Vysoké učenie technické v Brne
Fakulta informačných technológií

IPK-Počítačové komunikace a sítě

Projekt 2 – Varianta ZETA: Sniffer paketů

Adam Ševčík – xsevci64

Obsah:

1. Zadanie

Vytvorte komunikující aplikaci podle konkrétní vybrané specifikace obvykle za použití libpcap a/nebo síťové knihovny BSD sockets (pokud není ve variantě zadání uvedeno jinak).

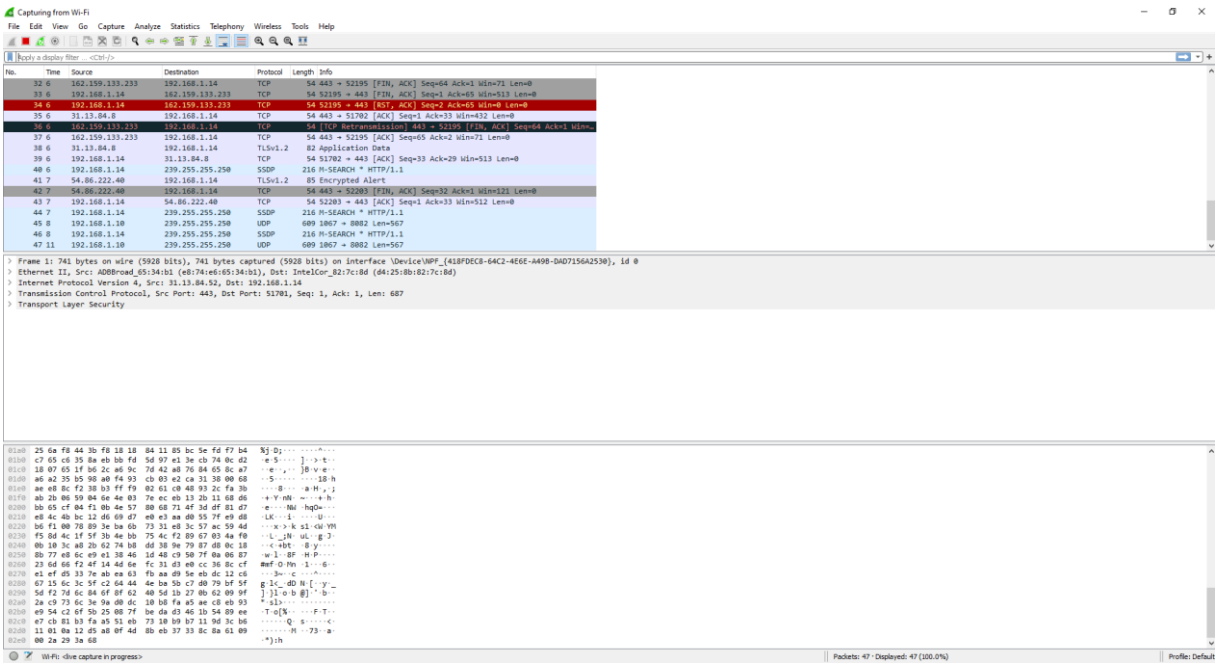
2. Naštudovanie problematiky

Pred začiatkom pracovania na samotnom projekte som si najskôr naštudoval potrebné informácie k problematike sniffovania packetov. Ako dobrý začiatok mi prišlo pozretie videa o tejto problematike (https://www.youtube.com/watch?v=V3C5Y2NXU68&t=269s) z ktorého som zhruba pochopil o čo ide. Ďalej pre lepšie chápanie som začal študovať stránku https://www.tcpdump.org/pcap.html?fbclid=IwAR1dvDpDM_vfgOBZxDy2YeT2J3t1TJLyZAB__VY44eezI7eBqzm3s1zM4Rw

Ktorá rozoberá základy programovania pomocou knižnice pcap. Po naštudovaní tejto stránky som pre úplne pochopenia prešiel zopár slidov z prednášok.

3. Inšpirácia

Po tom ako som zistil potrebné informácie mi prišlo vhodné pozrieť sa na nejakú vhodnú praktickú ukážku a tak som si stihol a nainštaloval program Wireshark. V ktorom som skúmal čo všetko taký packet sniffing obnáša a ako asi vyzerá výstup.



Ďalej som našiel na stránke github voľne dostupný kód https://gist.github.com/fffaraz/7f9971463558e9ea9545 ktorý riešil tento problém a ktorého časti som aj použil v mojej implementácií.

4. Implementácia

Po tomto teoretickom aj praktickom naštudovaní som sa pustil do samotného programovania.

a. Prijímanie parametrov

Ako prvé som sa zameral na prijímanie parametrov. Toto som napriek odporúčaniu implementoval bez použitia *getopt()* . Moja implementácia spočíva vo viacerých if/else if a to konkrétne pre každý parameter. Pre každý parameter exituje jedna premenná typu bool. Ak sa pri spustení pomocou príkazovej riadky nachádza na riadku príslušná premenná má hodnotu true. Ak sa nenachádza je to false. Ďalej sa po každom parametri skontroluje či nasleduje správna hodnota tohto parametru. Ak nie vypíše sa chybová hláška a program sa ukončí. Ak áno príslušná hodnota sa uloží do premennej na neskoršie použitie. V mojej implementácií som nehľadel na ostatné argumenty na príkazovom riadku. Ak sa nachádza pri spustení viac rovnakých parametrov vždy je braný do úvahy posledný parameter jedného typu teda napríklad príkaz *./ipk-sniffer -i eth0 -p 23 --tcp -n 2* je pre môj program to isté ako píkaz *./ipk-sniffer -i eth0 hello_world -p 23 goodbye_world --tcp -n 2* alebo príkaz *./ipk-sniffer -i eth0 -p 23 –n 45 --tcp -n 2* .

b. Vytvorenie filtru

Po prijatí a spracovaní parametrov som si vytvoril string ktorý som neskôr vložil do funkcie pcap_compile ako string filtru. Tento problém som si rozložil na tri časti. Jednu pre možnosť ak mám prijímať aj udp aj tcp packety, jednu pre možnosť ak len udp a poslednú pre možnosť ak len tcp packety. Problém som vyriešil tak že som si predpripravil string pred možnosti udp a tcp a ak sa mal filtrovať len jeden z nich vložil som ich do filtru. Ak sa mali filtrovať oba vložil som ich do stringu a spojil logickou spojkou ||. Ďalej ak bol jeden z parametrov parameter port tak som ho pridal na koniec stringu ako port number. Príklady filtrov: *udp || tcp port 8082 , udp port 23 , tcp ,...*

c. Vypísanie rozhraní

Ak sa medzi parametrami nenachádza parameter –i vypíšem všetky dostupné rozhrania a ich popis. Kód

```
for(device = alldevsp ; device != NULL ; device = device->next{

        printf("%s - %s\n" , device->name , device->description);
   }
```

d. „Otvorenie“ rozhrania

Ak je zadané správne rozhranie otvorí sa komunikácia pomocou funkcie pcap_open_live()

e. Nastavenie filtru a začiatok sniffovania

Podľa bodu b s pomocou dvoch funkcií z knižnice pcap *pcap_compile() a pcap_setfilter()* nastavím zvolený filter a pomocou funkcie pcap_loop() ktorá slúži na automaticke čítanie packetov privolám funkciu process_packet ktorá zaačne spracovanie packetu a to toľkokrát akú hodnotu má parameter –n.

f. Funkcia process_packet

Funkcia process_packet slúži na rozhodnutie o tom packet s akým protokolom sa má spracovať. Táto funkcia zároveň aj získa čas prijatia packetu na neskoršie vypísanie. Podľa toho aký protokol má práve spracovávaný packet sa zavolá príslušná funkcia.

g. Funkcie udp_packet_print a tcp_packet_print

Tieto dve funkcie maju rovnakú funkčnosť. Jediný rozdiel je ,že udp_packet_print vytvorí štruktúru udphdr teda tcp header:
*struct tcphdr *udph=(struct tcphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));*
tcp_packet_print vytvorí štruktúru tcphdr teda tcp header:
*struct tcphdr *tcph=(struct tcphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));*
Ďalej obe funkcie vytvoria štruktúru iphdr teda ip header :
*struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ethhdr));*
A vypočítajú dĺžku headeru pre neskoršie vypísanie dát
*int header_size = sizeof(struct ethhdr) + iphdrlen + tcph->doff*4;*
Nakoniec obe funkcie zavolajú funkciu print_ip_header a dvakrát funkciu data_print pre vypísanie headeru a dát packetu.

h. Funkcia print_ip_header

Funkcia slúži na vypísanie prvého riadku každého packetu. Najskôr získa source a desitnation ip ktoré nakopíruje do pomocnej premennej kvôli tomu , že pri používaní *inet_ntoa* sa mi uložila source ip adresa aj do destination ip adresy podľa toho aká je to verzia ip a potom vypíše čas získaný z predchádzajúcej funkcie ,source ip , source port , destination ip a destination port. Príklad
11:52:49.079012 pcvesely.fit.vutbr.cz : 4093 > 10.10.10.56 : 80

i. Funkcia data_print

Posledná funkcia je funkcia data_print slúži na vypísanie dát pri tejto funkcii som sa inšpiroval funkciou z githubu spomínaného v časti inšpirácia. Funkcia najskôr vypíše počet vypísaných bajtov v hexadecimálnom tvar napirklad 0x0010 potom sa vypíše 16 bajtov v hexa a ASCII kóde príklad:
0x0010: 05 a0 52 5b 40 00 36 06 5b db d9 43 16 8c 93 e5 ..R[@.6. [..C....
Ak je znak nestlačiteľný v časti ASCII sa namiesto neho vypíše bodka.

5. Testovanie

Moje testovanie prebiehalo počas celej tvorby projektu keď som po pridaní každej novej funkcie otestoval jej funkčnosť. Finálnu verziu som bohužiaľ nestihol otestovať tak ako by som chcel z dôvodu časovej náročnosti.