

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÝCH TECHNOLOGIÍ**

**IPK projekt
SNIFFER PACKETOV**

Matúš Tvarožný
xtvaro00

24.4.2021

OBSAH

OBSAH	2
ÚVOD	3
IMPLEMENTÁCIA.....	3
VSTUPNÉ ARGUMENTY	3
PRÁCA SNIFFERU, POUŽITÉ FUNKCIE.....	4
TESTOVANIE.....	5
PRÍKLADY SPUSTENIA	8
ZDROJE	8

ÚVOD

Toto je dokumentácia k druhému projektu z predmetu IPK. Vybral som si zadanie ZETA, čiže analyzátor (ďalej ako sniffer) sieťových paketov, ktorý by mal fungovať podobne ako aplikácia WireShark. Sniffer dokáže zachytávať IPv4(ICMPv4, TCP, UDP), IPv6(ICMPv6) a ARP pakety, v prípade potreby len tie, ktoré prechádzajú zadaným portom. Jeho výstupom je vypísanie zachyteného paketu, čas jeho zachytenia vo formáte RFC3339 a oboch adries(v prípade ARP paketov MAC adries) a portov(v prípade TCP a UDP paketov), ktorými paket prešiel.

IMPLEMENTÁCIA

Projekt som implementoval v jazyku C++. Cela implementácia snifferu je obsiahnutá v súbore `ipk-sniffer.cpp`. Základ projektu je postavený na knižnici `pcap`, ktorej funkcie sa starajú o všetku prácu s paketmi a ich odchytávaním. Za zmienku tiež stoja knižnice ako `chrono` na výpis času v požadovanom formáte RFC3339, knižnica `netinet` napríklad na prácu so štruktúrami podporovaných paketov a par knižníc jazyka C++, ktoré mi poskytli nezbité funkcie pri práci na projekte.

VSTUPNÉ ARGUMENTY

Sniffer je možné spúšťať s viacerými vstupnými argumentami(prepláčačmi). Na ich poradí nezáleží.

Prvým z nich je prepínač `-i/--interface rozhranie`, je to povinný argument ktorý programu hovorí o tom na akom rozhraní sa budú analyzovať pakety. Ak tento argument nie je zadaný alebo je zadaný bez rozhrania program vypíše všetky dostupné rozhrania na aktuálnom zariadení a skončí. Rovnako sa program chová ak nie je zadaný žiadny vstupný argument.

Argument `-p port` slúži na filtrovanie paketov podľa portu. Tento argument nie je povinný a ak nebude zadaný, analyzované budú pakety na všetkých portoch na danom zariadení.

Argument `-n pocet_paketov` určuje počet paketov, ktoré budú vypísane na `stdout`, opäť nie je povinný a ak nebude uvedený vypíše sa jeden paket.

Nasledujúca skupina prepínačov slúži na nastavenie filtra paketov podľa ich protokolov. Ak nie je zadaný žiadny. analyzované budú všetky pakety, naopak ak je zadaný aspoň jeden, budú analyzované práve pakety daného typu. Sú to nasledujúce:

`-t/--tcp, -u/--udp, --icmp, --arp`

PRÁCA SNIFFERU, POUŽITÉ FUNKCIE

Vo funkcii `main()` je po definícii pár potrebných premenných volaná funkcia `ArgumentParser()`, ktorá má na starosti prácu so vstupnými argumentami.

Hneď na jej začiatku je volaná funkcia `LoadInterfaces()`, ktorá pomocou funkcie z knižnice `pcap pcap_findalldevs()` načíta všetky dostupné rozhrania na aktuálnom zariadení a vracia ukazovateľ na prvé rozhranie v zozname. Tento ukazovateľ je uložený do premennej `list_of_interfaces`. V prípade potreby tieto zariadenia vypísať na `stdout` je definovaná funkcia `PrintInterfaces()`. Prvý prechod všetkými vstupnými argumentami je implementovaný z dôvodu, ak by bol prepínač `-i/--interface rozhranie` zadaný bez rozhrania, napríklad posledný. Vypísanie všetkých dostupných rozhraní má prednosť. Na kontrolu či je zadané rozhranie validné a nachádza sa v zozname dostupných rozhraní je definovaná funkcia `CheckInterface()`, ktorá v prípade, ak je rozhranie nevalidné vypíše na `stderr` chybovú hlášku a program je ukončený. V poslednom rade sú tu nastavené aj globálne premenné.

Hneď po skončení tejto funkcie je volaná ďalšia funkcia a to `Filter()`. Ta sa stará o nastavenie filtra na základe globálnych premenných. `Filter` je uložený v globálnej premennej `filter` ako obyčajný reťazec, ktorý ale zodpovedá požadovanej syntaxi funkcií `pcap_compile()`, ktorá je volaná neskôr.

Po dobehnutí funkcie `Filter()` sa na rad dostáva ďalšia funkcia z knižnice `pcap` a to `pcap_open_live()`, ktorá je základom celej implementácie, otvára dane rozhranie a vracia jeho popisovač.

Teraz potrebujeme získať masku a IP siete na to nám posluží funkcia `pcap_lookupnet()`. Z tejto dvojice potrebujeme iba masku kvôli už spomínanej funkcii `pcap_compile()`, ktorá si ju pýta ako vstupný parameter a prichádza na rad ako ďalšia.

`Pcap_compile()` slúži na skompilovanie filtra z reťazca do potrebného formátu `struct bpf_program`. Ak kompilácia prebehne úspešne (funkcia nevráti `PCAP_ERROR`) prichádza na rad funkcia na nastavenie filtra `pcap_setfilter()`.

Posledná potrebná funkcia `pcap_loop()` je volaná až teraz, zachytáva pakety podľa nastaveného filtra, a prebehne toľkokrát (toľko paketov zachytí), koľko jej je zadané. Jej tretím parametrom je funkcia `PacketSniffer()`.

Táto funkcia je veľmi dôležitá a komplexná, prebehne zakaždým ako je zachytený vyhovujúci paket. Najprv zavolá funkciu `PrintTime()`, ktorá vypíše aktuálny čas (čiže čas kedy bol paket zachytený) v požadovanom formáte. Prvý „switch“ rozhodne podľa premennej `ether_type`, ktorá sa nachádza v štruktúre `ether_header` či sa jedná o paket IPv4, IPv6 alebo priamo ARP. V vnútri „casu“ pre IPv4 sa nachádza ďalší switch, ktorý rozhodne o tom k akému protokolu daný paket patrí, pre korektný prístup k údajom ako sú IP adresy a čísla portov, cez ktorý sa k nám paket dostal. V prípade IPv6 sa mi podarilo nasimulovať (teda aj implementovať) iba ICMPv6 pakety. Tie na výpis adresy volajú funkciu `Ipv6Expander()`. Ak je načítaný paket protokolu ARP sú vypísané obe jeho MAC adresy. Na výpis paketu ako takého sa v každom prípade používa funkcia `PrintPacket()`.

Na konci je zatvorené „zariadenie“ na ktorom sa „sniffovalo“ a uvoľnená alokovaná pamäť funkciou `pcap_close()` a `pcap_freecode()`.

TESTOVANIE

Na väčšinu testov mi vystačilo WSL a ručná kontrola načítaných paketov s aplikáciou Wireshark. Keďže môj poskytovateľ internetu nepodporuje IPv6 a počítač mám pripojený pomocou wi-fi dostával som len TCP a UDP pakety(IPv4). Tieto po zachytení vyzerali nejak takto:

```
21 21:17:57.006212 192.168.160.1 192.168.175.255 UDP 86 57621 → 57621 Len=44
<
> Frame 21: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{B77F0CC5-2F01-4751-8D44-8016522E13CB}
> Ethernet II, Src: Microsof_d6:d2:56 (00:15:5d:d6:d2:56), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 192.168.160.1, Dst: 192.168.175.255
> User Datagram Protocol, Src Port: 57621, Dst Port: 57621
> Data (44 bytes)
<
0000  ff ff ff ff ff ff 00 15 5d d6 d2 56 08 00 45 00  .....]..V..E.
0010  00 48 d5 f6 00 00 80 11 00 00 c0 a8 a0 01 c0 a8  .H.....
0020  af ff e1 15 e1 15 00 34 04 2a 53 70 6f 74 55 64  .....4.*SpotUd
0030  70 30 7e 0f 0a ff 89 99 10 bf 00 01 00 04 48 95  p0~.....H.
0040  c2 03 13 34 a7 e9 3e 00 b5 48 e2 b4 ba 48 c0 96  ...4...>..H...H..
0050  61 d9 61 f0 e1 9a  a.a...

$ sudo ./ipk-sniffer -i eth0
2021-04-24T21:17:27.046+02:00 192.168.160.1 : 57621 > 192.168.175.255 : 57621, length 86 bytes, UDP
000000: ff ff ff ff ff ff 00 15 5d d6 d2 56 08 00 45 00  .....]..V..E.
000010: 00 48 d5 f5 00 00 80 11 93 5d c0 a8 a0 01 c0 a8  .H.....].....
000020: af ff e1 15 e1 15 00 34 04 2a 53 70 6f 74 55 64  .....4.*SpotUd
000030: 70 30 7e 0f 0a ff 89 99 10 bf 00 01 00 04 48 95  p0~.....H.
000040: c2 03 13 34 a7 e9 3e 00 b5 48 e2 b4 ba 48 c0 96  ...4...>..H...H..
000050: 61 d9 61 f0 e1 9a  a.a...
```

Na simuláciu ARP protokolov som používal nasledujúce:

```
$ arp -a
DESKTOP-A097TOK.mshome.net (192.168.160.1) at 00:15:5d:d6:d2:56 [ether] on eth0
$ sudo arping 192.168.160.1
ARPING 192.168.160.1
42 bytes from 00:15:5d:d6:d2:56 (192.168.160.1): index=0 time=268.900 usec
42 bytes from 00:15:5d:d6:d2:56 (192.168.160.1): index=1 time=194.500 usec
192 21:27:01.082151 Microsoft_d0:f9:29 Broadcast ARP 58 Who has 192.168.160.1? Tell 192.168.164.3
<
> Frame 192: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{B77F0CC5-2F01-4751-8D44-8016522E13CB},
> Ethernet II, Src: Microsof_d0:f9:29 (00:15:5d:d0:f9:29), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Address Resolution Protocol (request)
<
>
0000  ff ff ff ff ff ff 00 15 5d d0 f9 29 08 06 00 01  .....]..)...
0010  08 00 06 04 00 01 00 15 5d d0 f9 29 c0 a8 a4 26  .....]..)...&
0020  00 00 00 00 00 00 c0 a8 a0 01 00 00 00 00 00 00  .....
0030  00 00 00 00 00 00 00 00  .....

$ sudo ./ipk-sniffer -i eth0 --arp
2021-04-24T21:27:01.286+02:00 0:15:5d:d0:f9:29 > ff:ff:ff:ff:ff:ff, length 58 bytes, ARP
000000: ff ff ff ff ff ff 00 15 5d d0 f9 29 08 06 00 01  .....]..)...
000010: 08 00 06 04 00 01 00 15 5d d0 f9 29 c0 a8 a4 26  .....]..)...&
000020: 00 00 00 00 00 00 c0 a8 a0 01 00 00 00 00 00 00  .....
000030: 00 00 00 00 00 00 00 00  .....

```

Na simuláciu ICMPv6 protokolov:

```
$ sudo ping6 fe80:0000:0000:0000:0000:0000:0000:1
PING fe80:0000:0000:0000:0000:0000:0000:1(fe80::1) 56 data bytes
From fe80::215:5dff:fed0:f929%eth0 icmp_seq=1 Destination unreachable: Address unreachable
From fe80::215:5dff:fed0:f929%eth0 icmp_seq=2 Destination unreachable: Address unreachable
From fe80::215:5dff:fed0:f929%eth0 icmp_seq=3 Destination unreachable: Address unreachable
```

Kde fe80:0000:0000:0000:0000:0000:0000:1 je adresa nášho domáceho wi-fi routera.

```
1084 21:36:52.138988 fe80::215:5dff:fed0... ff02::1:ff00:1 ICMPv6 86 Neighbor Solicitation for fe80::1 from 00
<
>
> Frame 1084: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{B77F0CC5-2F01-4751-8D44-8016522E13CB},
> Ethernet II, Src: Microsof_d0:f9:29 (00:15:5d:d0:f9:29), Dst: IPv6mcast_ff:00:00:01 (33:33:ff:00:00:01)
> Internet Protocol Version 6, Src: fe80::215:5dff:fed0:f929, Dst: ff02::1:ff00:1
> Internet Control Message Protocol v6
<
>
0000 33 33 ff 00 00 01 00 15 5d d0 f9 29 86 dd 60 00 33.....]..).
0010 00 00 00 20 3a ff fe 80 00 00 00 00 00 00 02 15 ...:.....
0020 5d ff fe d0 f9 29 ff 02 00 00 00 00 00 00 00 00 ]....).
0030 00 01 ff 00 00 01 87 00 cd 7c 00 00 00 00 fe 80 .....|.
0040 00 00 00 00 00 00 00 00 00 00 00 00 01 01 01 .....
0050 00 15 5d d0 f9 29 ...]..)
```

```
$ sudo ./ipk-sniffer -i eth0
2021-04-24T21:36:52.566+02:00 fe80:0000:0000:0000:0215:5dff:fed0:f929 > ff02:0000:0000:0000:0001:ff00:0001, length
86 bytes, ICMPV6
000000: 33 33 ff 00 00 01 00 15 5d d0 f9 29 86 dd 60 00 33.....]..).
000010: 00 00 00 20 3a ff fe 80 00 00 00 00 00 00 02 15 ...:.....
000020: 5d ff fe d0 f9 29 ff 02 00 00 00 00 00 00 00 00 ]....).
000030: 00 01 ff 00 00 01 87 00 cd 7c 00 00 00 00 fe 80 .....|.
000040: 00 00 00 00 00 00 00 00 00 00 00 00 01 01 01 .....
000050: 00 15 5d d0 f9 29 ...]..)
```

Na simuláciu ICMPv4 protokolov:

```
$ sudo nping --icmp -e eth0 192.168.160.1
Starting Nping 0.7.80 ( https://nmap.org/nping ) at 2021-04-24 23:42 CEST
SENT (0.0050s) ICMP [192.168.164.38 > 192.168.160.1 Echo request (type=8/code=0) id=38754 seq=1] IP [ttl=64 id=45355 ipl
en=28 ]
SENT (1.0055s) ICMP [192.168.164.38 > 192.168.160.1 Echo request (type=8/code=0) id=38754 seq=2] IP [ttl=64 id=45355 ipl
en=28 ]
1479 21:43:27.760460 192.168.164.38 192.168.160.1 ICMP 42 Echo (ping) request id=0xb091, seq=1/256
<
>
> Frame 1479: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{B77F0CC5-2F01-4751-8D44-8016522E13CB},
> Ethernet II, Src: Microsof_d0:f9:29 (00:15:5d:d0:f9:29), Dst: Microsof_d6:d2:56 (00:15:5d:d6:d2:56)
> Internet Protocol Version 4, Src: 192.168.164.38, Dst: 192.168.160.1
> Internet Control Message Protocol
<
>
0000 00 15 5d d6 d2 56 00 15 5d d0 f9 29 08 00 45 00 ..]..V..]..).E.
0010 00 1c c6 8e 00 00 40 01 ee d9 c0 a8 a4 26 c0 a8 .....@.....&..
0020 a0 01 08 00 47 6d b0 91 00 01 ....Gm....
```

```
$ sudo ./ipk-sniffer -i eth0 --icmp
2021-04-24T21:43:27.447+02:00 192.168.164.38 > 192.168.160.1, length 42 bytes, ICMP
000000: 00 15 5d d6 d2 56 00 15 5d d0 f9 29 08 00 45 00 ..]..V..]..).E.
000010: 00 1c c6 8e 00 00 40 01 ee d9 c0 a8 a4 26 c0 a8 .....@.....&..
000020: a0 01 08 00 47 6d b0 91 00 01 ....Gm....
```

Krátka ukážka, že všetko funguje ako ma aj s TCP protokolom:

```
$ sudo nping --tcp -e eth0 192.168.160.1
Starting Nping 0.7.80 ( https://nmap.org/nping ) at 2021-04-24 23:49 CEST
SENT (0.0036s) TCP 192.168.164.38:24558 > 192.168.160.1:80 S ttl=64 id=28333 iplen=40 seq=4147852323 win=1480
SENT (1.0043s) TCP 192.168.164.38:24558 > 192.168.160.1:80 S ttl=64 id=28333 iplen=40 seq=4147852323 win=1480
```

```

$ sudo ./ipk-sniffer -i eth0 --tcp
2021-04-24T21:49:41.607+02:00 192.168.164.38 : 80 > 192.168.160.1 : 24558, length 54 bytes, TCP
000000: 00 15 5d d6 d2 56 00 15 5d d0 f9 29 08 00 45 00 ..]..V..]...E.
000010: 00 28 6e ad 00 00 40 06 46 aa c0 a8 a4 26 c0 a8 .(n...@.F....&..
000020: a0 01 5f ee 00 50 f7 3b 34 23 00 00 00 00 50 02 ...P.;4#....P.
000030: 05 c8 59 04 00 00 ..Y...

1550 21:49:41.224591 192.168.164.38 192.168.160.1 [TCP] 54 24558 → 80 [SYN] Seq=0 Win=1480 Len=0
<
> Frame 1550: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{B77F0CC5-2F01-4751-8D44-8016522E13C8},
> Ethernet II, Src: Microsof_d0:f9:29 (00:15:5d:d0:f9:29), Dst: Microsof_d6:d2:56 (00:15:5d:d6:d2:56)
> Internet Protocol Version 4, Src: 192.168.164.38, Dst: 192.168.160.1
> Transmission Control Protocol, Src Port: 24558, Dst Port: 80, Seq: 0, Len: 0
<
0000 00 15 5d d6 d2 56 00 15 5d d0 f9 29 08 00 45 00 ..]..V..]...E.
0010 00 28 6e ad 00 00 40 06 46 aa c0 a8 a4 26 c0 a8 .(n...@.F....&..
0020 a0 01 5f ee 00 50 f7 3b 34 23 00 00 00 00 50 02 ...P.;4#....P.
0030 05 c8 59 04 00 00 ..Y...

```

Pokusy o nasimulovanie TCP/UDP IPv6 boli nasledovné. Ako prvú som skúsil použiť funkciu nping s nasledovnými parametrami, bohužiaľ bez výsledku.

```

$ sudo nping --udp -6 -e eth0 fe80:0000:0000:0000:0000:0000:0000:1

Starting Nping 0.7.80 ( https://nmap.org/nping ) at 2021-04-25 00:23 CEST
sendto in send_packet: sendto(4, packet, 8, 0, fe80::1, 28) => Invalid argument
Offending packet: BOGUS! Can't parse supposed IP packet
SENT (0.0030s) UDP ::53 > fe80::1:40125
sendto in send_packet: sendto(4, packet, 8, 0, fe80::1, 28) => Invalid argument

```

Vždy to bol iba ICMPv6 protokol.

Potom som nasimuloval UDP IPv6 paket na referenčných strojoch, ale „sniffer“ zachytil stále iba ICMPv6.

```

Received 1 packets, got 1 answers, remaining 0 packets
<IPv6 version=6 tc=0 fl=163438 plen=13 nh=ICMPv6 hlim=64 src=2100::102 dst=2100::101 |<ICMPv6EchoReply type=Echo Reply code=0 cksum=0x461e id=0x0 seq=0x0 data='MATUS' |>>
>>> u = UDP()
>>> u.dport = 4444
>>> u.display()
###[ UDP ]###
sport= domain
dport= 4444
len= None
chksum= None

>>> send(i/u/"MATUS SENT VIA IPv6 UDP\n")
.
Sent 1 packets.
>>>

>>> u = UDP()
>>> u.dport = 4444
>>> u.display()
###[ UDP ]###
sport= domain
dport= 4444
len= None
chksum= None

>>> p[0]
<Ether dst=00:0c:29:fa:c7:f0 src=00:0c:29:22:08:ab type=IPv6 |<IPv6 version=6 tc=0 fl=0 plen=32 nh=UDP hlim=64 src=2100::101 dst=2100::102 |<UDP sport=domain dport=4444 len=32 cksum=0x5e04 |<DNS id=19777 qr=0 opcode=10 aa=1 tc=0 rd=0 ra=0 z=1 ad=0 cd=1 rcode=refused qdcount=21280 ancount=21317 nscount=20052 arcount=8278 qd='' an='' ns='' ar='' |<Raw load='IA IPv6 UDP\n' |>>>>
>>>

```

```
student@student-vm:~/IPK$ sudo ./ipk-sniffer -i ens33
[sudo] password for student:
2021-04-25T01:00:13.893+02:00 2100:0000:0000:0000:0000:0000:0000:0101 > ff02:00
00:0000:0000:0000:0001:ff00:0102, length 86 bytes, ICMPV6
000000: 33 33 ff 00 01 02 00 0c 29 22 08 ab 86 dd 60 00 33.....)"....`.
000010: 00 00 00 20 3a ff 21 00 00 00 00 00 00 00 00 00 ... :!.....
000020: 00 00 00 00 01 01 ff 02 00 00 00 00 00 00 00 00 .....
000030: 00 01 ff 00 01 02 87 00 02 c1 00 00 00 00 21 00 .....!.
000040: 00 00 00 00 00 00 00 00 00 00 00 00 01 02 01 01 .....
000050: 00 0c 29 22 08 ab .....
student@student-vm:~/IPK$
```

RECIEVER

WireShark mi na referenčnom stroji nefungoval.

PRÍKLADY SPUSTENIA

```
$ sudo ./ipk-sniffer
$ sudo ./ipk-sniffer -i
$ sudo ./ipk-sniffer -i eth0
$ sudo ./ipk-sniffer -i eth0 -n 0
$ sudo ./ipk-sniffer -i eth0 -p 32652
$ sudo ./ipk-sniffer -i eth0 --tcp --udp
$ sudo ./ipk-sniffer -i eth0 -n 0 -t --icmp
$ sudo ./ipk-sniffer -i eth0 -n 5 -u -p 32652 --arp -t
```

ZDROJE

1. <https://linuxos.sk/clanok/packet-capturing-s-libpcap-1/>
2. <https://linuxos.sk/clanok/packet-capturing-s-libpcap-2/>
3. <https://stackoverflow.com/questions/54325137/c-rfc3339-timestamp-with-milliseconds-using-stdchrono>
4. <https://linux.die.net/man/3/pcap>
5. <https://www.cplusplus.com/reference/>
6. <https://stackoverflow.com/questions/3727421/expand-an-ipv6-address-so-i-can-print-it-to-stdout>
7. <https://samsclass.info/124/proj11/P10xN-scapy-ipv6.html>
8. <https://www.tcpdump.org/manpages/>
9. https://www.programcreek.com/cpp/?code=mq1n%2FNoMercy%2FNoMercy-master%2FSource%2FClient%2FNM_Engine%2FINetworkScanner.cpp