Lecture 6: Writing a sniffer with winpcap library

\* for winpcap manual go to

http://www.winpcap.org/docs/docs\_40\_2/html/main.html

0. Overall steps

obtain network interface list

open the device

set filtering rules

sniff

1. Obtaining the device list: what network adapters are available?

- use pcap\_findalldevs()

- this will return a linked list of pcap\_if{}

#include <stdio.h>

#include "pcap.h"

int main(){

pcap\_if\_t \*alldevs=NULL;

char errbuf[PCAP\_ERRBUF\_SIZE];

// find all network adapters

if (pcap\_findalldevs(&alldevs, errbuf)==-1){

printf("dev find failed\n");

return -1;

}

if (alldevs==NULL){

printf("no devs found\n");

return -1;

}

// print them

pcap\_if\_t \*d; int i;

for(d=alldevs,i=0; d!=NULL; d=d->next){

printf("%d-th dev: %s ", ++i, d->name);

if (d->description)

printf(" (%s)\n", d->description);

else

printf(" (No description available)\n");

}

return 0;

}

- pcap\_if ={

pcap\_if \*next; // pointer to next interface

char \*name; // name of this device

char \*description; // device description

pcap\_addr \* addresses; // ip address

flgas;

}

- You should already have win10pcap driver installed.

--If not, download win10pcap installation executable and run to install

the win10pcap device driver

- From eclass, download winpcap developer's packs(WpdPack\_4\_0\_2.zip) and uncompress.

-- Remember the location of winpcap library and header files.

- Now satisfy the naming rule: include "pcap.h" and link with "ws2\_32.lib" and "wpcap.lib"

-- add "pcap.h" (actually this is already done in above program)

#include "pcap.h"

-- add "ws2\_32.lib" and "wpcap.lib" as linking modules

(Project>project properties>Linker>input output>additional dependencies>edit)

ws2\_32.lib

wpcap.lib

(or add below at the top of your code:

#pragma comment (lib, "ws2\_32.lib")

#pragma comment (lib, "wpcap.lib")

)

-- the location of non-standard header file and lib file should be specified

- add the path of winpcap header files in the include file search paths.

(Project>project properties>VC++ Directories>Include Directories)

- add the path of winpcap library in the library search path.

(Project>project properties>VC++ Directories>library Directories)

- Now compile and run.

(Sometimes, the platform of the program (the one shown in the main interface screen) is different from the platform in properties configuration. They should match.)

(for some later Virtual Studio version add following before #include "pcap.h"

#define WIN32

#define WPCAP

#define HAVE\_REMOTE

#define \_CRT\_SECURE\_NO\_WARNINGS

Refer to: http://www.rhyous.com/2011/11/12/how-to-compile-winpcap-with-visual-studio-2010/)

(For macOS

1) include <pcap.h> instead of "pcap.h"

.........

#include <pcap.h>

........

2) compile with -lpcap

$gcc -o sniffer sniffer.c -lpcap

or if you have \_\_gxx\_personality\_v0 error

$g++ -o sniffer sniffer.c -lpcap

3) run with "sudo"

$sudo ./sniffer )

2. Opening an adapter: pcap\_open()

.............

int inum;

printf("enter the interface number: ");

scanf("%d", &inum);

for(d=alldevs, i=0; i<inum-1; d=d->next, i++); // jump to the inum-th dev

// open

pcap\_t \*fp;

if ((fp = pcap\_open\_live(d->name, // name of the device

65536, // capture size

1, // promiscuous mode

20, // read timeout

errbuf

))==NULL){

printf("pcap open failed\n");

pcap\_freealldevs(alldevs);

return -1;

}

printf("pcap open successful\n");

If reading from a pcap file, use

pcap\_t \* pcap\_open\_offline(const char \*fname, char \*errbuf)

struct pcap {

int fd;

int snapshot;

....

int bufsize;

u\_char \*buffer;

.......

};

3. Set filtering rules for the traffic: pcap\_compile(), pcat\_setfilter()

struct bpf\_program fcode;

if (pcap\_compile(fp, // pcap handle

&fcode, // compiled rule

(char \*)("host 165.246.38.152 and port 9924"), // filter rule

1, // optimize

NULL) < 0){

printf("pcap compile failed\n");

pcap\_freealldevs(alldevs);

return -1;

}

if (pcap\_setfilter(fp, &fcode) <0 ){

printf("pcap setfilter failed\n");

pcap\_freealldevs(alldevs);

return -1;

}

printf("filter setting successful\n");

4. Capturing the packet: pcap\_next\_ex()

// capture. you have to implement print\_raw\_packet, print\_ether\_header, etc.

pcap\_freealldevs(alldevs); // we don't need this anymore

struct pcap\_pkthdr \*header;

const unsigned char \*pkt\_data;

int res;

while((res=pcap\_next\_ex(fp, &header,&pkt\_data))>=0){// 1 if success

if (res==0) continue; // 0 if time-out

print\_raw\_packet(pkt\_data, header->caplen);

print\_ether\_header(pkt\_data);

print\_ip\_header(pkt\_data);

print\_tcp\_header(pkt\_data);

print\_data(pkt\_data);

}

header contains timestamp and actual captured packet length as below.

struct pcap\_pkhdr{ // defined in pcap.h

struct timeval ts; // time stamp

bpf\_u\_int32 caplen; // length of portion present

bpf\_u\_int32 len; // length of this packet

};

Timestamp can be read as follows.

struct timeval this\_ts=header->ts; // timestamp of this packet

double pkt\_time=this\_ts.tv\_sec + this\_ts.tv\_usec/1.0e6; // time value of this packet

char timestr[256];

sprintf(timestr, "%d.%06d", (int)this\_ts.tv\_sec, (int)this\_ts.tv\_usec); // disply sec and usec

printf("sec and usec:%s\n", timestr);

printf("packet timestamp:%f\n", pkt\_time); // display timestamp

5. homework

1) Make a packet sniffer using winpcap library. Let it dump the raw byte stream of the packet.

Compare it with the output of windump.

2) Improve your sniffer such that it also prints all the fields in ethernet header, ip header, and tcp header. Use ntohs for "short" data type and ntohl for "int" data type in order to display them correclty. For the data part, just show them in hexadecimal numbers.

dest MAC: ......

src MAC: .......

protocol type: ........

IP version: ...

IP header length: .....

......................

Use following structures.

struct ether\_addr {

unsigned char ether\_addr\_octet[6];

};

struct ether\_header {

struct ether\_addr ether\_dhost;

struct ether\_addr ether\_shost;

unsigned short ether\_type; // 0x0800 for IP

};

struct ip\_hdr{

unsigned char ip\_header\_len:4;

unsigned char ip\_version:4;

unsigned char ip\_tos;

unsigned short ip\_total\_length;

unsigned short ip\_id;

unsigned char ip\_frag\_offset:5;

unsigned char ip\_more\_fragment:1;

unsigned char ip\_dont\_fragment:1;

unsigned char ip\_reserved\_zero:1;

unsigned char ip\_frag\_offset1;

unsigned char ip\_ttl;

unsigned char ip\_protocol;

unsigned short ip\_checksum;

unsigned int ip\_srcaddr;

unsigned int ip\_destaddr;

};

struct tcp\_hdr{

unsigned short source\_port;

unsigned short dest\_port;

unsigned int sequence;

unsigned int acknowledge;

unsigned char ns:1;

unsigned char reserved\_part1:3;

unsigned char data\_offset:4;

unsigned char fin:1;

unsigned char syn:1;

unsigned char rst:1;

unsigned char psh:1;

unsigned char ack:1;

unsigned char urg:1;

unsigned char ecn:1;

unsigned char cwr:1;

unsigned short window;

unsigned short checksum;

unsigned short urgent\_pointer;

};