VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IPK – Implementačná dokumentácia

Sniffer Paketov

24. apríla 2022 Boris Veselý

Obsah

1 Úvod

Úlohou projektu bolo vytvoriť komunikujúcu aplikáciu podla konkrétnej vybranej špecifikácie za použitia knihovni `libpcap` a/alebo sieťovej knihovni `BSD sockets`. Projekt mal byt vypracovaný v jazyku `C/C++/C#`. Ja som si vybral, že projekt budem robiť v jazyku `C++` a použijem knižnicu `libpcap`.

2 Implementácia

2.1 Návrh aplikácie

Program je ulozený v jednom súbore `main.c` Program požaduje výber rozhrania na ktorom sa packety majú odchytávať, ak je program spustený bez vybraného rozhrania tak pomocou funkcie vypíše na `STDOUT` všetky dostupné rozhrania. Následne keď je zadané rozhranie programu môžeme špecifikovať ktoré protokoly má odchytávať a aj port, ako posledné môžeme zadať koľko packetov má odchytiť. Ak nezadáme žiadne protokoly, porty alebo počet packetov tak program odchytá všetky protokoly (tcp, udp, arp, icmp), porty a odchytí jeden packet a ukončí sa. Ako referenčná aplikácia bola použitá aplikácia WireShark.

2.2 Funkcia `main()`

Po spustení programu sa volá automaticky funkcia `main()` ktorá nastaví aby po ukončení pomocou ctrl+c program zatvoril otvorený `pcap handle` následne sa volá funkcia na parsovanie argumentov a podla argumentov otvára príslušné rozhranie, aplikuje filter ak bol nejaký zadaný a voláme funkciu na parsovanie aktuálneho packetu.

2.3 Parsovanie argumentov

Na parsovanie argumentov bola použitá knižnica `getopt.h`. Pomocou ktorej som nastavil známe argumenty. Táto knižnica mi umožnila nastavenie nepovinných argumentov a tak isto zaistila, že argumenty môžu byť v rôznom poradí. Následne ak je zadaný nejaký argument tak pomocou switchu určím ktorý to bol a zapíšem údaje do mojej štruktúry `flags`, kde sú následne uložené všetky zadané argumenty a je používaná ďalej v programe.

```
typedef struct flags
{
    char *inter;
    int port;
    bool tcp;
    bool udp;
    bool arp;
    bool icmp;
    bool notDef;
    int num;
} flags;
```

2.4 Sniffing

Program po parsovaní argumentov prechádza na „Sniffing packetov“. Ako prvé sa program pokúsi otvoriť zadané rozhranie pomocou funkcií `pcap_lookupnet(), pcap_open_live()` z knižnice `libpcap`. Následne nasleduje funkcia `create_filter()` ktorá vytvorí filter na základe zadaných argumentov. Nasledujú funkcie `pcap_compile(), pcap_setfilter()` tak isto z knižnice `libpcap` ktoré skompilujú vytvorený filter a nastavia ho na otvorené rozhranie. Následne program prechádza na čakanie na paket z otvoreného rozhrania pomocou funkcie `pcap_next()`. Program získaný paket následne parsuje vo funkcii `process_packet()`

2.5 Spracovanie paketu

Program z paketu vyberie „ethernet header“ pomocou štruktúry `ether_header` vďaka čomu získam MAC adresu odosielateľa a prijímateľa, ďalej viem zistiť či paket je IPv4, IPv6 alebo ARP. Podľa toho použijem buď štruktúru `iphdr` pre IPv4, `ip6_hdr` pre IPv6 alebo `ether_arp` pre ARP paket. Keď už mám dáta v týchto štruktúrach, môžem z nich vytiahnuť IP adresy odosielateľa a prijímateľa a protokol. Pre protokoly TCP a UDP musíme použiť štruktúry `udphdr` a `tcphdr` vďaka ktorým môžem vypísať port odosielateľa a prijímateľa.

2.6 Hex dump

Na vypísanie celého paketu je použitá funkcia `hexDump()` ktorá vypíše paket vo formáte kde jednotlivé riadky sú očíslované nasleduje 16 znakov v hexadecimálnom formáte a 16 znakov v ASCII znakoch, netlačiteľné znaky sú reprezentované bodkou.

3 Testovanie

Testovanie bolo použité na overenie správnosti vypísaných dát. Pre referenciu dát bola použitá aplikácia WireShark.

3.1 UDP test

```
timestamp: 2022-04-24T22-07-03.942+02:00
src MAC: 00:15:5d:b2:d2:28
dst MAC: ff:ff:ff:ff:ff:ff
frame length: 305 bytes
src IP: 172.20.32.1
des IP: 172.20.47.255
src port: 54915
dst port: 54915

0x0000  ff ff ff ff ff ff 00 15  5d b2 d2 28 08 00 45 00  ........]..(..E.
0x0010  01 23 68 d8 00 00 80 11  28 c9 ac 14 20 01 ac 14  .#h.....(... ...
0x0020  2f ff d6 83 d6 83 01 0f  20 52 00 42 6f 72 69 73  /....... R.Boris
0x0030  2d 4c 61 70 74 6f 70 00  00 00 00 00 00 00 00 00  -Laptop.........
0x0040  00 00 80 07 ad 66 7d 02  00 00 00 bc 7f d2 f7 00  .....f}.........
0x0050  00 00 00 00 f2 60 7d 02  00 00 33 27 00 00 00 00  .....`}...3'....
0x0060  00 00 70 07 ad 66 7d 02  00 00 40 db 0b 61 7d 02  ..p..f}...@..a}.
0x0070  00 00 30 4c c6 63 7d 02  00 00 c0 bf 7f d2 f7 00  ..0L.c}.........
0x0080  00 00 c6 74 c9 dc fa 7f  00 00 07 01 00 00 00 00  ...t............
0x0090  00 00 e0 bd 7f d2 f7 00  00 00 40 e4 c2 66 7d 02  ..........@..f}.
0x00a0  00 00 60 97 6e 63 7d 02  00 00 88 87 87 7b 39 66  ..`.nc}......{9f
0x00b0  30 35 61 64 32 33 2d 61  35 36 31 2d 34 38 61 66  05ad23-a561-48af
0x00c0  2d 39 32 37 63 2d 33 38  35 62 38 63 30 64 30 34  -927c-385b8c0d04
0x00d0  34 30 7d 00 0a 61 7d 02  00 00 e0 bd 7f d2 1c 00  40}..a}.........
0x00e0  00 00 00 00 00 00 00 00  00 00 01 00 00 00 00 00  ................
0x00f0  00 00 80 bc 7f d2 f7 00  00 00 00 00 00 00 00 00  ................
0x0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0x0110  00 00 00 00 00 00 7d 02  00 00 e4 2d 7f 2d eb b3  ......}....-.-..
0x0120  00 00 8a 51 65 dc fa 7f  00 00 07 01 00 99 a1 ee  ...Qe...........
boris@Boris-Laptop~/D/IPK-sniffer $
```

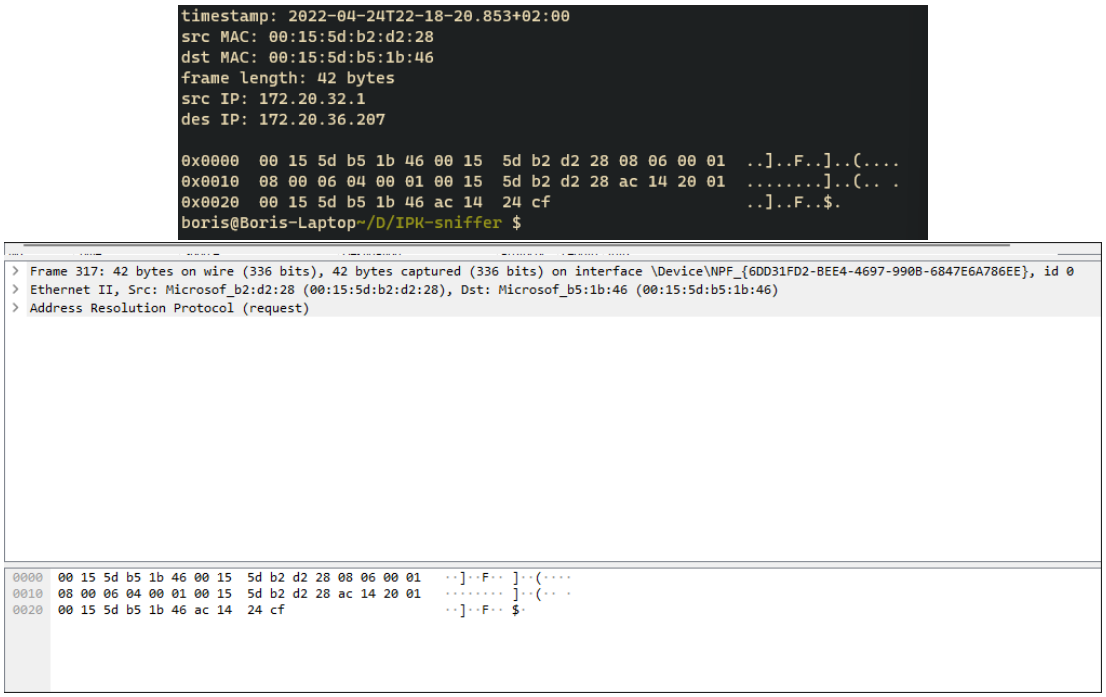
```
> Frame 29: 305 bytes on wire (2440 bits), 305 bytes captured (2440 bits) on interface \Device\NPF_{6DD31FD2-BEE4-4697-990B-6847E6A786EE}, id 0
> Ethernet II, Src: Microsof_b2:d2:28 (00:15:5d:b2:d2:28), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 172.20.32.1, Dst: 172.20.47.255
> User Datagram Protocol, Src Port: 54915, Dst Port: 54915
> Data (263 bytes)
```

```
0000  ff ff ff ff ff ff 00 15  5d b2 d2 28 08 00 45 00   ........ ]··(··E·
0010  01 23 68 d8 00 00 80 11  00 00 ac 14 20 01 ac 14   ·#h····· ···· ···
0020  2f ff d6 83 d6 83 01 0f  20 52 00 42 6f 72 69 73   /·······  R·Boris
0030  2d 4c 61 70 74 6f 70 00  00 00 00 00 00 00 00 00   -Laptop· ········
0040  00 00 80 07 ad 66 7d 02  00 00 00 bc 7f d2 f7 00   ·····f}· ········
0050  00 00 00 00 f2 60 7d 02  00 00 33 27 00 00 00 00   ·····`}· ··3'····
0060  00 00 70 07 ad 66 7d 02  00 00 40 db 0b 61 7d 02   ··p··f}· ··@··a}·
0070  00 00 30 4c c6 63 7d 02  00 00 c0 bf 7f d2 f7 00   ··0L·c}· ········
0080  00 00 c6 74 c9 dc fa 7f  00 00 07 01 00 00 00 00   ···t···· ········
0090  00 00 e0 bd 7f d2 f7 00  00 00 40 e4 c2 66 7d 02   ········ ··@··f}·
00a0  00 00 60 97 6e 63 7d 02  00 00 88 87 87 7b 39 66   ··`·nc}· ·····{9f
00b0  30 35 61 64 32 33 2d 61  35 36 31 2d 34 38 61 66   05ad23-a 561-48af
00c0  2d 39 32 37 63 2d 33 38  35 62 38 63 30 64 30 34   -927c-38 5b8c0d04
00d0  34 30 7d 00 0a 61 7d 02  00 00 e0 bd 7f d2 1c 00   40}··a}· ········
00e0  00 00 00 00 00 00 00 00  00 00 01 00 00 00 00 00   ········ ········
00f0  00 00 80 bc 7f d2 f7 00  00 00 00 00 00 00 00 00   ········ ········
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0110  00 00 00 00 00 00 7d 02  00 00 e4 2d 7f 2d eb b3   ·····}· ···-·-··
0120  00 00 8a 51 65 dc fa 7f  00 00 07 01 00 99 a1 ee   ···Qe··· ········
0130  aa                                                 ·
```

wireshark_vEthernet (WSL)NAL3K1.pcapng | Packets: 31 · Displayed: 31 (100.0%) · Dropped: 0 (0.0%) | Profile: Default

Obr. 1: Test UDP

3.2 ARP test



Obr. 2: Test ARP

4 Literatúra

[1] Carstens, T.: PROGRAMMING WITH PCAP. [online], rev. 2010, [vid. 2022-04-24].
URL https://www.tcpdump.org/pcap.html

[2] Moon, S.: How to code a Packet Sniffer in C with Libpcap on Linux. [online], rev. 2020-07-31, [vid. 2022-04-24].
URL https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/

[3] mq1n: NoMercy. [online], [vid. 2022-04-24].
URL https://www.programcreek.com/cpp/?code=mq1n%2FNoMercy%2FNoMercy-master%2FSource%2FClient%2FNM_Engine%2FINetworkScanner.cpp

[4] Wik, L. E.: Optional arguments with getopt_long(3). [online], rev. 2021-08-13, [vid. 2022-04-24].
URL https://cfengine.com/blog/2021/optional-arguments-with-getopt-long/