Vysoké učení technické v Brně

Fakulta informačních technologií

Počítačové komunikace a sítě

2021/2022

Dokumentace projektu

Varianta Zeta – Sniffer paketů

Ivan Golikov (xgolik00) 24.04.2022

1. Obsah

2. Úvod

Cílem řešeného projektu byl návrh a implementace síťového analyzátoru. Tento analyzátor bude schopný na určitém síťovém rozhraní zachytávat a filtrovat pakety.
Program podporuje zachytávání paketů na protokolu TPC, UDP a ICMP. Déle podporuje filtrování vypisovaných paketů podle protokolu, portu a rozhraní. Pakety se vypisují na `stdout` s oddělenou hlavičkou. Před vypsáním paketu bude vypsáná informace o paketu(čas, src a dst MAC adresa, delka framu, src a dst IP adresa, src a dst port).

3. Implementace

3.1. Základní údaje

K implementaci byla použita knihovna `<pcap.h>`. Jedná se o open source knihovnu jazyka C, která poskytuje API pro zachycení paketů z datových linek. Funguje na všech Unix systémech.
Samotný kód je uložen v jednom souboru `ipk-sniffer.c`. Kromě toho `pcap.h` poskytuje funkce a konstanty a hlavičky od `netinet` a `arpa.h` má potřebné datové struktury pro práci s pakety. Program obsahuje tři globální proměnné. První slouží jako deskriptor pro knihovnu `pcap.h`, ve druhé je uložena délka hlavičky transportní vrstvy paketů, ve třetí je počet packet pro vzpsaní informace na koncí program.
Celý program je rozložen na šest hlavních funkcí, které jsou popsány níže.

3.2. Funkce main()

Hlavní část programu, funkce `main()`, má za úkol zpracování argumentů programu, vytvoření řetězce pro filtrování a následné spuštění snifferu.
Argumenty jsou zpracovávány pomocí funkce `getopt_long()` , která umožňuje zpracovávat nejen krátké argumenty, ale i dlouhé s jejich hodnotami.
Součástí zpracování argumentů je i funkce `printHelp()`, jež vytiskne nápovědu k použití programu, a funkce `printInterfaces()`, která má za úkol v případě nespecifikování žádného rozhraní, na kterém chceme zachytávat pakety, vypsat všechny možné rozhraní.
Dále funkce kontroluje, zda jsou zadány filtry zároveň na TCP, UDP I na ICMP komunikaci. V takovém případě skončí s chybovým hlášením.
S pomocí funkce `signal()` nastavuje, jak má program správně skončit v případě přerušení systémem nebo uživatelem.

3.3. Funkce createPcapHandle()

Funkce `main()` volá funkci `createPcapHandle()`.
Tato funkce slouží zejména k získání soket deskriptoru pro sniffer paketů. Tento soket bude sloužit jako koncový bod při práci s pakety v datové vrstvě.

Tato funkce je posloupností pěti dalších funkcí knihovny pcap.h. Nejprve se zkontroluje, zda funkce má k dispozici rozhraní definované uživatelem. Pokud ne, pomocí funkce `pcap_lookupdev()` získá první dostupné. Tato funkce by v programu neměla být nikdy spuštěna, slouží pouze jako pojistka pro případné další použití této funkce.
Po kontrole je přistoupeno k funkci `pcap_open_live()`, která otevře vybrané rozhraní pro síťové přenosy a vrátí potřebný deskriptor.
Dále je zapotřebí aplikovat filtr. K tomu slouží následující trojice knihovních funkcí. `pcap_lookupnet()` nám získá masku sítě našeho připojení pro následující funkci. `pcap_compile()` převede námi vytvořený řetězec filtrů na knihovnou interpretovatelný kód, který může být aplikován pomocí třetí funkce `pcap_setfilter()`.
Tím je náš filtr použit přímo na deskriptoru síťového soketu pro sniffer paketů.

```
// open the device for live capture.
handle = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
    fprintf(stderr, "pcap_open_live(): %s\n", errbuf);
    return NULL;
}

// get network device source IP address and netmask.
if (pcap_lookupnet(device, &srcip, &netmask, errbuf) == PCAP_ERROR) {
    fprintf(stderr, "pcap_lookupnet(): %s\n", errbuf);
    return NULL;
}

// convert the packet filter epxression into a packet filter binary.
if (pcap_compile(handle, &bpf, filter, 1, netmask) == PCAP_ERROR) {
    fprintf(stderr, "pcap_compile(): %s\n", pcap_geterr(handle));
    return NULL;
}

// bind the packet filter to the libpcap handle.
if (pcap_setfilter(handle, &bpf) == PCAP_ERROR) {
    fprintf(stderr, "pcap_setfilter(): %s\n", pcap_geterr(handle));
    return NULL;
}
```

Obrázek 1 - Aplikace řetězcového filtru

3.4. Funkce getLinkHeaderLen()

Tato funkce je také volána z funkce `main()`. Slouží především k zjištění typu přenosu datové linky. Podle toho, o jaký typ se jedná, je následně definována délka hlavičky datové vrstvy. Tato informace je uložena v globální proměnné `linkhdrlen`.

3.5. Funkce packetHandler()

Tato funkce zajišťuje analýzu příchozího paketu. Má za úkol vypsat úvodní řádek dle zadání projektu a následně spustit funkci pro výpis surových dat. Ze všeho nejdříve tato funkce zjistí aktuální čas s vysokou přesností tak, aby se co nejvíce blížila času zachycení paketu.
Následně je určen typ IP komunikace. Program podporuje komunikaci přes IPv4 nebo IPv6. V obojích připadu se obě IP adresy zjistí pomocí funkce `inet_ntop()`.

3.6. Funkce printData()

Jedná se o funkci, která vypisuje data z paketu po jednotlivých bajtech. Vypisuje jak v hexadecimální, tak v ASCII podobě.
Funkci tvoří hlavní cyklus, který zajišťuje výpis všech bajtů. Velkou roli zde hraje logická proměnná `headPrinted`. Díky ní se může v pořádku oddělit hlavička a pokračovat ve výpisu bez narušení formátu paketu.

```
// if it finished the hexa line or head is printing
if (headPrinted ? ((i - afterHeaderOffset) % 16 == 0 && i != headSize) : (i != 0 && i % 16 == 0)) {

    printf(" ");
    for (j = i - 16; j < i; j++) {
        if (isprint(packet[j])) {
            printf("%c", (unsigned char) packet[j]);
        }
        else {
            printf(".");
        }

        if (j == i - 9) {
            printf(" ");
        }
    }
    printf("\n");

    if (line_counter < 10) printf("0x00%d:", line_counter++ * 10);
    else if (line_counter < 100) printf("0x0%d:", line_counter++ * 10);
    else printf("0x%d:", line_counter++ * 10);
}

// two spaces after the 0x0000 etc.
if ((headPrinted ? i - afterHeaderOffset : i) % 16 == 0) {
    printf("  ");
}

// space between each 8 bytes in hexa
if ((((headPrinted ? i - afterHeaderOffset : i) - 8) % 16) == 0) {
    printf(" ");
}

// this prints the hexa representation
printf("%02X ", (unsigned char) packet[i]);
```

Obrázek 2 - Zajištění vypsání dat ve formátu ASCII a hexadecimálním formátu

Funkce dále zajišťuje také vypsání počtu bajtů na začátku každého řádku.
Také se stará o vypisování mezer mezi bajty i mezi jednotlivými částmi výpisu.

```
timestamp: 2022-57-24T19:57:41.145847+01:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 90 bytes
src IP: ::1
dst IP: ::1
src port: 52756
dst port: 8080

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 DD 60 00  ........ ......`.
0x0010:  81 49 00 24 06 40 00 00  00 00 00 00 00 00 00 00  .I.$.@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01 CE 14  1F 90 56 BC 0C 05 7E B1  ........ ..V...~.
0x0040:  6D 8A 80 18 02 00 00 2C  00 00 01 01 08 0A 46 97  m......, ......F.
0x0050:  2E 7A 46 96 63 D0                                 .zF.c.

0x0060:  69 70 6B 0A                                       ipk.

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


0 packets captured
4 packets received by filter
0 packets dropped
```

Obrázek 3 - Příklad vypsání TCP paketu

3.7. Funkce stopCapture()

Tato funkce se volá při ukončení programu, ať už kvůli chybě, systémovému přerušení, uživatelskému přerušení nebo vypsání zadaného počtu paketů. Používá strukturu `pcap_stat` za účelem vypsání statistik o odchycených paketech. Důležité je uzavření soketu pro síťové přenosy.

```
void stopCapture() {

    struct pcap_stat stats;

    if (pcap_stats(handle, &stats) >= 0) {
        printf("\n%d packets captured\n", countPacket);
        printf("%d packets received by filter\n", stats.ps_recv);
        printf("%d packets dropped\n\n", stats.ps_drop);
    }

    pcap_close(handle);
    exit(EXIT_SUCCESS);
}
```

Obrázek 4 - Funkce pro ukončení programu

4. Testování programu

Program jsem testoval metodou porovnávání výstupu s ověřeným zdrojem. Pro generování testovacích paketů jsem využil příkaz `curl` případně `curl -6` pro IPv6 verzi.
Jako ověřený zdroj považuji program `wireshark`. Zde jsem si nastavil stejné filtrování jako v mém programu a následně mohl porovnat výstupy obou programů.

```
▸ Frame 93: 96 bytes on wire (768 bits), 96 bytes captured (768 bits)
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.53
▸ User Datagram Protocol, Src Port: 50309, Dst Port: 53
▸ Domain Name System (query)

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 52 ec a0 40 00 40 11  4f c4 7f 00 00 01 7f 00   ·R··@·@· O·......
0020  00 35 c4 85 00 35 00 3e  fe 85 ab f5 01 00 00 01   ·5···5·> ........
0030  00 00 00 00 00 00 01 06  76 6f 72 74 65 78 04 64 61   ·····v ortex·da
0040  74 61 09 6d 69 63 72 6f  73 6f 66 74 03 63 6f 6d   ta·micro soft·com
0050  00 00 01 00 01 00 00 29  04 00 00 00 00 00 00 00   ·····) ........
```

Obrázek 9 - Program wireshark použitý k testování

```
timestamp: 2022-30-24T20:30:03.861583+01:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 90 bytes
src IP: ::1
dst IP: ::1
src port: 41904
dst port: 8080

0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 DD 60 00  ........ ......`.
0x0010:  BE 32 00 24 06 40 00 00  00 00 00 00 00 00 00 00  .2.$.@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01 A3 B0  1F 90 D1 20 96 82 05 35  ........ ... ...5
0x0040:  69 4E 80 18 02 00 00 2C  00 00 01 01 08 0A AB 9D  iN.....,  ........
0x0050:  7A FF AB 9C B3 26                                 z....&

0x0060:  69 70 6B 0A                                       ipk.
```

Obrázek 8 - Výstup mého programu je shodný s výstupem z programu wireshark

5. Zdroje

Při zpracovávání projektu jsem využil pouze internetové zdroje. Níže se nachází seznam všech zdrojů, ze kterých jsem čerpal.

1. pcap_findalldevs example. *Embedded Guru* [online]. Dostupné z: http://embeddedguruji.blogspot.com/2014/01/pcapfindalldevsexample.html
2. GitHub - yuan901202/ethernet_packet_sniffer: [NWEN302] Ethernet Packet Sniffer in C. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 02.05.2020]. Dostupné z: https://github.com/yuan901202/ethernet_packet_sniffer
3. TCPDUMP/LIBPCAP public repository. *TCPDUMP/LIBPCAP public repository* [online]. Dostupné z: https://www.tcpdump.org/
4. WinPcap · Documentation. *WinPcap - Home* [online]. Copyright © 2018 [cit. 02.05.2020]. Dostupné z: https://www.winpcap.org/docs/default.htm
5. arpa_inet.h.0p - Linux manual page. *Michael Kerrisk - man7.org* [online]. Dostupné z: http://man7.org/linux/man-pages/man0/arpa_inet.h.0p.html
6. <netinet/in.h>. *The Open Group Publications Catalog* [online]. Copyright © 1997 The Open Group [cit. 02.05.2020]. Dostupné z: https://pubs.opengroup.org/onlinepubs/007908799/xns/netinetin.h.html
7. c - Problems finding pcap.h and linking - Stack Overflow. *Stack Overflow*

- *Where Developers Learn, Share, & Build Careers* [online]. Dostupné z: https://stackoverflow.com/questions/13337349/problems-finding-pcaph-and-linking

8. pcap_findalldevs(3): list of capture devices - Linux man page. *Linux Documentation* [online]. Dostupné z: https://linux.die.net/man/3/pcap_findalldevs
9. pcap-linktype(7) - Linux man page. *Linux Documentation* [online]. Dostupné z: https://linux.die.net/man/7/pcap-linktype
10. inet_ntop(3) - Linux manual page. *Michael Kerrisk - man7.org* [online]. Dostupné z: http://man7.org/linux/man-pages/man3/inet_ntop.3.html
11. header - What is the size of udp packets if i send 0 payload data in c#? - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Dostupné z: https://stackoverflow.com/questions/4218553/what-is-the-size-of-udppackets-if-i-send-0-payload-data-in-c