Vysoké učení technické v Brně
Fakulta informačních technologií

IPK – 2. projekt

Varianta ZETA: Sniffer paketů

24. dubna 2022 Vojtěch Hájek(xhajek51)

Obsah

1 Úvod

Cílem druhého projektu počítačových sítí bylo si vytvořit funkční komunikující program, podle vybrané specifikace. Můj projekt je variantou ZETA, tedy Sniffer paketů. Cílem projektu bylo seznámit se s danou problematikou a vymyslet vlastní implementaci programu.

2 Analyzátor paketů

Sniffer paket, či analyzátor paketů je počítačový program umožňující zachytávat, zaznamenávat a analyzovat komunikaci v počítačové síti . Zachycení paketu je proces zachycování a zaznamenávání komunikace. Když analyzátor zachytí paket se sítě je schopný například vypsat posílané data. Můžeme jej využít na analýzu problému v síti, odhalování pokusů o proniknutí do sítě, nebo detekci zneužívání sítě interními a externími uživateli [10].

3 Implementace

Projekt je implementován podle požadavků zadání, není k němu přidané žádné rozšíření. Po spuštení programu se inicializují potřebné proměnné, které se následně budou využívat a začne se zpracovávat argumenty. Pokud je program spuštěn bez přepínače -i / −−interface, vypíše se seznam všech dostupných komunikujících zařízení (viz sekce 3.2).

3.1 Zpracování argumentů

Zpracování argumentů využívá knihovny `getopt` [1], která umožňuje zpracování argumentů v jakémkoliv pořadí. Některé parametry podporují možnost dlouhého zápisu, či se zapisují pouze dlouhým zápisem (např. −−interface), pro tyto dlouhé parametry se využívá struktura `option` ve které jsou zapsány. Pomocí funkce `getopt_long()` se do rozdělí jednotlivé argumenty. Poté se v přepínači vyberou příslušné akce pro daný argument [9].

U každého parametru se rozlišuje nutnost zápisu argumentu. Parametr je buď může přímo vyžadovat (např. −−port vyžaduje argument number), či je tzv. `optional`, tedy argument může i nemusí být uveden (např. −−interface) a nebo je parametr úplně bez argumentu.

3.2 Vypsání všech dostupných komunikujících zařízení

Je zde využita knihovna `pcap` [11] a konkrétně funkce `pcap_findalldevs()`. Pomocí této funkce se do řetězce načte seznam všech dotupných komunikujících zařízeních a následně se vypíšou na výstup. Po vypsání se řetězec uvolní funkcí `pcap_freealldevs()`.

3.3 Otevření paketů

Pro otevření paketu se využívá knihovna `pcap` [11]. Nejdříve se komunikující zařízení otevře pomocí funkce `pcap_open_live()`, dále se zkontroluje datová linka a to pomocí funkce `pcap_datalink()`, ta musí být typu `DLT_EN10MB`, aby jsme zjistili zda zařízení podporuje Ethernet hlavičku. Pokud jedna z těchto funkcí nezdaří je vypsána chybová hláška, pomocí errbufferu, do kterého se uloží chybové hlášení [8].

```c
/**
 * @brief function to print all available devices
 * @param errbuf error buffer
 */
void findalldevs(char *errbuf) {
    pcap_if_t *list, *device;
    if (pcap_findalldevs(&list, errbuf)) {
        printf( format: "Error finding devices : %s", errbuf);
        exit( status: EXIT_FAILURE);
    }

    for (device = list; device != NULL; device = device->next) {
        printf( format: "%s \n", device->name);
    }
    pcap_freealldevs(list);
}
```

Obrázek 1: Ukázka kódu

3.4 Filtr paketů

Program filtruje pakety podle zadaných parametrů zadaných uživatelem, pokud uživatel žádný takový parametr nezadá filtruje se podle všech. Podporující typy paketů pro program: TCP, UDP, ICMPv4, ICMPv6, ARP.

Pro zpracování a vytvoření řetězce, podle kterého se bude filtrovat, je vytvořena funkce `filter_expression`. Do funkce se pošlou informace, který následně určí jak bude vypadat řetězec. Pokud jsou zadány dva typy paketů, je mezi nimi logikcý výraz `or` (např. `icmp or arp`), pokud je zádán číslo portu je ke kompatibilním typům přidán logický výraz `and` (např. `udp and port 10`). Řetězec je zapisován do globální proměnné `expression`.

Před samotným filtrováním paketů je potřeba řetězec zkompilovat, o to se zde stará funkce `pcap_compile()` a to jej zkompiluje do filtrovacího programu `struct bpf_program fp`. Filtrování se pak provede pomocí funkce `pcap_setfilter()`. Při případném selháním některé z funkcí ze vypíše chybová hláška uložená v `errbuffer`.

```c
// sets global string expression to filter
filter_expression(port_flag, port, tcp_flag, udp_flag, arp_flag, icmp_flag);

// compiles filter
if (pcap_compile(handle, &fp, expression, 0, net) == -1) {
    fprintf( stream: stderr, format: "Couldn't compile filter %s \n", errbuf);
    exit( status: EXIT_FAILURE);
}

// sets filter
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf( stream: stderr, format: "Couldn't set filter %s \n", errbuf);
    exit( status: EXIT_FAILURE);
}
```

Obrázek 2: Ukázka kódu

3.5 Zpracování paketů

O zpracování paketů se stará funkce `pcap_loop()`, která zavolá funkci `proccess_pakcet()` [3]. Funkce se volá vícekrát podle parametru -n, pokud parametr není zadán, je vychozí hodnota jedna. Zpracování paketu nejdříve zpracuje hlavičku. Podle které se vypíše čas timestamp, který je ve formátu `RFC3339` [12], k tomu slouží funkce `strftime` [4]. Dále se z Ether hlavičky vypíše source a destination MAC adresy. Poté se z hlavičky vypíše šířka framu pomocí `header->caplen` [2].

```
// gets timestamp string
char time[30];
strftime( s: time,  maxsize: sizeof(time),  format: "%Y-%m-%dT-%H:%M:%S",  tp: localtime( timer: &header->ts.tv_sec));

// prints timestamp
// https://man7.org/linux/man-pages/man3/strftime.3.html
printf( format: "timestamp: %s.%06ldZ\n", time, header->ts.tv_usec);

// prints source MAC, dst MAC and frame lenght
printf( format: "src MAC: %.2X:%.2X:%.2X:%.2X:%.2X:%.2X \n", eth->h_source[0], eth->h_source[1], eth->h_source[2],
       eth->h_source[3], eth->h_source[4], eth->h_source[5]);
printf( format: "dst MAC: %.2X:%.2X:%.2X:%.2X:%.2X:%.2X \n", eth->h_dest[0], eth->h_dest[1], eth->h_dest[2], eth->h_dest[3],
       eth->h_dest[4], eth->h_dest[5]);
printf( format: "frame lenght: %d bytes \n", header->caplen);
```

Obrázek 3: Ukázka kódu

Pomocí vytvořené struktury `ether_header` se určí ether typ, tedy jestli paket je ipv4, ipv6, nebo arp typu. Pro typy s ip hlavičkami se vytvoří struktura ip hlavičky `iphdr`, nebo `ip6_hdr`, tyto struktury jsou z knihovny `netinet` [7]. Podle kterých se určí typ protokolu [5]. Pro výpis source a destination IP adres se využíva funkce `print_ip()` a `print_ipv6()`. Pro výpis TCP a UDP portů se využívají funkce `print_tcp()` a `print_udp()`.

O výpis dat z paketu se dále stará funkce `print_data`. Jedná se o funcki, která vypisuje jednotlivé ASCII znaky, stejným způsobem jako vypisuje např. Wireshark [6].

```
0x0000:    00 50 56 FC 5C 00 00 0C 29 9A CD 7D 08 00 45 00        .PV.\...)..}..E.
0x0010:    00 3C C3 7E 40 00 40 06 85 39 C0 A8 FB 80 5D B8        .<.~@.@..9...◆].
0x0020:    D8 22 B1 EA 2D DC F1 13 4E 9A 00 00 00 00 A0 02        ."..-...N.......
0x0030:    FA F0 F2 32 00 00 02 04 05 B4 04 02 08 0A 19 3D        ...2...........=
0x0040:    51 C7 00 00 00 00 01 03 03 07                          Q.........
```

Obrázek 4: Ukázka výpisu dat z paketu

3.6 Ukončení programu

Pro předčasné ukončení programu může uživatel zavolat signál SIGINT, tedy stisknout klávesy CTRL+C. Před ukončením je zavolána funkce `pcap_close()`, která zavře soubory spojené s paketem a uvolní prostředky.

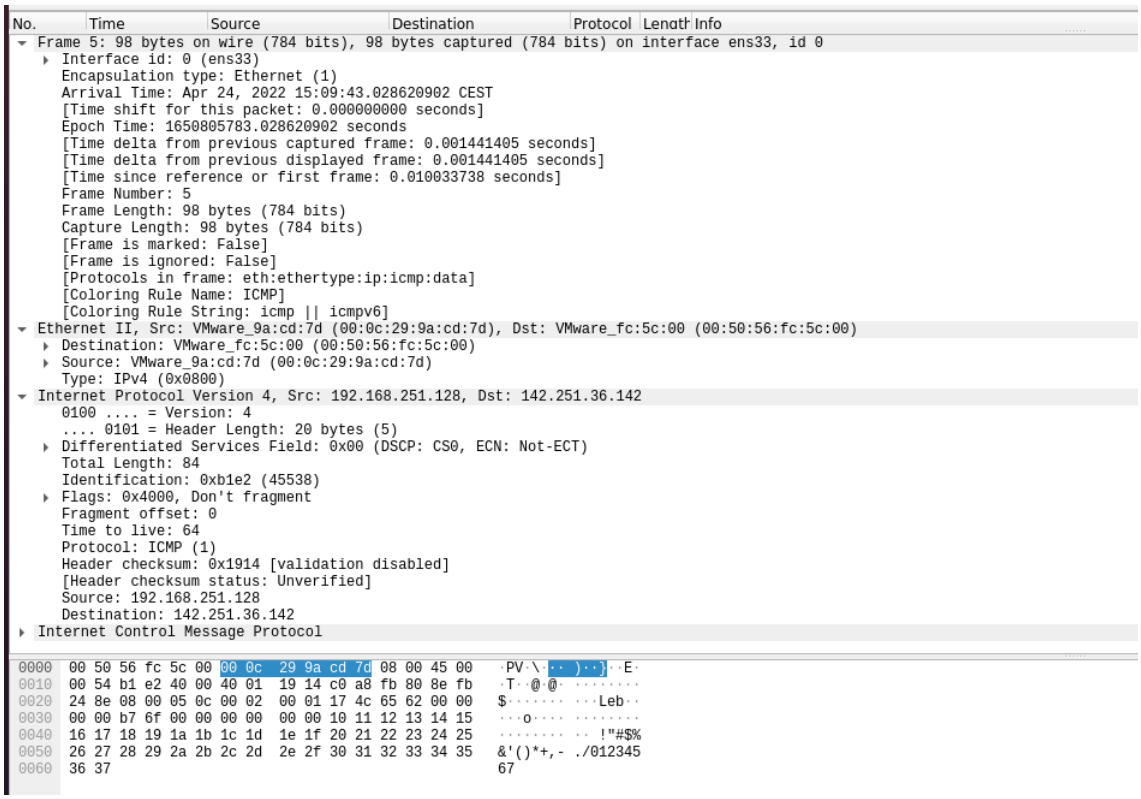
4 Testování

Testování probíhalo na poskytnuté školní virtuální mašině s operačním systémem Ubuntu 20.04.4 LTS. Pro porovnání výsledku testování byl použit program Wireshark [6]. Pro generování testovacích paketů byly využity například příkazy `curl` a `ping`.

```
student@student-vm:/mnt/hgfs/ipk$ sudo ./ipk-sniffer -i ens33 --icmp
timestamp: 2022-04-24T-15:09:43.028620Z
src MAC: 00:0C:29:9A:CD:7D
dst MAC: 00:50:56:FC:5C:00
frame lenght: 98 bytes
src IP: 192.168.251.128
dst IP: 142.251.36.142

0x0000:    00 50 56 FC 5C 00 00 0C 29 9A CD 7D 08 00 45 00         .PV.\...)..}..E.
0x0010:    00 54 B1 E2 40 00 40 01 19 14 C0 A8 FB 80 8E FB         .T..@.@.....◆..
0x0020:    24 8E 08 00 05 0C 00 02 00 01 17 4C 65 62 00 00         $..........Leb..
0x0030:    00 00 B7 6F 00 00 00 00 00 00 10 11 12 13 14 15         ...o............
0x0040:    16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25         .......... !"#$%
0x0050:    26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35         &'()*+,-./012345
0x0060:    36 37                                                   67
student@student-vm:/mnt/hgfs/ipk$
```

Obrázek 5: Testování ICMPv4



Obrázek 6: Testování ICMPv4 zobrazení ve Wireshark

5 Použité knihovny

- stdio.h
- stdlib.h
- getopt.h
- stdbool.h
- string.h
- signal.h
- pcap.h
- time.h
- arpa/inet.h
- netinet/if_ether.h
- netinet/ip.h
- netinet/udp.h
- netinet/ip6.h
- netinet/tcp.h

Použité zdroje

[1] Getopt library. [online]. Dostupné z: <https://linux.die.net/man/3/getopt>

[2] Pkhtdr structure. [online]. Dostupné z: <https://www.winpcap.org/docs/docs_412/html/structpcap__pkthdr.html>

[3] Proccess packet implementation. [online]. Dostupné z: <https://www.binarytides.com/packet-sniffer-code-c-linux/>

[4] Strftime function. [online]. Dostupné z: <https://man7.org/linux/man-pages/man3/strftime.3.html>

[5] Wiki: List of IP protocol numbers. [online]. Dostupné z: <https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers>

[6] Wireshark. [online]. Dostupné z: <https://www.wireshark.org/about-sharkfest.html>

[7] Netinet library. [online], 1997. Dostupné z: <https://pubs.opengroup.org/onlinepubs/7908799/xns/netinetin.h.html>

[8] Pcap implementation. [online], 2002. Dostupné z: <https://www.tcpdump.org/pcap.html>

[9] Getopt inspired implementation. [online], 2004. Dostupné z: <https://www.informit.com/articles/article.aspx?p=175771&seqNum=3>

[10] Wiki: Packet Analyzer. [online]. Dostupné z: <https://en.wikipedia.org/wiki/Packet_analyzer>

[11] Library pcap. [online], January 2022. Dostupné z: <https://www.tcpdump.org/manpages/pcap.3pcap.html>

[12] Tumorang, I.: Standard RFC-3339 format. [online]. Dostupné z: <https://medium.com/easyread/understanding-about-rfc-3339-for-datetime-formatting-in-software-engineeri