Projektová dokumentácia

Implementácia sieťového analyzátora

Simon Košina, xkosin09

23. apríla 2021

Obsah

1 Úvod

Cieľom projektu bolo navrhnúť a vytvoriť sieťový analyzátor, ktorý bude schopný na určitom sieťovom rozhraní zachytávať a filtrovať pakety. Výsledný program je schopný zachytávať TCP, UDP, ICMPv4, ICMPv6 pakety a ARP rámce, dané pakety sú vypisované na štandardný výstup.

```
$ ./ipk-sniffer --interface lo
2021-04-21T15:50:10.408+02:00 127.0.0.1 : 51376 > 127.0.0.53 : 53, length 80 bytes
000000:  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0x0010:  00 42 d9 d8 40 00 40 11  62 9c 7f 00 00 01 7f 00   .B..@.@. b.......
0x0020:  00 35 c8 b0 00 35 00 2e  fe 75 29 d2 01 20 00 01   .5...5.. .u).. ..
0x0030:  00 00 00 00 00 01 09 6c  6f 63 61 6c 68 6f 73 74   .......l ocalhost
```

Obr. 1: Príklad výstupu

2 Riešenie

Riešenie pozostáva z niekoľkých krokov. Program najprv spracuje argumenty z príkazového riadka, neskôr je vytvorený odpovedajúci filter a je získaný deskriptor pre dané sieťové rozhranie. Deskriptor sa následne využije na získanie paketov a ich výpis.

2.1 Spracovávanie argumentov, výpis rozhraní

Argumenty sú analyzované vo funkcii `main()`. O dlhé verzie argumentov sa postará sekvenčný prechod polom `argv` a krátke argumenty má na starosti funkcia `getopts()`.
Pre argumenty `-t` alebo `--tcp`, `-u` alebo `--udp`, `--arp` a `--icmp` je vytvorené pole príznakov `proto_flags` a pole `proto_names`, kde sú v rovnakom poradí uložené odpovedajúce čiastkové reťazce slúžiace na zloženie výsledného reťazca pre filtrovanie paketov.
V prípade, že nebude špecifikované sieťové rozhranie, je program po spracovaní argumentov ukončený a pomocou funkcie `printInterfaces()` je vypísaný zoznam rozhraní.
Funkcia `printInterfaces()` získa volaním funkcie `pcap_findalldevs()` zoznam dostupných rozhraní, ktorý sekvenčne prejde, pričom vypíše ich nźvy.

2.2 Zachytávanie paketov

Najprv je získaný deskriptor uvedeného rozhrania pomocou funkcie `getCaptureHandle()`. Jej argumentmi sú názov rozhrania a reťazec, popisujúci filter. `getCaptureHandle()` využíva hneď niekoľko funkcií z knižnice `pcap.h` (5). Začneme funkciou `pcap_open_live()`, ktorá slúži na získanie, už spomenutého deskriptoru rozhrania. Následne je volaná funkcia `pcap_lookupnet`, aby sme zistili sieťovú adresu a masku rozhrania. Maska je využitá vo funkcii `pcap_compile`, kde dochádza ku kompilácii reťazca na filtrujúci program. Zkompilovaný program je aplikovaný na získaný deskriptor pomocou funkcie `pcap_setfilter`.
Pred započatím zachytávania paketov je pomocou funkcie `signal()` definovaná reakcia na signály `SIGINT`, `SIGTERM` a `SIGQUIT`. Tieto signály sú obslúžené funkciou `teardown()`, ktorá zapezpečí uvoľnenie aktuálne využívaných zdrojov.
Samotné zachytávanie paketov je vyriešené pomocou funkcie `pcap_loop()`, ktorej je predaný odpovedajúci deskriptor rozhrania, počet paketov na zachytenie a obslužná funkcia `printPacket()`.

2.3 Spracovávanie odchytených paketov

Odchytené pakety sú predané funkcii `printPacket()` a to v podobe štruktúri `pcap_pkthdr` a ukazovateľa na pole jednotlivých bytov paketu. Tieto údaje, spolu s inými sú následne predávané funkciám pre spracovanie hlavičiek jednotlivých vrstiev.
Funkcia `handleEthernet()` má na starosti spracovanie Ethernet hlavičky paketu. Po kontrole veľkosti obdržaných dát získa informácie o následujúcom protokole.
V prípade, že sa jedná o ARP paket je zavolaná funkcia `printARP()`, ktorá zabezpečí výpis paketu a následne je spracovávanie daného paketu ukončené. Vo výpise je namiesto zdrojovej a cieľovej IP adresy uvedená zdrojová a cieľová MAC adresa.

```
2021-04-21T18:07:30.219+02:00 0:0:0:0:0:0 > ff:ff:ff:ff:ff:ff, length 42 bytes
000000:  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                     ........ ..
```

Obr. 2: Výpis ARP paketu

IPv4 pakety sú obslúžené funkciou `handleIPv4()`. Tá opäť vykoná kontrolu paketu a zistí jeho zdrojovú a cieľovú IPv4 adresu. Výstupom funkcie je dĺzka IPv4 hlavičky, teda počet bitov, ktoré je potrebné preskočiť aby sme sa dostali k hlavičke ďalšej vrstvy.
Obdobne sú spracovávané aj IPv6 pakety vo funkcii `handleIPv6`. Namiesto IPv4 adries sú uložené IPv6 adresy. Hlavný rozdiel je v ošetrení rozšírených IPv6 hlavičiek. Z každej IPv6 hlavičky vieme zistiť typ následujúcej hlavičky. V tele funkcie je preto cyklus, ktorý prechádza jednotlivými hlavičkami až pokiaľ nenarazí na koniec. Funkcia vracia akumulovanú dĺžku všetkých IPv6 hlavičiek.
Následujce spracovanie TCP, UDP alebo ICMP hlavičiek, čo zabezpečujú funkcie `handleTCP()`, `handleUDP()` a `handleICMP()`. Ich hlavnou úlohou, až na `handleICMP()`, je získanie čísla zdrojového a cieľového portu. Funkcia `handleICMP()` zisťuje typ a následný kód tejto správy, čo sa odrazí aj na samotnom výpise paketu.

```
2021-04-21T18:43:24.801+02:00 127.0.0.1 > 127.0.0.1, type 8, code 0, length 42 bytes
000000:  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0x0010:  00 1c 84 2e 00 00 40 01  f8 b0 7f 00 00 01 7f 00   ......@. ........
0x0020:  00 01 08 00 94 fa 63 00  00 05                     ......c. ..
```

Obr. 3: Výpis ICMP paketu

2.4 Výpis dát paketov

Po získaní potrebných údajov je možné vypísať dáta odchyteného packetu. Informácie o adresách a dlźke sú vypísané vo funkcii `printPacket()`.
Pre výpis času je využitá funkcia `printTime()`, ktorá získa časové údaje priamo z štruktúry `pcap_pkthdr`. Funkcia postupne vytvára a modifikuje reťazec s časom, tak aby odpovedal formátu. Táto funkcia bola inšpirovaná odpoveďou(1) na otázku položenú na fóre `stackoverflow.com`.

Ďalej následuje výpis dát paketu, ktorý sa uskutoční volaním funkcie `printPacketData()`. Funkcii sú prostredníctvom argumentov predané informácie o dĺžke paketu a ukazateľ na pole bytov, predstavujúce dáta. Jednotlivé byty sú vypísané na štandardný výstup ako v hexadecimálnom, tak aj v ASCII formáte spoločne s počiatočným offsetom.

3 Testovanie

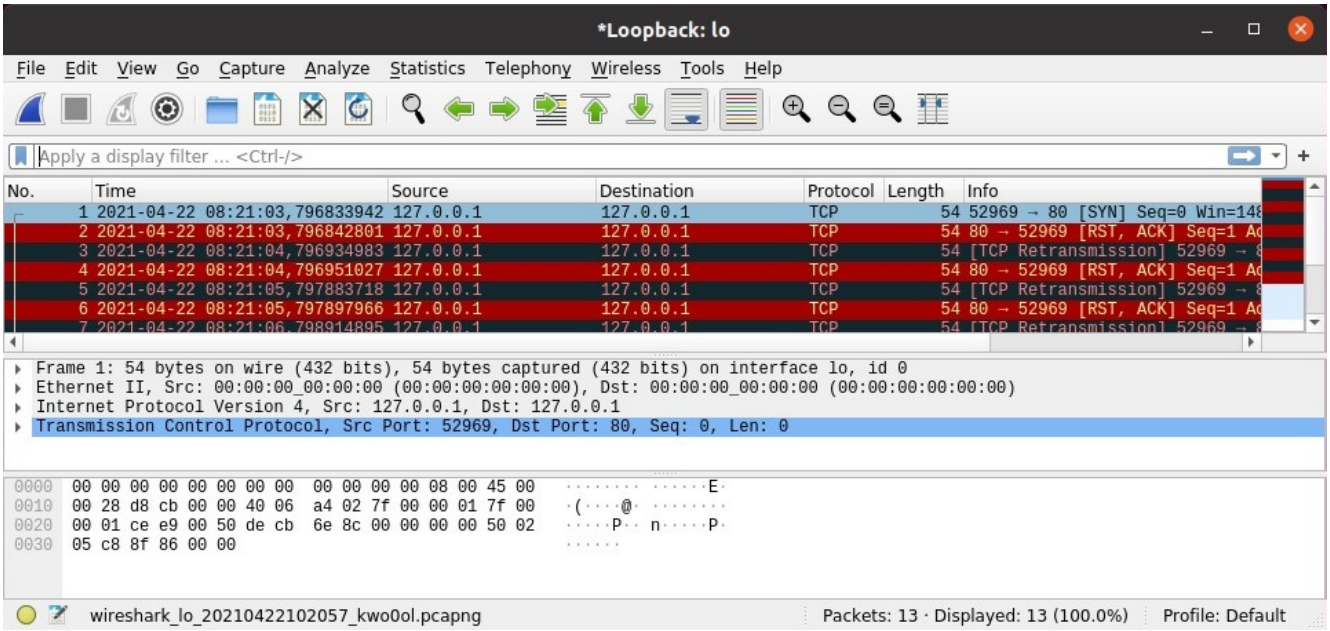
Na generovanie paketov bol využitý nástroj `nping`(2) s rôznymi argumentami pre jednotlivé typy paketov. Následne bol spustený program `ipk-sniffer` s príslušnými parametrami a zároveň s ním aj program Wireshark(3), tak aby výstupy týchto dvoch programov mohli byť porovnávané.

```
$ nping localhost --tcp
Starting Nping 0.7.80 ( https://nmap.org/nping ) at 2021-04-22 10:21 CEST
SENT (0.0042s) TCP 127.0.0.1:52969 > 127.0.0.1:80 S ttl=64 id=55499 iplen=40
seq=3737874060 win=1480
RCVD (0.0042s) TCP 127.0.0.1:80 > 127.0.0.1:52969 RA ttl=64 id=0 iplen=40 seq=0 win=0
```

Obr. 4: Spustenie programu nping(2) pre zasielanie TCP paketov

```
$ ./ipk-sniffer --interface lo
2021-04-22T10:21:03.796+02:00 127.0.0.1 : 52969 > 127.0.0.1 : 80, length 54 bytes
000000:  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0x0010:  00 28 d8 cb 00 00 40 06  a4 02 7f 00 00 01 7f 00   .(....@. ........
0x0020:  00 01 ce e9 00 50 de cb  6e 8c 00 00 00 00 50 02   .....P.. n.....P.
0x0030:  05 c8 8f 86 00 00
```

Obr. 5: Výpis daného paketu programom `ipk-sniffer`



Obr. 6: Daný paket zachytený programom Wireshark(3)

4 Záver

Môžem zhodnotiť, že projekt bol pre mňa veľmi prínosný. Rýchlo sa zorientovať v danej problematike mi pomohla séria návodov(4) od pána menom Martìn Casado, kde bola vysvetlená práca s knižnicou libpcap(5). Projekt mi pomohol lepšie pochopiť látku preberanú na prednáškach a prakticky si overiť dané poznatky.
Riešenie by malo byť plne funkčné podľa špecifikácie v zadaní. Jediným rozdielom je snáď výpis ARP a ICMP paketov.

Referencie

[1] chux - Reinstate Monica
(`https://stackoverflow.com/users/2410359/chux-reinstate-monica`),
I'm trying to build an RFC3339 timestamp in C. How do I get the timezone offset?,
URL (verzia: 21. Novembra 2018): `https://stackoverflow.com/questions/48771851`

[2] Nping. 0.7.80. August 2019 . Luis MartinGarcia, Fyodor. `https://www.nmap.org/nping`. 16.4.2021.

[3] Wireshark. 3.2.3. Apríl 2020. Gerald Combs. `https://www.wireshark.org/`. 16.4.2021.

[4] Martìn Casado. Apríl 2006. Zobrazené 16.4.2021.
`http://yuba.stanford.edu/~casado/pcap/section1.html`.

[5] TCPDUMP/LIBPCAP public repository. December 2020. Zobrazené 16-18.4.2021.
`http://www.tcpdump.org/`.