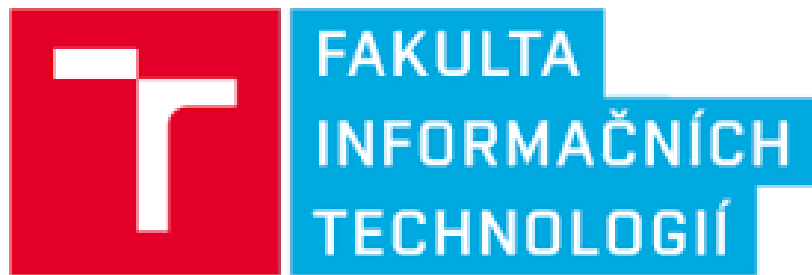


Vysoké učení technické v Brně
Fakulta informačních technologií



Počítačové komunikace a sítě

Dokumentace projektu č.2

Network Sniffer

Obsah

| | |
|--|---|
| Úvod | 3 |
| Krátky súhrn teórie..... | 3 |
| Implementácia | 4 |
| Spracovanie argumentov | 4 |
| Vytvorenie stringu na filtrovanie | 4 |
| Konfigurácia sieťového analyzátora | 4 |
| Zachytávanie paketov a ich výpis | 4 |
| Použitie..... | 5 |
| Spustenie program | 5 |
| Testovanie | 6 |
| Zdroje..... | 7 |

Úvod

Vitajte v dokumentácii k projektu **Network Sniffer**, odvážny čitatelia. Tento projekt je zameraný na zachytávanie a filtrovanie paketov na určitom rozhraní.

Krátky súhrn teórie

Ethernet hlavička – poskytuje zdrojovú a cieľovú MAC adresu na prvých 12 bytoch, od 14 bytu sa môžu vyskytovať dáta alebo hlavička protokolu vyššej vrstvy.

IP hlavička – či už IPv4 alebo IPv6 poskytuje informácie o zdrojovej alebo cieľovej IP adrese. Ale taktiež určujú špecifickejšie protokoly. V tejto implementácii sa jednalo o ICMP, IGMP, NDP a MLD

Typy správ ARP – protokol ARP získava informácie o tom aká MAC adresa sa vzťahuje na akú IP adresu a funguje na druhej vrstve

TCP a UDP hlavička – obsahuje viacero informácií, môj sniffer sa predovšetkým sústreďuje na získanie čísla zdrojového a cieľového portu

Implementácia

Projekt je implementovaný v jazyku C.

Spracovanie argumentov

Keďže je veľa typov argumentov a rôzne argumenty môžu aj nemusia mať hodnotu rozhodol som sa spracovávať argumenty pomocou selekcie. Správne zadaný argument sa uloží do predpripravenej štruktúry `Setup`, v prípade že sa jedná o hodnotu uloží sa tam daná hodnota a zmení sa boolean hodnota príslušného argumentu v `bool FLAGS[ENUM_LEN]`, ak sa jedná iba o zistenie či daný argument bol alebo nebol zadaný zmení sa iba boolean hodnota v poli. V prípade nesprávneho zadanie argumentov sa vypíše chybová hláška podľa situácie a korektne sa ukončí program.

Vytvorenie stringu na filtrovanie

Knižnice na konfiguráciu sieťového analyzátora očakávajú reťazec s informáciou o filtrovaní protokolov v určitom tvare, tvorbu tohto reťazca zabezpečuje funkcia `FilterStringCreating` pomocou už spomínaného poľa booleanov `setup.FLAGS[]`

Konfigurácia sieťového analyzátora

Prebieha pomocou knižnice `pcap` a jej funkcií. Za zmienku stojí hlavne funkcia `pcap_open_live` na ustanovenie spojenia, `pcap_compile` a `pcap_setfilter` na správne spracovanie a nastavenie filtru so zadanými protokolmi

Zachytávanie paketov a ich výpis

Zachytávanie paketov prebieha pomocou `pcap_loop` funkcie, ktorá požadovala implementáciu `packet_handler` funkcie. V tejto funkcii som definoval potrebné premenné a štruktúry na získanie a následné rozdelenie informácií z hlavičiek paketov.

Tieto informácie som vypísal v požadovanom formáte na štandardný výstup.

Použitie

Na preklad programu slúži Makefile, jeho spustenie je možné prevedením príkazu `make` cez terminal

Spustenie program

Execution

```
./ipk-sniffer [-i interface | --interface interface] {-p|--port-source|--port-destination port [--tcp|-t] [--udp|-u]} [--arp] [--icmp4] [--icmp6] [--igmp] [--mld] {-n num}
```

Program sa dá spustiť s veľa rôznymi kombináciami argumentov. Asi najdôležitejšie je si uvedomiť, že všetky protokoly sme schopný púšťať so všetkými a tým modifikovať filter vyhľadávania paketov. Základná hodnota počtu paketov na zobrazenie je jeden, toto číslo môžeme upraviť pomocou argumentu `-n`. Spustením `ipk-snifferu` bez parametrov alebo iba s parametrom `-i` alebo `--interface` bez názvu sieťového rozhrania nám vypíše dostupné sieťové rozhrania a kombinácia argumentu `-i` alebo `--interface` s inými argumentami je možná iba zadaním hodnoty sieťového rozhrania. Navyše argumenty je možné zadávať v ľubovoľnom poradí. Nesprávne zadanie argumentov je ošetrené chybovými hláškami a korektnou termináciou podľa danej situácie.

Testovanie

Správna funkčnosť programu bola testovaná pomocou software Wireshark a porovnávaníu výpisov z Wiresharku s mojimi výpismi. Ďalej všetky typy protokolov boli tiež otestované rôznymi spôsobmi ako sú napríklad pingy na nejaký vzdialený server a podobne.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|---------------------------|----------------|----------|--------|---|
| 1 | 0.000000000 | fe80::efad:9512:d121:a5e1 | ff02::fb | MDNS | 102 | Standard query 0x0000 PTR _pgpk ey-hkp._tcp.local, "QM" question |
| 2 | 0.000071503 | 10.0.2.15 | 224.0.0.251 | MDNS | 82 | Standard query 0x0000 PTR _pgpk ey-hkp._tcp.local, "QM" question |
| 3 | 1.352238093 | 10.0.2.15 | 34.107.221.82 | TCP | 74 | 39986 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=4193292750 TSe... |
| 4 | 1.384302203 | 34.107.221.82 | 10.0.2.15 | TCP | 60 | 80 → 39986 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 |
| 5 | 1.384367426 | 10.0.2.15 | 34.107.221.82 | TCP | 54 | 39986 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0 |
| 6 | 1.384611374 | 10.0.2.15 | 34.107.221.82 | HTTP | 355 | GET /canonical.html HTTP/1.1 |
| 7 | 1.384622269 | 34.107.221.82 | 10.0.2.15 | TCP | 60 | 80 → 39986 [ACK] Seq=1 Ack=302 Win=65535 Len=0 |
| 8 | 1.391706394 | 10.0.2.15 | 192.168.240.1 | DNS | 88 | Standard query 0x4175 A contile.services.mozilla.com |
| 9 | 1.391840753 | 10.0.2.15 | 192.168.240.1 | DNS | 88 | Standard query 0x0471 AAAA contile.services.mozilla.com |
| 10 | 1.437255332 | 34.107.221.82 | 10.0.2.15 | HTTP | 352 | HTTP/1.1 200 OK (text/html) |
| 11 | 1.437297203 | 10.0.2.15 | 34.107.221.82 | TCP | 54 | 39986 → 80 [ACK] Seq=302 Ack=299 Win=63942 Len=0 |
| 12 | 1.439192998 | 192.168.240.1 | 10.0.2.15 | DNS | 88 | Standard query response 0x0471 AAAA contile.services.mozilla.com |
| 13 | 1.444256890 | 192.168.240.1 | 10.0.2.15 | DNS | 104 | Standard query response 0x4175 A contile.services.mozilla.com A 34.117.237.239 |
| 14 | 1.446646112 | 10.0.2.15 | 34.117.237.239 | TCP | 74 | 56086 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3741700649 TS... |
| 15 | 1.476990329 | 34.117.237.239 | 10.0.2.15 | TCP | 60 | 443 → 56086 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 |
| 16 | 1.477088971 | 10.0.2.15 | 34.117.237.239 | TCP | 54 | 56086 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0 |
| 17 | 1.478227333 | 10.0.2.15 | 192.168.240.1 | DNS | 73 | Standard query 0xa861 AAAA ipv4only.arpa |

Frame 17: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface enp0s3
Ethernet II, Src: PCSysmtec_f6:94:75 (08:00:27:f6:94:75), Dst: IPv6mcast_ff02::fb
Internet Protocol Version 6, Src: fe80::efad:9512:d121:a5e1, Dst: ff02::fb
User Datagram Protocol, Src Port: 5353, Dst Port: 5353
Multicast Domain Name System (query)

0000 33 33 00 00 00 fb 08 00 27 f6 94 75 86 dd 60 0f 33.....'..u..`.
0010 d2 b5 00 30 11 ff fe 80 00 00 00 00 00 00 ef ad ...0.....
0020 95 12 d1 21 a5 e1 ff 02 00 00 00 00 00 00 00 00 ...!.....
0030 00 00 00 00 00 fb 14 e9 14 e9 00 30 fa 83 00 000....
0040 00 00 00 01 00 00 00 00 00 00 0b 5f 70 67 70 6b_pgpk
0050 65 79 2d 68 6b 70 04 5f 74 63 70 05 6c 6f 63 61 ey-hkp._tcp.loca
0060 6c 00 00 0c 00 01 l.....

```
Daniel@Ubuntu: ~
src MAC: 08:00:27:f6:94:75
dst MAC: 33:33:00:00:00:fb
frame length: 102 bytes
src IP: fe80::efad:9512:d121:a5e1
dst IP: ff02::fb
src port: 5353
dst port: 5353

0x0000: 33 33 00 00 00 fb 08 00 27 f6 94 75 86 dd 60 0f 33.....'..u..`.
0x0010: d2 b5 00 30 11 ff fe 80 00 00 00 00 00 00 ef ad ...0.....
0x0020: 95 12 d1 21 a5 e1 ff 02 00 00 00 00 00 00 00 00 ...!.....
0x0030: 00 00 00 00 00 fb 14 e9 14 e9 00 30 fa 83 00 00 .....0....
0x0040: 00 00 00 01 00 00 00 00 00 00 0b 5f 70 67 70 6b ....._pgpk
0x0050: 65 79 2d 68 6b 70 04 5f 74 63 70 05 6c 6f 63 61 ey-hkp._tcp.loca
0x0060: 6c 00 00 0c 00 01 l.....
```

Zdroje

IPK prezentácie

IPK prednášky

Článok o pcap knižnici

<https://vichargrave.github.io/programming/develop-a-packet-sniffer-with-libpcap/>

Skrátená dokumentácia pcap knižnice

<https://www.tcpdump.org/pcap.html>

EtherType wikipédia

<https://en.wikipedia.org/wiki/EtherType>