VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačných technologií

POČÍTAČOVÉ KOMUNIKÁCIE A SIETE

2019/2020

Dokumentácia k projektu č. 2

Varianta: ZETA: Sniffer paketov

25. apríla 2020 Dávid Oravec (xorave05)

Obsah

1 Uvedenie do problematiky - zadanie

Navrhnite a implementujte sieťový analyzátor v C/C++/C#, ktorý bude schopný na určitom sieťovom rozhraní zachytávať a filtrovať pakety.

1.1 Prvotné zoznámenie sa s problemtikou

Pred začatím bolo nutné sa zoznámiť s danou problematiko a zistiť, čo je vlastne paket, aký má tvar, čo obsahuje a podobne. Okrem študíjnych materiálov a prednášok, boli využité aj rôzne internetové zdroje viz. [1] [2]

1.2 Sniffovanie paketov

Pri sniffovaní paketov sa uvažujú iba pakety s protokolom TCP a/alebo UDP. Ostatné pakety s iným protokolom sú ignorované. Pri sniffovaní je potrebné si najprv otvoriť rozhranie, na ktorom budú tieto pakety zachytávané.

2 Implementácia

Projekt bol implementovaný v jazyku C s pomocou knižnice `libpcap`. Jazyk C bol najvhodnejšia voľba, kvôli znalosti daného jazyka. Projekt sa skladá z jedného modulu a jedného hlavičkového súboru.

2.1 Odovzdané súbory

- `ipk-sniffer.c` - jadro celého projektu, obsahuje implementáciu packet-snifferu
- `ipk-sniffer.h` - hlavičkový súbor s prototypmi funkcií a s potrebnými knižnicami
- `Makefile` - slúži na zlinkovanie súborov a skompilovanie projektu

2.2 ipk-sniffer.c

V tomto module sa nachádza celá logika projektu. Modul je členený do niekoľkých funkcií.

Na začiatok je volaná funkcia `initStruct()`. Účelom tejto funkcie je vytvoriť a inicializovať atribúty štruktúry `userArgs` na počiatočné hodnoty a vykonať alokáciu pamäte pre potrebné premenné. Pokiaľ sa alokácia nepodarí, z funkcie je vrátená nenulová hodnota a program končí.

Následne program pokračuje s vykonávaním funkcie `parseArgs(argc, argv)`. Pomocou funkcie `getopt()`, ktorá je súčasťou knižnice `getopt.h`, sa postupne parsujú argumenty a ukladajú sa do štruktúry `userArgs`, ktorá v sebe uchováva tieto dôležité hodnoty.

2.2.1 Podporované prepínače a obmedzenia

- **-i** *interface* $\Rightarrow$ rozhranie na ktorom sa pakety zachytávajú
  - ak prepínač nie je zadaný, vypíše sa zoznam dostupných rozhraní v systéme a program končí úspešne
  - ak je prepínač zadaný, ale nenasleduje za ním názov rozhrania a žiadny ďalší prepínač, program sa chová ako v predošlom bode a končí úspešne
  - ak je prepínač zadaný, ale je zadané zlé rozhranie / nie je zadané žiadne a následujú ďalšie prepínače, program vypíše chybovú hlášku a končí neúspešne

- **-p** *port* ⇒ port na ktorom sa pakety zachytávajú
  - voliteľný prepínač, ak nie je zadaný zachytávajú sa pakety na všetkých portoch
  - ak je zadaný, ale chýba jeho hodnota / je zadaná hodnota mimo interval $< 0, 65535 >$ / nie je zadaná číselná hodnota, tak program vypíše chybovú hlášku a končí neúspešne.
- **-n** *num* ⇒ počet zachytávaných paketov
  - ak je zadané záporné číslo / nie je zadaná číselná hodnota, program vypíše chybovú hlášku a končí neúspešne
- **-t** ⇒ zachytávané sú iba pakety s protokolom TCP
- **-u** ⇒ zachytávané sú iba pakety s protokolom UDP
  - ak nie je ani jeden z prepínačov zadaný alebo sú zadané obidva, tak sú zachytávané oba typy paketov

Po spracovaní argumentov je vykonávané otváranie rozhrania pre zachytávanie paketov. Tento proces zabezpečujú funkcie z knižnice `pcap.h` [3]. Najprv je získaná maska podsiete pomocou funkcie `pcap_lookupnet()`, ktorá je potrebná pri neskoršom aplikovaní filtra na rozhranie.

Nasleduje samotné otvorenie rozhrania pomocou funkcie `pcap_open_live()` a aplikácia filtra pomocou funkcií `pcap_compile()` a `pcap_setfilter()`. Tento filter je využívaný pri zachytávaní paketov na špecifickom porte, kedy je mu predávaný argument `userArgs.port`. Tento atribút v sebe uchováva hodnotu portu, ktorú zadal uživateľ na príkazovom riadku pri spustení (prepínač -p).

Pomocou funkcie `pcap_loop()` sa začne opakované volanie callback funkcie `processPacket()`.

2.3 Spracovanie paketu

Každý paket sa začína spracovávať v callback funckcií `processPacket()`. V tejto funkcií sú využité štruktúry z knižníc `netinet/ip.h`, `netinet/tcp.h` a `netinet/udp.h`. Na základe protokolu ip sa potom volá príslušná funkcia na výpis paketu.

2.4 Vytvorenie hlavičky výstupu

Hlavička výstupu projektu je vytváraná vo funkcii `getTimestamp()` a má podľa zadania tvar

```
čas IP|FQDN : port > IP|FQDN : port
```

Pri získavaní timestamp je použitý systémový čas. Vytváranie formátu s mikrosekundami bolo náročnejšie, avšak je to vyriešené zapísaním systémového času, vo formáte `HH:MM:SS`, do poľa znakov a následne sú do poľa pridané mikrosekundy aby formát odpovedal zadaniu.

Na získanie doménového mena z ip adresy je využitá funkcia `gethostbyaddr()`. Pokiaľ sa nepodarí získať doménové meno, tak v hlavičke ostane ip adresa.

2.5 Výpis paketov

Na výpis paketov sú vytvorené funkcie `printTCP()` a `printUDP()`.

2.5.1 printTCP()

Na začiatku funkcie sa vďaka pomocným atribútom `userArgs.tcp` a `userArgs.udp` vyrieši situácia, kedy uživateľ chce zobraziť iba UDP pakety, ale sniffer zachytí a pokúsi sa o zobrazenie TCP paketu. Je to v podstate implementovaný „pseudo-filter“.

Nasleduje výpis hlavičky na štandardný výstup a potom sa volá funkcia `dataFlush()`. Táto funkcia bola spolu z `print_hex_ascii_line()` prebratá z [4] a modifikovaná na účely projektu. Modifikácia spočívala v pridaní časti kódu, ktorá zabezpečuje oddelenie hlavičky paketu od dát v pakete prázdnym riadkom. Pre správne oddelenie, je funkcií `dataFlush()` poslaný obsah paketu, jeho veľkosť a veľkosť hlavičky paketu.

2.5.2 printUDP()

Táto funkcia bola implementovaná analogicky k funkcií `printTCP()`. Jediná zmena je implementovaná pri posielaní parametrov do funkcie `dataFlush()`. Keďže UDP paket má rozdielnu dĺžku hlavičky ako TCP paket, je nutné vypočítať túto veľkosť zvlášť.

3 Testovanie

Projekt bol testovaný na referenčnom Linux image pomocou open-source programu `Wireshark`. Príprava na testovanie a jeho proces prebiehal v 4 krokoch

1. otvorenie Wireshark v prvom termináli pomocou príkazu `sudo wireshark` a pripojenie sa na vybrané rozhranie
2. spustenie projektu v novom termináli pomocou príkazu
   `sudo ./ipk-sniffer -i` *`nazov_rozhrania_rovnaky_ako_vo_Wireshark`*
3. otvorenie tretieho terminálu, cez ktorý sa posielajú dotazy pomocou príkazu `curl`
4. po odoslaní dotazu sa zobrazia prijaté aj odoslané pakety vo `Wireshark` a v `ipk-sniffer` a následne bola kontrolovaná zhodnosť paketov oboch výstupov

4 Implementované rozšírenie

Program podporuje aj pakety s ip protokolom 6 (IPv6). Keďže sa v zadaní o tom nepísalo, zaradil som to do sekcie rozšírení.

4.1 Implementácia

Pre podporu IPv6 bolo nutné pridať knižnicu `netinet/ip6.h`. Do pôvodnej implementácie bola pridaná podpora IPv6 jednoduchým zavedením premennej `ipv6` typu bool, ktorá je implicitne nastavená na `false`. Do callback funkcie `processPacket()` bola na začiatok pridaná jednoduchá podmienka, ktorá v prípade IPv6 paketu nastaví premennú `ipv6` na `true`.

```
if (ntohs(ethernet_header->ether_type) == ETHERTYPE_IPV6) {
          ipv6 = true;
     }
```

Od chvíle ako sa jedná o IPv6 paket, sa vykonávajú jednotlivé funkcie tak ako pri IPv4. Zistí sa protokol (TCP/UDP), vytvorí sa timestamp, ak je to možné, preložia sa IP adresy na doménové mená a vypisuje sa hlavička spolu s obsahom paketu.

Pri prekladaní IPv6 na doménové meno bola opäť využitá funkcia `gethostbyaddr()`, tentokrát ale bolo nutné použiť štruktúry `in6_addr` a `sockaddr_in6`.

Ďalším rozdielom bola dĺžka IP hlavičky v pakete. Pri IPv4 to bolo 20B, ale pri IPv6 to je až 40B.

4.2 Testovanie

Testovanie IPv6 podpory bolo komplikované, pretože nemám možnosť doma používať IPv6 a kvôli momentálnej situácii sa s tým nedalo nič spraviť. Jediný spôsob akým bola táto funkcionalita otestovaná

1. otvorenie Wireshark v prvom termináli pomocou príkazu `sudo wireshark` a pripojenie sa na rozhranie `lo`
2. spustenie projektu v novom termináli pomocou príkazu
   `sudo ./ipk-sniffer -i lo`
3. otvorenie tretieho terminálu, cez ktorý sa posielal dotaz napríklad v tvare [5]
   `curl -g -6 "http://[::1]:8080/"`
4. po odoslaní dotazu sa zobrazia prijaté aj odoslané pakety vo `Wireshark` a v `ipk-sniffer` a následne bola kontrolovaná zhodnosť paketov oboch výstupov

Literatúra

[1] What is a packet? https://computer.howstuffworks.com/question5251.htm

[2] What Does a Packet Look Like? http://web.deu.edu.tr/doc/oreily/networking/firewall/ch06_03.htm

[3] Programming with pcap https://www.tcpdump.org/pcap.html

[4] Packet sniffer file from Tim Carstens https://www.tcpdump.org/sniffex.c

[5] How can I use curl with ::1 for ipv6 based loopback? First answer https://superuser.com/questions/885753/how-can-i-use-curl-with-1-for-ipv6-based-loopback