PACKET SNIFFER

IPK PROJEKT (ZETA) 2020

Autor: David Fridrich (xfridr08)

Popis projektu

Pro tento projekt bylo za úkol implementovat ‘packet sniffer’, tedy program, který bude odposlouchávat na různých zařízeních a/nebo portech a odchytávat packety, data, která procházejí skrz.

Projekt je rozdělen do dvou souborů. ipk-sniffer.cpp a ipk-sniffer.h, kde .h soubor je typický hlavičkový soubor, který obsahuje deklarace a popis funkcí a .cpp je samotný zdrojový soubor v jazyce Cpp. Je použit jeden objekt typu class, pro snadnější práci s argumenty přeposílané z příkazové řádky.

Popis samotného programu

Program při spuštění inicializuje objekt typu class pro následné používání. Pomocí funkce getopt_long(*1) jsou parsovány argumenty, v případě chybného argumentu rovnou program vypíše chybu (s odkazem na –help argument pro snazší orientaci).

Pro funkce pracující s počítačovou sítí jsou “includovány” následující knihovny:

```
#include<libnet.h>
#include<pcap.h>

#include <netinet/ip.h> //ipv4 header
#include <netinet/tcp.h> //tcp header
#include <netinet/udp.h> //udp header
#include <netinet/if_ether.h> //ethernet header
#include <netinet/ether.h>  //ethernet header declarations
#include <netinet/ip6.h>
```

První dvě ze zadání, následující pro snazší práci s packety samotnými.

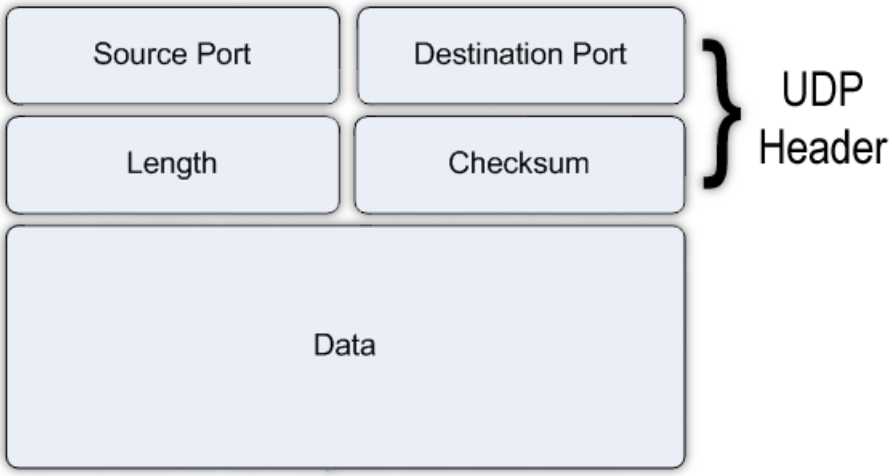
Program pokračuje jednoduchou kontrolou, jestli byl zadán argument –i. Pokud ne, využije funkci *pcap_findalldevs() (*2 funkce z knihovny pcap.h)*k tomu, aby vypsal všechna dostupná zařízení a úspěšně skončil.

Pokud –i argument zadán byl, zkontroluje zda parametry pro filtr (-p, -u, -t) byly zadány, pokud ano, nastaví podle toho filtr, který je následně použit při *pcap_compile() a pcap_setfilter()*. Jestliže se jakákoliv z těchto funkcí nepovede, vypíše na chybový výstup zprávu a ukončí se. Pokud vše projde bez problému, můžeme začít s odchytáváním packetů.

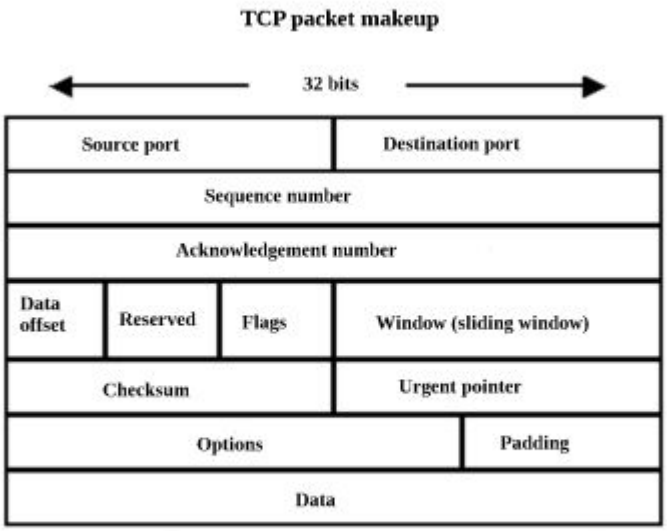
Zavolá se funkce *pcap_loop()* (*3),která mimo jiné jako argument dostane argument –n, který udává kolik packetů se má chytit předtím než se má skončit. Jestliže je toto číslo záporné, potom budeme chytat packety do nekonečna. Další velmi důležitý argument pcap_loop je tzv. Callback funkce, což je funkce, která je zavolána vždy, když je odchycen paket, tedy pro každá paket se tato funkce zavolá jednou. Toto je funkce, ve které se zpracovává paket samotný.

V callback funkci je jako první vyřešen čas, tedy přetypován na tisknutelný typ a následně vytisknut, jakožto úplně první část výstupu našeho paketu. Následně je paket rozdělen do několika částí, jelikož se z nich skládá. Rozlišujeme UDP a TCP paket.

UDP (User Datagram Protocol):



A TCP (Transmission Control Protocol):



Standardní velikost TCP packetu je 20 bytů (tedy 20-60 bytů je možná velikost)

Oba můžeme rozdělit na 2 části, header a data (neboli payload).

Mezi hlavní rozdíly mezi těmito dvěma protokoly patří UDP je tzv. “connectionless”, a TCP je naopak “connection-oriented”. UDP je rychlejší, a TCP je naopak bezpečnější.

V této části programu je vyhodnocováno, jaká packet byl tedy zachycen (jestli např. TCP) a podle toho je s ní dále zacházeno. Každý má své velikosti, (např. TCP) je tedy nejdříve rozdělen na Ethernet -> ip -> tcp -> payload. Jestliže známe velikosti každé části, je jednoduché se orientovat v celém paketu, protože si ho takto můžeme rozkouskovat na více menších částí. Z každého paketu je vypsána “source” tedy zdroj, odkud přišel a “destination” tedy cíl. Zároveň jsou vypsány porty, na kterých bylo posloucháno. Tímto jsou zpracovány informace z “headeru” packetu a nyní jsou odeslány do funkce, s příslušným ukazatelem a velikostí pro výpis samotného obsahu.

Funkce *void printOut(const u_char *buffer, int len) (*4)* zpracovává packet podle velikosti bytů zadané parametrem a pomocí funkce *void print_hex_ascii_line(const u_char *buffer, int len, int offset); (*převzata z *4, upravena funkce printf pro výpis hexa hodnot pro offset).

Tato funkce nejprve tiskne offset, tedy počet vypsaných bajtů. Následně pomocí for cyklu vypíše hexadecimální reprezentaci dat, která obsahuje packet. Následně jsou tato data vytisknuta pomocí ASCII, pokud je možné je vytisknout (pokud ne, jsou nahrazeny takové byty tečkou).

Odkazy pomocí (*) z předchozích stran:

(některé části kódu jsou detailněji popsané ve zdrojovém souboru a konkrétně jim přiřazené zdroje a inspirace)

*1 - https://linux.die.net/man/3/getopt_long

*2 - https://www.tcpdump.org/manpages/pcap.3pcap.html

*3 - všechny tyto pcap funkce a 'code snippets' (ve zdrojovém souboru označeny komentářem) byly inspirovány zde https://www.tcpdump.org/pcap.html

*4 - inspirováno ze souboru sniffex.c stáhnutelný zde: https://www.tcpdump.org/sniffex.c (část licence v ipk-sniffer.h)

Další zdroje:

https://www.tcpdump.org/linktypes.html

https://link.springer.com/chapter/10.1007/978-981-10-7901-6_97

Citace

Programming with pcapTCPDUMP/LIBPCAP public repository. *TCPDUMP/LIBPCAP public repository* [online]. Dostupné z: https://www.tcpdump.org/pcap.html

Steps on how to add the UDP header definition class coding using C++ .NET for the protocols header definition class in Windows network programming. *The Windows socket - Winsock Windows network programming tutorials using C#, C++/CLI and VB .NET with working code examples and program samples* [online]. Dostupné z: https://www.winsocketdotnetworkprogramming.com/clientserversocketnetworkcommunication8e_4.html

Network Traffic Analysis and Packet Sniffing Using UDP | SpringerLink. *Home - Springer* [online]. Copyright © 2019 Springer Nature Switzerland AG. Part of [cit. 03.05.2020]. Dostupné z: https://link.springer.com/chapter/10.1007/978-981-10-7901-6_97

Link-Layer Header Types | TCPDUMP/LIBPCAP public repository. *TCPDUMP/LIBPCAP public repository* [online]. Dostupné z: https://www.tcpdump.org/linktypes.html