VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

Počítačové komunikace a sítě

Dokumentace projektu

Varianta ZETA: Sniffer paketů

Daniel Kamenický (xkamen21)
Brno, 29. 4. 2020

Obsah

Úvod

Program demonstruje analyzátor paketů (packet sniffer). Po spuštění začne přijímat pakety na námi předaném rozhraní a vypisuje jejich dekódované data.

Návrh aplikace

Celý návrh probíhal v programovacím jazyce C.

Parametry

První věcí bylo ošetření vstupních parametrů a vymyšlení uchování dat. Zde se použila struktura **typedef struct Data**, do které se uchovala data o výskytu parametrů a také se do struktury uchovala data, která předával parametr.

Rozhraní

Další problematikou bylo otevření vstupního rozhraní pro poslech. Zde jsme využili knihovnu **pcap.h**, která byla doporučena v zadání. Bylo zapotřebí pochopit jak daná knihovna funguje a jak ji využít.

Získání dat

Poslední problematikou bylo získání dat z paketu a jejich vypsání. Program zpracovává pouze pakety protokolu TCP a UDP. Zde jsme využili dvě funkce z knihovny **pcap.h**. Návrh obsahuje dvě různé funkce pro UDP nebo TCP paket, které se liší jen v získávání dat. Poté je proces v obou případech stejný.

Implementace

Parametry

```
typedef struct Data{
    bool param_i;
    bool param_p;
    bool param_t;
    bool param_u;
    bool param_n;

    char i_data[50];
    int p_data;
    int n_data;
}Data;
```

Všechna data jsou uložena do zde zmíněné struktury. O výskytu jednotlivého parametru se starají proměnné typu **bool**. Při spuštění programu jsou všechny proměnné inicializované na hodnotu **False**. Při získávání jednotlivých parametrů použitím for cyklu se hodnota mění na **True**, při výskytu parametru. Tento systém používáme i pro kontrolu duplicity parametrů.

Pro ukladání dat se v struktuře **Data** nachází další tři proměnné. Jelikož bylo předem známo co za data budem dostávat, uzpůsobili jsme tomu i datové typy.

Ve funkci main je pak struktura inicializována a naplněna pomocí funkce:

```
void get_args( Data *data, int argc, char const *argv[ ] );
```

Rozhraní

Celá sekce týkající se rozhraní spočíva na knihovně **pcap.h**.

Zde byl důležitý parametr -i který předává rozhraní na kterém chceme poslouchat. Jestliže parametr chybí, program vypíše všechna dostupná rozhraní. Dostupná rozhraní nám pomohla najít funkce **pcap_findalldevs**. Rozhraní se uloží do seznamu pcap_if_t *interfaces, z kterého se dále vypíší na standardní výstup.

Po získání jména rozhraní jej program otevře pro poslouchámí pomocí funkce **pcap_open_live**.

Pro nastavení filtru byly využity funkce **pcap_compile**, která dany filtr sestaví a funkce **pcap_setfilter**, která sestavený filtr aplikuje.

V poslední řadě pomocí funkce **pcap_loop** zachytáváme dané pakety a volá se **my_callback** funkce rozebrána v sekci Získání dat. Jednim z parametrů funkce **pcap_loop** je celé číslo které nám udává počet paketů, které bude funkce číst. Po ukonční poslouchaní se rozhraní uzavře pomocí funkce **pcap_close**.

Získání dat

Získání a vypsání dat pacektů provádí funkce **my_callback** a funkce ji volané. Je předávána jako jeden z parametrů funkce **pcap_loop**. Jednim z parametrů funkce **my_callback** je const u_char *packet obsahující data která vypisujeme. Díky IP hlavičce paketu víme o jaký protokol se jedná (TCP nebo UDP). V závislosti na protokolu je volenou funkcí buď **print_tcp_packet** nebo **print_udp_packet**. Jediným rozdílem je délka hlavičky UDP a TCP paketu.

Po získání všech potřebných dat, jako je velikost hlavičky a velikost celkového paketu, kterou jsme dostali ve struktuře:

```
const struct pcap_pkthdr *header
```

Přesněji pod:

```
header -> caplen
```

předané ve funkci my_callback, začne tisk data na standardní výstup.

Před vypisováním dat hlavičky a těla paketu program vypíše aktualní čas pomocí funkce **PrintTime**. Dále vypíše zdrojovou IP adresu, zdrojový paket, cílovou IP adresu a cilový paket pomocí funkcí **PrintSourceIP** a **PrintDestIP**. Pokud se dá IP adresa reprezentovat jako jméno domény, je vypsáno jméno dané domény.

Pro výpis dat máme funkci **PrintData**. Jedná se o for cyklus, který projíždí celý paket a tiskne z něj data. Netisknutelná data jsou reprezentována “.”.

Testování

Prvotní testování bylo prováděno vypsáním rozhraní, vybráním jednoho z nich a čekáním na příchozí pakety. Jelikož časový rozestup příchozích paketů byl ruznorodý a někdy bylo zapotřebí čekat i více jak 3 minuty, bylo potřeba způsob zefektivnit.

Pro zlepšení byl využit druhý terminál s příikazem curl. Na jednom terminálu byl spuštěn sniffer a mezi tím na druhém terminálu byl vydán příkaz curl k vyslání paketů. Efektivita debugování mnohonásobně vzrostla.

Pro otestování správných dekódovaných dat z paketu byla využita aplikace Wireshark, se kterou byl projekt kontrolován v celém průběhu implementace. Příchozí paket byl zobrazen v aplikaci a porovnán s výsledkem programu. Jediná nevýhoda byla těžší přehlednost mezi Wiresharkem a ipk-sniffrem. Mnohdy nastal problém, zda li se jedná o stejný paket.

Rozšíření

Parametr --help

Rozšířením o parametr --help jsme docílili lepšímu seznámení uživatele s prostředím parametrů programu. Při vyžádání si o pomocný výpis uživatel obdrží všechny informace ohledně parametrů programu.

Zdroje

- [1] CARSTENS, Tim. Programming with pcap, 2002, [citace 29. 4. 2020] URL: https://www.tcpdump.org/pcap.html
- [2] SILVER MOON. C Packet Sniffer Code with libpcap and linux sockets, 26.4.2013 [citace 29. 4. 2020] URL:https://www.binarytides.com/packet-sniffer-code-in-c-using-linux-sockets-bsd-part-2/
- [3] Neznámý. pcap_findalldevs, 21. 1. 2016, [citace 29. 4. 2020] URL: http://embeddedguruji.blogspot.com/2014/01/pcapfindalldevs-example.html
- [4] URL: https://stackoverflow.com/