konstantou `ETHERTYPE_IPV6` z knihovny `<net/ethernet.h>`. Kvůli budoucímu vypisování portu je potřeba ukládat do proměnné `size` velikost hlavičky ethernetu a IPv4 nebo IPv6 hlavičky. Už v této funkci získám IP adresy pomocí funkce `inet_ntop()` a uložím je do globálních proměnných. Nakonec se pomocí switchu vyhodnotí protokol zachyceného paketu a zavolá se k němu určená funkce na vypsání informací o paketu/rámci.

3.4. funkce pro vypisování informací o paketu/rámci

Na výpis informací mám 4 funkce, pro každý protokol jednu (kromě ICMP), které všechny fungují na stejný princip, viz. obrázek 7.

```
// Print icmpv4 or icmpv6 packet
void print_icmp_packet(const u_char * Buffer, bool Ipv6)
{
    print_ethernet_header(Buffer);
    printf("frame length: %d bytes\n", header_length);
    if(Ipv6) print_ip(src_ipv6, dest_ipv6);      // ICMPv6
    else print_ip(src_ipv4, dest_ipv4);          // ICMPv4
    printf("src port:\n");
    printf("dst port:\n");
    print_data(Buffer, header_length);
}
```

Obrázek 7 - Funkce pro výpis informací o protokolu ICMPv6 nebo ICMPv4

ICMP a ARP funkce však neobsahují informaci o portu, protože žádný nemají. Funkce vypisují MAC adresy, délku paketu v bytech, cílovou a zdrojovou IP adresu, cílový a zdrojový port (pokud je) a nakonec se zavolá funkce `print_data()`

3.5. funkce `print_data()`

Tato funkce vypisuje data z paketu po jednotlivých bytech, a to v hexadecimálním tvaru i v ASCII. Funkce je složena z jednoho velkého `for` cyklu a pár vnořených. Funkce vypisuje

po řádcích, začíná vždy offsetem, poté 16 bytů v hexadecimálním tvaru oddělené mezerami a nakonec 16 bytů v ASCII oddělené mezerou po 8 bytech.

```
// Print the number of bytes printed at very start of each line
if (line_counter < 10) printf("0x00%d:", line_counter++ * 10);
else if (line_counter < 100) printf("0x0%d:", line_counter++ * 10);
else printf("0x%d:", line_counter++ * 10);
```

Obrázek 8 - Vypisování offsetu

4. Příklady výpisu

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ ./ipk-sniffer

Available Interfaces are :
1. eno1 - (No description)
2. lo - (No description)
3. any - (Pseudo-device that captures on all interfaces)
4. wlp4s0 - (No description)
5. bluetooth-monitor - (Bluetooth Linux Monitor)
6. nflog - (Linux netfilter log (NFLOG) interface)
7. nfqueue - (Linux netfilter queue (NFQUEUE) interface)
8. bluetooth0 - (Bluetooth adapter number 0)
```

Obrázek 9 - Vypsání aktivních rozhraní

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1

timestamp: 2022-04-01T09:48:04.195357+02:00
src MAC: 94:3f:c2:07:ca:04
dst MAC: 38:f3:ab:a5:21:f6
frame length: 114 bytes
src IP: e:116c
dst IP: 2001:67c:1220:c1a2:81ad:9e11:dc2e:116c
src port: 443
dst port: 53136

0x0000:  38 f3 ab a5 21 f6 94 3f c2 07 ca 04 86 dd 60 00  8...!...? .....`.
0x0010:  00 00 00 3c 06 38 2a 03 28 80 f0 3d 00 12 fa ce  ...<.8*. (..=....
0x0020:  b0 0c 00 00 00 02 20 01 06 7c 12 20 c1 a2 81 ad  ..... . .|. ....
0x0030:  9e 11 dc 2e 11 6c 01 bb cf 90 73 2f 5b 84 d3 b7  ....l.. ..s/[...
0x0040:  4b 0a 80 18 01 37 86 69 00 00 01 01 08 0a 26 f6  K....7.i .....&.
0x0050:  ec b5 da 3e 69 f2 17 03 03 00 17 61 2f fd fd 04  ...>i... ..a/...
0x0060:  f7 5f 79 e0 9e fc 94 8a 3b be fc c0 92 ce 6c b1  ._y..... ;.....l.
0x0070:  a7 e2                                           ..
```

Obrázek 10 - Výpis informací paketu libovolného protokolu

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1 --udp -p 2007

timestamp: 2022-04-01T09:54:31.235451+02:00
src MAC: b0:0c:d1:a1:df:77
dst MAC: ff:ff:ff:ff:ff:ff
frame length: 62 bytes
src IP: 147.229.216.84
dst IP: 147.229.219.255
src port: 2007
dst port: 2007

0x0000:  ff ff ff ff ff ff b0 0c d1 a1 df 77 08 00 45 00  ....w..E.
0x0010:  00 30 8f 3a 00 00 80 11 cf 63 93 e5 d8 54 93 e5  .0.:... .C...T..
0x0020:  db ff 07 d7 07 d7 00 1c 89 40 42 43 20 31 35 4c  .....@BC 15L
0x0030:  41 50 54 4f 50 2d 48 56 4c 43 42 33 36 4e      APTOP-HV LCB36N
```

Obrázek 11 - Vypsání informací o paketu na rozhraní eno1 protokolu UDP na portu 2007

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1 --icmp

timestamp: 2022-04-01T09:58:06.947515+02:00
src MAC: 50:eb:f6:29:55:c8
dst MAC: 33:33:ff:09:4c:e6
frame length: 86 bytes
src IP: fe80::c480:163a:bd32:ac66
dst IP: ff02::1:ff09:4ce6
src port:
dst port:

0x0000:  33 33 ff 09 4c e6 50 eb f6 29 55 c8 86 dd 60 00  33..L.P. .)U...`.
0x0010:  00 00 00 20 3a ff fe 80 00 00 00 00 00 00 c4 80  ... :... .....
0x0020:  16 3a bd 32 ac 66 ff 02 00 00 00 00 00 00 00 00  ...2.f.. .....
0x0030:  00 01 ff 09 4c e6 87 00 54 66 00 00 00 00 fe 80  ....L... Tf.....
0x0040:  00 00 00 00 00 00 e1 76 82 af 49 09 4c e6 01 01  ....v ..I.L...
0x0050:  50 eb f6 29 55 c8      P..)U.
```

Obrázek 12 - Vypsání informací o paketu protokolu ICMPv6

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1 --icmp

timestamp: 2022-04-01T09:59:00.547530+02:00
src MAC: 00:50:56:ad:78:ee
dst MAC: ff:ff:ff:ff:ff:ff
frame length: 60 bytes
src IP: 147.229.216.2
dst IP: 147.229.216.0
src port:
dst port:

0x0000:  ff ff ff ff ff ff 00 50 56 ad 78 ee 08 00 45 00  ....P V.x...E.
0x0010:  00 1c e4 43 00 00 26 01 d8 cf 93 e5 d8 02 93 e5  ...C..&. ....
0x0020:  d8 00 08 00 b5 5e 42 a1 00 00 00 00 00 00 00 00  ....^B. ....
0x0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....
```

Obrázek 13 - Vypsání informací o paketu protokolu ICMPv4


```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1 --tcp --udp

timestamp: 2022-04-01T10:01:11.875371+02:00
src MAC: 38:f3:ab:a5:21:f6
dst MAC: 94:3f:c2:07:ca:04
frame length: 129 bytes
src IP: 2001:67c:1220:c1a2:81ad:9e11:dc2e:116c
dst IP: 2a02:598:a::78:196
src port: 59164
dst port: 443

0x0000:  94 3f c2 07 ca 04 38 f3 ab a5 21 f6 86 dd 60 0e  .?....8.  ..!...`.
0x0010:  48 96 00 4b 06 40 20 01 06 7c 12 20 c1 a2 81 ad  H..K.@ .  .|. ....
0x0020:  9e 11 dc 2e 11 6c 2a 02 05 98 00 0a 00 00 00 00  ....l*.  ....
0x0030:  00 00 00 78 01 96 e7 1c 01 bb f2 9b b4 e1 ae bc  ...x....  ....
0x0040:  5c 12 80 18 01 f5 39 9d 00 00 01 01 08 0a 0e 11  \.....9.  ....
0x0050:  be 5a aa a9 97 ca 17 03 03 00 26 45 92 48 4f b0  .Z.....  ..&E.HO.
0x0060:  f3 60 72 79 1b 98 d0 d1 dd 1b 83 e5 0b 9c b7 3e  .`ry....  .....>
0x0070:  28 11 73 2a 09 68 b4 91 d1 77 5e 7e 18 48 4d 17  (.s*.h..  .w^~.HM.
0x0080:  55                                           U
```

Obrázek 14 - Vypsání informací o paketu protokolu TCP nebo UDP (který zachytí dřív)

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1 --arp -n 2

timestamp: 2022-04-01T10:03:19.107517+02:00
src MAC: 94:3f:c2:07:ca:04
dst MAC: ff:ff:ff:ff:ff:ff
frame length: 60 bytes
src IP: 202.4.147.229
dst IP: 216.1.0.0
src port:
dst port:

0x0000:  ff ff ff ff ff ff 94 3f c2 07 ca 04 08 06 00 01  ....? .....
0x0010:  08 00 06 04 00 01 94 3f c2 07 ca 04 93 e5 d8 01  ....? .....
0x0020:  00 00 00 00 00 00 93 e5 d8 5b 00 00 00 00 00 00  ....[.....
0x0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....  ....

timestamp: 2022-04-01T10:03:19.107633+02:00
src MAC: 94:3f:c2:07:ca:04
dst MAC: ff:ff:ff:ff:ff:ff
frame length: 60 bytes
src IP: 202.4.147.229
dst IP: 216.1.0.0
src port:
dst port:

0x0000:  ff ff ff ff ff ff 94 3f c2 07 ca 04 08 06 00 01  ....? .....
0x0010:  08 00 06 04 00 01 94 3f c2 07 ca 04 93 e5 d8 01  ....? .....
0x0020:  00 00 00 00 00 00 93 e5 d9 c3 00 00 00 00 00 00  ....  ....
0x0030:  00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....  ....
```

Obrázek 15 - Vypsání informací o dvou rámcích protokolu ARP

5. Testování

Testování u mě probíhalo ve stylu souběžného běhu Wiresharku a spouštění programu na mém vlastním systému Ubuntu 20.04. Orientoval jsem se podle času chycení paketu (čas v programu byl pozadu před Wireshark časem přibližně o 0,4s) a také jsem používal filtrování ve Wiresharku, do kterého jsem napsal typ protokolu a číslo portu ať už cílového, nebo zdrojového, abych našel ten stejný paket rychleji. Dalším urychlením hledání byla hodnota Length ve Wiresharku a frame length ve výpisu programu. V poslední řadě jsem porovnal IP a MAC adresy a výpis paketu v ASCII/hexadecimálním tvaru.

The image shows a Wireshark capture of a TCP RST packet and a terminal window running a custom sniffing tool.

Wireshark Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
1983	32.496508724	2a03:2880:f03d:d:fa...	2001:67c:1220:c1a2:...	TLSv1.2	111	Application Data
1984	32.496544552	2001:67c:1220:c1a2:...	2a03:2880:f03d:d:fa...	TCP	86	55428 → 443 [ACK] Seq=4980 Ack
2183	35.787432774	2001:67c:1220:c1a2:...	2001:67c:1220:809:...	TCP	86	35446 → 443 [RST, ACK] Seq=1 A

Frame 2183 Details:

- 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eno1, id 0
- Interface id: 0 (eno1)
- Encapsulation type: Ethernet (1)
- Arrival Time: Apr 1, 2022 10:13:10.858712595 CEST
- [Time shift for this packet: 0.000000000 seconds]
- Epoch Time: 1648800790.858712595 seconds
- [Time delta from previous captured frame: 0.002784288 seconds]
- [Time delta from previous displayed frame: 3.290888222 seconds]
- [Time since reference or first frame: 35.787432774 seconds]
- Frame Number: 2183
- Frame Length: 86 bytes (688 bits)
- Capture Length: 86 bytes (688 bits)
- [Frame is marked: False]
- [Frame is ignored: False]
- [Protocols in frame: eth:ethertype:ipv6:tcp]
- [Coloring Rule Name: TCP RST]
- [Coloring Rule String: tcp.flags.reset eq 1]
- Ethernet II, Src: 38:f3:ab:a5:21:f6 (38:f3:ab:a5:21:f6), Dst: HewlettP_07:ca:04 (94:3f:c2:07:ca:04)
- Internet Protocol Version 6, Src: 2001:67c:1220:c1a2:81ad:9e11:dc2e:116c, Dst: 2001:67c:1220:809::93e5:9e2
- Transmission Control Protocol, Src Port: 35446, Dst Port: 443, Seq: 1, Ack: 1, Len: 0

Packet Bytes:

Offset	Hex	ASCII
0000	94 3f c2 07 ca 04 38 f3 ab a5 21 f6 86 dd 60 0f	.?....8. ..!...`.
0010	ea 95 00 20 06 40 20 01 06 7c 12 20 c1 a2 81 ad	...@.
0020	9e 11 dc 2e 11 6c 20 01 06 7c 12 20 08 09 00 00l.
0030	00 00 93 e5 09 e2 8a 76 01 bb 66 56 ed b2 e7 04v ..fV....
0040	16 cb 80 14 01 f5 e6 2d 00 00 01 01 08 0a b1 35-5
0050	1a 25 7f 4c 35 33	...%.L53

Terminal Output:

```
(base) balalek@Lenovo-MB:~/Plocha/IPK/proj2$ sudo ./ipk-sniffer -i eno1 --tcp -p 443
timestamp: 2022-04-01T10:13:11.267303+02:00
src MAC: 38:f3:ab:a5:21:f6
dst MAC: 94:3f:c2:07:ca:04
frame length: 86 bytes
src IP: 2001:67c:1220:c1a2:81ad:9e11:dc2e:116c
dst IP: 2001:67c:1220:809::93e5:9e2
src port: 35446
dst port: 443

0x0000: 94 3f c2 07 ca 04 38 f3 ab a5 21 f6 86 dd 60 0f .?....8. ..!...`.
0x0010: ea 95 00 20 06 40 20 01 06 7c 12 20 c1 a2 81 ad ...@. .|. ....
0x0020: 9e 11 dc 2e 11 6c 20 01 06 7c 12 20 08 09 00 00 ....l. .|. ....
0x0030: 00 00 93 e5 09 e2 8a 76 01 bb 66 56 ed b2 e7 04 .....v ..fV....
0x0040: 16 cb 80 14 01 f5 e6 2d 00 00 01 01 08 0a b1 35 .....- .....5
0x0050: 1a 25 7f 4c 35 33 ...%.L53
```

Obrázek 16 - Testování

6. Použité zdroje

1. WIK, L.E.: Optional arguments with getopt_long(3) [online], rev. 13. srpna 2021, [cit. 2022-03-25]. Dostupné z: <https://cfengine.com/blog/2021/optional-arguments-with-getopt-long/>
2. The GNU C Library: Example of Parsing Long Options with getopt_long [online], [cit. 2022-03-25]. Dostupné z: https://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Option-Example.html
3. SILVER MOON: How to code a Packet Sniffer in C with Libpcap on Linux [online], rev. 31. července 2020, [cit. 2022-03-27]. Dostupné z: <https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/>
4. TCPDUMP/LIBPCAP public repository. TCPDUMP/LIBPCAP public repository [online], [cit. 2022-03-27]. Dostupné z: <https://www.tcpdump.org/manpages/pcap.3pcap.html>
5. TANWAR P.: Capturing packets in your C program with Libpcap [online], rev. 1. února 2011, [vid. 2022-03-27]. Dostupné z: <https://www.opensourceforu.com/2011/02/capturing-packets-c-program-libpcap/>
6. Iana: Protocol numbers [online], rev. 26. února 2021, [vid. 2022-03-28]. Dostupné z: <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>
7. header - I'm trying to build an RFC3339 timestamp in C. How do I get the timezone offset? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [cit. 2022-03-30]. Dostupné z: <https://stackoverflow.com/questions/48771851/im-trying-to-build-an-rfc3339-timestamp-in-c-how-do-i-get-the-timezone-offset>
8. FALLAHI, F.: Isniffer.c. [online], rev. 6. dubna 2020, [cit. 2022-03-30]. Dostupné z: <https://gist.github.com/fffaraz/7f9971463558e9ea9545>
9. Wiki.wireshark: DisplayFilters [online], [vid. 2022-04-01]. Dostupné z: <https://wiki.wireshark.org/DisplayFilters>