HEADER FILES

-------------------------------------------------------------------------

#include<time.h>

#include<stdio.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<string.h>

#include<sys/select.h>

#include<pthread.h>

#include<signal.h>

#include<stdlib.h>

#include<fcntl.h>

#include<sys/shm.h>

#include<unistd.h>

#include<sys/un.h>

#include<netinet/ip.h>

#include<arpa/inet.h>

#include<pcap.h>

#include<errno.h>

#include<netinet/if\_ether.h>

#include<net/ethernet.h>

#include<netinet/ether.h>

#include<netinet/udp.h>

#include<sys/ipc.h>

#include<sys/msg.h>

/\*\*

This function is used to send file descriptor over Unix domain socket

You can use this function with file descriptor return

by any one of below functions

1 . socketpair();

2 . socket(AF\_UNIX,...);

3 . socket(AF\_LOCAL,...);

@param socket file\_descriptor\_of\_sender

@param fd\_to\_send

\*/

SHARE MEMORY

-----------------------------------------------------------------------------------------------------------------------------------

int state=1;

key\_t h=ftok(".",state++); // value of state should on every program where this share memory is used

int shmid=shmget(h,sizeof(int),IPC\_CREAT|0666);

share\_memory=shmat(shmid,(const void\*)0,0);

SEMAPHORE

-------------------------------------------------------------------------

void sem\_wait(int semid)

{

struct sembuf sb;

sb.sem\_num=0;

sb.sem\_op=-1;

sb.sem\_flg=0;

if((semop(semid,&sb,1))==-1)

{

perror("\nFailed to acquire semaphore.");

exit(0);

}

}

void sem\_try\_wait(int semid)

{

struct sembuf sb;

sb.sem\_num=0;

sb.sem\_op=-1;

sb.sem\_flg=IPC\_NOWAIT;;

return semop(semid,&sb,1);

}

void sem\_signal(int semid)

{

struct sembuf sb;

sb.sem\_num=0;

sb.sem\_op=1;

sb.sem\_flg=0;

if((semop(semid,&sb,1))==-1)

{

perror("\nFailed to release semaphore.");

exit(0);

}

}

int state=1;

key\_t h=ftok(".",state++); // value of state should on every program where this semaphore is used

int sem\_id;

if((sem\_id=semget(h,1,0666|IPC\_CREAT))==-1)

{

printf("error in creation semaphore\n");

exit(0);

}

int semaphore\_value=1;

if((semctl(sem\_id,0,SETVAL,semaphore\_value))==-1)

{

printf("error to set value\n");

}

(OR)

#define sname "/mysem"

sem\_t \*sem = sem\_open(sname, O\_CREAT, 0644, 0);

sem\_t \*sem = sem\_open(sname,1);

sem\_wait(sem);

sem\_post(sem);

-------------------------------------------------------------------------

MSG QUEUE

-------------------------------------------------------------------------

struct mymsg

{

long type;

char msg[20];

};

struct mymsg msg1;

key\_t key;

int mqpid;

int ret;

int len;

system("touch f1.txt");

if((key=ftok("f1.txt",'B')) == -1)

{

perror("key");

exit(1);

}

if((mqpid=msgget(key,0644|IPC\_CREAT))==-1)

{

perror("Key");

exit(1);

}

if(msgsnd( mqpid ,&msg1 ,len+1 , 0) == -1)

{

perror("msgsnd");

exit(1);

}

memset(msg1.msg,'\0',sizeof(msg1.msg));

if(msgrcv( mqpid , &msg1 , sizeof(msg1.msg),1 ,0) == -1)

{

perror("msgrcv");

exit(1);

}

----------------------------------------------------------------------------------------------------------------------------------

FIFO

----------------------------------------------------------------------------------------------------------------------------------

char name[50];

if(mkfifo(name,0666)==-1)

{

perror("mkfifo()1");

exit(1);

}

if((wfd=open("./wellknownfifo",O\_WRONLY))==-1)

{

perror("open()");

exit(1);

}

write(wfd,buffer,sizeof(buffer));

char buffer[50];

if(mkfifo("./wellknownfifo",0666)==-1)

{

perror("mkfifo()");

exit(1);

}

if((rfd=open("./wellknownfifo",O\_RDONLY))==-1)

{

perror("open()");

exit(1);

}

read(rfd,buffer,50);

----------------------------------------------------------------------------------------------------------------------------------

SELECT

----------------------------------------------------------------------------------------------------------------------------------

fd\_set readset;

FD\_ZERO(&readset);

int max=-1;

for(i=0;i<no\_of\_file\_descriptors;i++)

{

FD\_SET(fd[i], &readset);

if(fd[i]>max)

max=fd[i];

}

struct timeval t;

t.tv\_sec=3;

t.tv\_usec=100;

int rv = select(max + 1, &readset, NULL, NULL, &t);

if (rv == -1)

{

perror("select");

}

else if (rv == 0)

{

printf("Timeout occurred!\n");

}

else

{

int i;

// check for events

for(i=0;i<no\_of\_file\_descriptors;i++)

if (FD\_ISSET(fd[i], &readset))

{

}

}

pthread

----------------------------------------------------------------------------------------------------------------------------------

void do\_thread\_service(void \*arg)

{

int \*args= (int\*)arg ;

}

pthread\_t t\_service;

if(pthread\_create(&t\_service,NULL,(void\*)&do\_thread\_service ,(void\*)args)!=0)

perror("\npthread\_create ");

CONNECTION ORIENTED SERVER ( usage -: "./a.out port\_no")

---------------------------------------------------------------------------------------------------------------------------------

if(argc!=2)

printf("\n usage ./a.out port\_no");

int sfd;

struct sockaddr\_in serv\_addr,cli\_addr;

socklen\_t cli\_len;

int port\_no=atoi(argv[1]);

if((sfd = socket(AF\_INET,SOCK\_STREAM,0))==-1)

perror("\n socket ");

else printf("\n socket created successfully");

bzero(&serv\_addr,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(port\_no);

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

int opt=1;

setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt));

if(bind(sfd,(struct sockaddr \*) &serv\_addr,sizeof(serv\_addr))==-1)

perror("\n bind : ");

else printf("\n bind successful ");

listen(sfd,10);

cli\_len=sizeof(cli\_addr);

int nsfd;

if((nsfd = accept(sfd , (struct sockaddr \*)&cli\_addr , &cli\_len))==-1)

perror("\n accept ");

else printf("\n accept successful");

//break after exec in child

CONNECTION ORIENTED CLIENT ( usage -: "./a.out port\_no")

---------------------------------------------------------------------------------------------------------------------------------

if(argc!=2)

printf("\n usage ./a.out port\_no");

int sfd;

struct sockaddr\_in serv\_addr;

int port\_no=atoi(argv[1]);

bzero(&serv\_addr,sizeof(serv\_addr));

if((sfd = socket(AF\_INET , SOCK\_STREAM , 0))==-1)

perror("\n socket");

else printf("\n socket created successfully\n");

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(port\_no);

//serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

inet\_pton(AF\_INET,"127.0.0.1", &serv\_addr.sin\_addr);

if(connect(sfd , (struct sockaddr \*)&serv\_addr , sizeof(serv\_addr))==-1)

perror("\n connect : ");

else printf("\nconnect succesful");

CONNECTION LESS SERVER ( usage -: "./a.out port\_no")

---------------------------------------------------------------------------------------------------------------------------------

if(argc!=2)

printf("\n usage ./a.out port\_no");

int sfd;

struct sockaddr\_in serv\_addr,cli\_addr;

socklen\_t cli\_len;

int port\_no=atoi(argv[1]);

if((sfd = socket(AF\_INET,SOCK\_DGRAM,0))==-1)

perror("\n socket ");

else printf("\n socket created successfully");

bzero(&serv\_addr,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(port\_no);

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if(bind(sfd,(struct sockaddr \*) &serv\_addr,sizeof(serv\_addr))==-1)

perror("\n bind : ");

else printf("\n bind successful ");

cli\_len = sizeof(cli\_addr);

fgets( buffer , 256 , stdin );

sendto(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &cli\_addr , cli\_len);

recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &cli\_addr , & cli\_len );

CONNECTION LESS CLIENT ( usage -: "./a.out port\_no")

---------------------------------------------------------------------------------------------------------------------------------

if(argc!=2)

printf("\n usage ./a.out port\_no");

int sfd;

struct sockaddr\_in serv\_addr;

int port\_no=atoi(argv[1]);

char buffer[256];

bzero(&serv\_addr,sizeof(serv\_addr));

if((sfd = socket(AF\_INET , SOCK\_DGRAM , 0))==-1)

perror("\n socket");

else printf("\n socket created successfully\n");

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(port\_no);

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

socklen\_t serv\_len = sizeof(serv\_addr);

fgets( buffer , 256 , stdin );

sendto(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &serv\_addr , serv\_len);

recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &serv\_addr , & serv\_len );

UNIX SOCKET CONNECTION ORIENTED SERVER ( usage -: "./a.out")

---------------------------------------------------------------------------------------------------------------------------------

#define ADDRESS "mysocket"

int usfd;

struct sockaddr\_un userv\_addr,ucli\_addr;

int userv\_len,ucli\_len;

usfd = socket(AF\_UNIX , SOCK\_STREAM , 0);

perror("socket");

bzero(&userv\_addr,sizeof(userv\_addr));

userv\_addr.sun\_family = AF\_UNIX;

strcpy(userv\_addr.sun\_path, ADDRESS);

unlink(ADDRESS);

userv\_len = sizeof(userv\_addr);

if(bind(usfd, (struct sockaddr \*)&userv\_addr, userv\_len)==-1)

perror("server: bind");

listen(usfd, 5);

ucli\_len=sizeof(ucli\_addr);

int nusfd;

nusfd=accept(usfd, (struct sockaddr \*)&ucli\_addr, &ucli\_len);

UNIX SOCKET CONNECTION ORIENTED CLIENT ( usage -: "./a.out")

---------------------------------------------------------------------------------------------------------------------------------

#define ADDRESS "mysocket"

int usfd;

struct sockaddr\_un userv\_addr;

int userv\_len,ucli\_len;

usfd = socket(AF\_UNIX, SOCK\_STREAM, 0);

if(usfd==-1)

perror("\nsocket ");

bzero(&userv\_addr,sizeof(userv\_addr));

userv\_addr.sun\_family = AF\_UNIX;

strcpy(userv\_addr.sun\_path, ADDRESS);

userv\_len = sizeof(userv\_addr);

if(connect(usfd,(struct sockaddr \*)&userv\_addr,userv\_len)==-1)

perror("\n connect ");

else printf("\nconnect succesful");

SEND\_FD AND RECV\_FD

---------------------------------------------------------------------------------------------------------------------

int send\_fd(int socket, int fd\_to\_send)

{

struct msghdr socket\_message;

struct iovec io\_vector[1];

struct cmsghdr \*control\_message = NULL;

char message\_buffer[1];

/\* storage space needed for an ancillary element with a paylod of length is CMSG\_SPACE(sizeof(length)) \*/

char ancillary\_element\_buffer[CMSG\_SPACE(sizeof(int))];

int available\_ancillary\_element\_buffer\_space;

/\* at least one vector of one byte must be sent \*/

message\_buffer[0] = 'F';

io\_vector[0].iov\_base = message\_buffer;

io\_vector[0].iov\_len = 1;

/\* initialize socket message \*/

memset(&socket\_message, 0, sizeof(struct msghdr));

socket\_message.msg\_iov = io\_vector;

socket\_message.msg\_iovlen = 1;

/\* provide space for the ancillary data \*/

available\_ancillary\_element\_buffer\_space = CMSG\_SPACE(sizeof(int));

memset(ancillary\_element\_buffer, 0, available\_ancillary\_element\_buffer\_space);

socket\_message.msg\_control = ancillary\_element\_buffer;

socket\_message.msg\_controllen = available\_ancillary\_element\_buffer\_space;

/\* initialize a single ancillary data element for fd passing \*/

control\_message = CMSG\_FIRSTHDR(&socket\_message);

control\_message->cmsg\_level = SOL\_SOCKET;

control\_message->cmsg\_type = SCM\_RIGHTS;

control\_message->cmsg\_len = CMSG\_LEN(sizeof(int));

\*((int \*) CMSG\_DATA(control\_message)) = fd\_to\_send;

return sendmsg(socket, &socket\_message, 0);

}

int recv\_fd(int socket)

{

int sent\_fd, available\_ancillary\_element\_buffer\_space;

struct msghdr socket\_message;

struct iovec io\_vector[1];

struct cmsghdr \*control\_message = NULL;

char message\_buffer[1];

char ancillary\_element\_buffer[CMSG\_SPACE(sizeof(int))];

/\* start clean \*/

memset(&socket\_message, 0, sizeof(struct msghdr));

memset(ancillary\_element\_buffer, 0, CMSG\_SPACE(sizeof(int)));

/\* setup a place to fill in message contents \*/

io\_vector[0].iov\_base = message\_buffer;

io\_vector[0].iov\_len = 1;

socket\_message.msg\_iov = io\_vector;

socket\_message.msg\_iovlen = 1;

/\* provide space for the ancillary data \*/

socket\_message.msg\_control = ancillary\_element\_buffer;

socket\_message.msg\_controllen = CMSG\_SPACE(sizeof(int));

if(recvmsg(socket, &socket\_message, MSG\_CMSG\_CLOEXEC) < 0)

return -1;

if(message\_buffer[0] != 'F')

{

/\* this did not originate from the above function \*/

return -1;

}

if((socket\_message.msg\_flags & MSG\_CTRUNC) == MSG\_CTRUNC)

{

/\* we did not provide enough space for the ancillary element array \*/

return -1;

}

/\* iterate ancillary elements \*/

for(control\_message = CMSG\_FIRSTHDR(&socket\_message);

control\_message != NULL;

control\_message = CMSG\_NXTHDR(&socket\_message, control\_message))

{

if( (control\_message->cmsg\_level == SOL\_SOCKET) &&

(control\_message->cmsg\_type == SCM\_RIGHTS) )

{

sent\_fd = \*((int \*) CMSG\_DATA(control\_message));

return sent\_fd;

}

}

return -1;

}

UNIX SOCKET CONNECTION LESS SERVER ( usage -: "./a.out")

---------------------------------------------------------------------------------------------------------------------------------

#define ADDRESS "mysocket"

int usfd;

struct sockaddr\_un userv\_addr,ucli\_addr;

int userv\_len,ucli\_len;

usfd = socket(AF\_UNIX , SOCK\_DGRAM , 0);

perror("socket");

bzero(&userv\_addr,sizeof(userv\_addr));

userv\_addr.sun\_family = AF\_UNIX;

strcpy(userv\_addr.sun\_path, ADDRESS);

unlink(ADDRESS);

userv\_len = sizeof(userv\_addr);

if(bind(usfd, (struct sockaddr \*)&userv\_addr, userv\_len)==-1)

perror("server: bind");

fgets( buffer , 256 , stdin );

sendto(usfd , buffer , 256 , 0 , ( struct sockaddr \* ) &ucli\_addr , ucli\_len);

recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &ucli\_addr , & uscli\_len );

UNIX SOCKET CONNECTION LESS CLIENT ( usage -: "./a.out")

---------------------------------------------------------------------------------------------------------------------------------

#define ADDRESS "mysocket"

int usfd;

struct sockaddr\_un userv\_addr;

int userv\_len,ucli\_len;

usfd = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if(usfd==-1)

perror("\nsocket ");

bzero(&userv\_addr,sizeof(userv\_addr));

userv\_addr.sun\_family = AF\_UNIX;

strcpy(userv\_addr.sun\_path, ADDRESS);

userv\_len = sizeof(userv\_addr);

fgets( buffer , 256 , stdin );

sendto(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &userv\_addr , userv\_len);

recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr \* ) &userv\_addr , & userv\_len );

SOCKET PAIR ( usage -: "./a.out")

---------------------------------------------------------------------------------------------------------------------------------

int usfd[2];

if(socketpair(AF\_UNIX,SOCK\_STREAM,0,usfd)==-1)

perror("socketpair ");

int c=fork();

if(c==-1)

perror("\nfork ");

else if(c>0)

{

close(usfd[1]);

}

else if(c==0)

{

close(usfd[0]);

dup2(usfd[1],0);

execvp(file\_name,args);

}

RAW SOCKETS

---------------------------------------------------------------------------------------------------------------------------------

void print\_ipheader(struct iphdr\* ip)

{

cout<<"------------------------\n";

cout<<"Printing IP header....\n";

cout<<"IP version:"<<(unsigned int)ip->version<<endl;

cout<<"IP header length:"<<(unsigned int)ip->ihl<<endl;

cout<<"Type of service:"<<(unsigned int)ip->tos<<endl;

cout<<"Total ip packet length:"<<ntohs(ip->tot\_len)<<endl;

cout<<"Packet id:"<<ntohs(ip->id)<<endl;

cout<<"Time to leave :"<<(unsigned int)ip->ttl<<endl;

cout<<"Protocol:"<<(unsigned int)ip->protocol<<endl;

cout<<"Check:"<<ip->check<<endl;

cout<<"Source ip:"<<inet\_ntoa(\*(in\_addr\*)&ip->saddr)<<endl;

//printf("%pI4\n",&ip->saddr );

cout<<"Destination ip:"<<inet\_ntoa(\*(in\_addr\*)&ip->daddr)<<endl;

cout<<"End of IP header\n";

cout<<"------------------------\n";

}

RAW SOCKET SERVER

-----------------------------------------------------------------------------------------------------

if(argc<2)cout<<"Enter protocal in arguments";

int rsfd=socket(AF\_INET,SOCK\_RAW,atoi(argv[1]));

perror("socket");

int optval=1;

setsockopt(rsfd, IPPROTO\_IP, SO\_BROADCAST, &optval, sizeof(int));//IP\_HDRINCL

cout<<"opt"<<endl;

struct sockaddr\_in client;

client.sin\_family=AF\_INET;

client.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

char buff[]="hello";

client.sin\_addr.s\_addr=INADDR\_ANY;

unsigned int client\_len=sizeof(client);

cout<<"sending"<<endl;

sendto(rsfd,buff,strlen(buff)+1,0,(struct sockaddr\*)&client,sizeof(client));

perror("send");

RAW SOCKET CLIENT

---------------------------------------------------------------------------------------------------------------------------------

if(argc<2)cout<<"Enter protocol in arguments\n";

int rsfd=socket(AF\_INET,SOCK\_RAW,atoi(argv[1]));

if(rsfd==-1)custom\_perror("socket")

char buf[BUF\_LEN];

struct sockaddr\_in client;

socklen\_t clilen=sizeof(client);

cout<<"receive"<<endl;

recvfrom(rsfd,buf,BUF\_LEN,0,(sockaddr\*)&client,(socklen\_t\*)clilen);

perror("recv");

struct iphdr \*ip;

ip=(struct iphdr\*)buf;

cout<<(buf+(ip->ihl)\*4)<<endl;

GETPEERNAME (usage: only after accept; only on nsfd)

---------------------------------------------------------------------------------------------------------------------------------

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

{

int s;

struct sockaddr\_in peer;

int peer\_len;

peer\_len = sizeof(peer);

if (getpeername(s, &peer, &peer\_len) == -1) {

perror("getpeername() failed");

return -1;

}

/\* Print it. \*/

printf("Peer's IP address is: %s\n", inet\_ntoa(peer.sin\_addr));

printf("Peer's port is: %d\n", (int) ntohs(peer.sin\_port));

}

PASSING ARGUMENTS THROUGH EXEC

-------------------------------------------------------------------------------------------------------------------------------

string msg;

char \*\*arg=new char\*[2];

arg[0]=strdup(msg.c\_str());

arg[1]=NULL;

int c=fork();

if(c>0);

else if(c==0)

{

if(execvp("./s",arg)==-1)

cout<<eroor"<<endl;

exit(1);

}

//retrieving in child

int main(int argc, char const \*argv[])

{

string info=argv[argc];

}

MKFIFO

------------------------------------------------------------------------------------------------------------------------------

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <string.h>

int fd;

mkfifo("fifo1.fifo",0666);

fd=open("./fifo1.fifo",O\_RDONLY);

POLL

--------------------------------------------------------------------------------------------------------------------------------

int size;

struct pollfd fds[size];

fds[i]=open(" ", 0666);

fds[i].events=POLLIN;

int ret=poll(fds, size, timeout);

if(fds[i].revents & POLLIN)

{

}

LIBPCAP

-------------------------------------------------------------------------------------------------------------------------------

#include<pcap.h>

#include<stdio.h>

#include<stdlib.h> // for exit()

#include<string.h> //for memset

#include<sys/socket.h>

#include<arpa/inet.h> // for inet\_ntoa()

#include<net/ethernet.h>

#include<netinet/ip\_icmp.h> //Provides declarations for icmp header

#include<netinet/udp.h> //Provides declarations for udp header

#include<netinet/tcp.h> //Provides declarations for tcp header

#include<netinet/ip.h> //Provides declarations for ip header

void process\_packet(u\_char \*, const struct pcap\_pkthdr \*, const u\_char \*);

void process\_ip\_packet(const u\_char \* , int);

void print\_ip\_packet(const u\_char \* , int);

void print\_tcp\_packet(const u\_char \* , int );

void print\_udp\_packet(const u\_char \* , int);

void print\_icmp\_packet(const u\_char \* , int );

void PrintData (const u\_char \* , int);

FILE \*logfile; //to store the output

struct sockaddr\_in source,dest;

int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;

int main()

{

pcap\_if\_t \*alldevsp , \*device;

pcap\_t \*handle; //Handle of the device that shall be sniffed

char errbuf[100] , \*devname , devs[100][100];

int count = 1 , n;

// get the list of available devices

printf("Finding available devices ... ");

if( pcap\_findalldevs( &alldevsp , errbuf) )

{

printf("Error finding devices : %s" , errbuf);

exit(1);

}

printf("Done");

//Print the available devices

printf("\nAvailable Devices are :\n");

for(device = alldevsp ; device != NULL ; device = device->next)

{

printf("%d. %s - %s\n" , count , device->name , device->description);

if(device->name != NULL)

{

strcpy(devs[count] , device->name);

}

count++;

}

//Ask user which device to sniff

printf("Enter the number of the device you want to sniff : ");

scanf("%d" , &n);

devname = devs[n];

//Open the device for sniffing

printf("Opening device %s for sniffing ... " , devname);

handle = pcap\_open\_live(devname , 65535 , 1 , 0 , errbuf);

if (handle == NULL)

{

fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);

exit(1);

}

printf("Done\n");

logfile=fopen("log.txt","w");

if(logfile==NULL)

{

printf("Unable to create file.");

}

//Put the device in sniff loop

pcap\_loop(handle , 0 , process\_packet , NULL);

return 0;

}

void process\_packet(u\_char \*args, const struct pcap\_pkthdr \*header, const u\_char \*buffer)

{

int size = header->len;

//Get the IP Header part of this packet , excluding the ethernet header

struct iphdr \*iph = (struct iphdr\*)(buffer + sizeof(struct ethhdr));

++total;

switch (iph->protocol) //Check the Protocol and do accordingly...

{

case 1: //ICMP Protocol

++icmp;

print\_icmp\_packet( buffer , size);

break;

case 2: //IGMP Protocol

++igmp;

break;

case 6: //TCP Protocol

++tcp;

print\_tcp\_packet(buffer , size);

break;

case 17: //UDP Protocol

++udp;

print\_udp\_packet(buffer , size);

break;

default: //Some Other Protocol like ARP etc.

++others;

break;

}

printf("TCP : %d UDP : %d ICMP : %d IGMP : %d Others : %d Total : %d\r", tcp , udp , icmp , igmp , others , total);

}

void print\_ethernet\_header(const u\_char \*Buffer, int Size)

{

struct ethhdr \*eth = (struct ethhdr \*)Buffer;

fprintf(logfile , "\n");

fprintf(logfile , "Ethernet Header\n");

fprintf(logfile , " |-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->h\_dest[0] , eth->h\_dest[1] , eth->h\_dest[2] , eth->h\_dest[3] , eth->h\_dest[4] , eth->h\_dest[5] );

fprintf(logfile , " |-Source Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->h\_source[0] , eth->h\_source[1] , eth->h\_source[2] , eth->h\_source[3] , eth->h\_source[4] , eth->h\_source[5] );

fprintf(logfile , " |-Protocol : %u \n",(unsigned short)eth->h\_proto);

}

void print\_ip\_header(const u\_char \* Buffer, int Size)

{

print\_ethernet\_header(Buffer , Size);

unsigned short iphdrlen;

struct iphdr \*iph = (struct iphdr \*)(Buffer + sizeof(struct ethhdr) );

iphdrlen =iph->ihl\*4;

memset(&source, 0, sizeof(source));

source.sin\_addr.s\_addr = iph->saddr;

memset(&dest, 0, sizeof(dest));

dest.sin\_addr.s\_addr = iph->daddr;

fprintf(logfile , "\n");

fprintf(logfile , "IP Header\n");

fprintf(logfile , " |-IP Version : %d\n",(unsigned int)iph->version);

fprintf(logfile , " |-IP Header Length : %d DWORDS or %d Bytes\n",(unsigned int)iph->ihl,((unsigned int)(iph->ihl))\*4);

fprintf(logfile , " |-Type Of Service : %d\n",(unsigned int)iph->tos);

fprintf(logfile , " |-IP Total Length : %d Bytes(Size of Packet)\n",ntohs(iph->tot\_len));

fprintf(logfile , " |-Identification : %d\n",ntohs(iph->id));

fprintf(logfile , " |-TTL : %d\n",(unsigned int)iph->ttl);

fprintf(logfile , " |-Protocol : %d\n",(unsigned int)iph->protocol);

fprintf(logfile , " |-Checksum : %d\n",ntohs(iph->check));

fprintf(logfile , " |-Source IP : %s\n" , inet\_ntoa(source.sin\_addr) );

fprintf(logfile , " |-Destination IP : %s\n" , inet\_ntoa(dest.sin\_addr) );

}

void print\_tcp\_packet(const u\_char \* Buffer, int Size)

{

unsigned short iphdrlen;

struct iphdr \*iph = (struct iphdr \*)( Buffer + sizeof(struct ethhdr) );

iphdrlen = iph->ihl\*4;

struct tcphdr \*tcph=(struct tcphdr\*)(Buffer + iphdrlen + sizeof(struct ethhdr));

int header\_size = sizeof(struct ethhdr) + iphdrlen + tcph->doff\*4;

fprintf(logfile , "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TCP Packet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

print\_ip\_header(Buffer,Size);

fprintf(logfile , "\n");

fprintf(logfile , "TCP Header\n");

fprintf(logfile , " |-Source Port : %u\n",ntohs(tcph->source));

fprintf(logfile , " |-Destination Port : %u\n",ntohs(tcph->dest));

fprintf(logfile , " |-Sequence Number : %u\n",ntohl(tcph->seq));

fprintf(logfile , " |-Acknowledge Number : %u\n",ntohl(tcph->ack\_seq));

fprintf(logfile , " |-Header Length : %d DWORDS or %d BYTES\n" ,(unsigned int)tcph->doff,(unsigned int)tcph->doff\*4);

fprintf(logfile , " |-Urgent Flag : %d\n",(unsigned int)tcph->urg);

fprintf(logfile , " |-Acknowledgement Flag : %d\n",(unsigned int)tcph->ack);

fprintf(logfile , " |-Push Flag : %d\n",(unsigned int)tcph->psh);

fprintf(logfile , " |-Reset Flag : %d\n",(unsigned int)tcph->rst);

fprintf(logfile , " |-Synchronise Flag : %d\n",(unsigned int)tcph->syn);

fprintf(logfile , " |-Finish Flag : %d\n",(unsigned int)tcph->fin);

fprintf(logfile , " |-Window : %d\n",ntohs(tcph->window));

fprintf(logfile , " |-Checksum : %d\n",ntohs(tcph->check));

fprintf(logfile , " |-Urgent Pointer : %d\n",tcph->urg\_ptr);

fprintf(logfile , "\n");

fprintf(logfile , " DATA Dump ");

fprintf(logfile , "\n");

fprintf(logfile , "IP Header\n");

PrintData(Buffer,iphdrlen);

fprintf(logfile , "TCP Header\n");

PrintData(Buffer+iphdrlen,tcph->doff\*4);

fprintf(logfile , "Data Payload\n");

PrintData(Buffer + header\_size , Size - header\_size );

fprintf(logfile , "\n###########################################################");

}

void print\_udp\_packet(const u\_char \*Buffer , int Size)

{

unsigned short iphdrlen;

struct iphdr \*iph = (struct iphdr \*)(Buffer + sizeof(struct ethhdr));

iphdrlen = iph->ihl\*4;

struct udphdr \*udph = (struct udphdr\*)(Buffer + iphdrlen + sizeof(struct ethhdr));

int header\_size = sizeof(struct ethhdr) + iphdrlen + sizeof udph;

fprintf(logfile , "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UDP Packet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

print\_ip\_header(Buffer,Size);

fprintf(logfile , "\nUDP Header\n");

fprintf(logfile , " |-Source Port : %d\n" , ntohs(udph->source));

fprintf(logfile , " |-Destination Port : %d\n" , ntohs(udph->dest));

fprintf(logfile , " |-UDP Length : %d\n" , ntohs(udph->len));

fprintf(logfile , " |-UDP Checksum : %d\n" , ntohs(udph->check));

fprintf(logfile , "\n");

fprintf(logfile , "IP Header\n");

PrintData(Buffer , iphdrlen);

fprintf(logfile , "UDP Header\n");

PrintData(Buffer+iphdrlen , sizeof udph);

fprintf(logfile , "Data Payload\n");

//Move the pointer ahead and reduce the size of string

PrintData(Buffer + header\_size , Size - header\_size);

fprintf(logfile , "\n###########################################################");

}

void print\_icmp\_packet(const u\_char \* Buffer , int Size)

{

unsigned short iphdrlen;

struct iphdr \*iph = (struct iphdr \*)(Buffer + sizeof(struct ethhdr));

iphdrlen = iph->ihl \* 4;

struct icmphdr \*icmph = (struct icmphdr \*)(Buffer + iphdrlen + sizeof(struct ethhdr));

int header\_size = sizeof(struct ethhdr) + iphdrlen + sizeof icmph;

fprintf(logfile , "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ICMP Packet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

print\_ip\_header(Buffer , Size);

fprintf(logfile , "\n");

fprintf(logfile , "ICMP Header\n");

fprintf(logfile , " |-Type : %d",(unsigned int)(icmph->type));

if((unsigned int)(icmph->type) == 11)

{

fprintf(logfile , " (TTL Expired)\n");

}

else if((unsigned int)(icmph->type) == ICMP\_ECHOREPLY)

{

fprintf(logfile , " (ICMP Echo Reply)\n");

}

fprintf(logfile , " |-Code : %d\n",(unsigned int)(icmph->code));

fprintf(logfile , " |-Checksum : %d\n",ntohs(icmph->checksum));

//fprintf(logfile , " |-ID : %d\n",ntohs(icmph->id));

//fprintf(logfile , " |-Sequence : %d\n",ntohs(icmph->sequence));

fprintf(logfile , "\n");

fprintf(logfile , "IP Header\n");

PrintData(Buffer,iphdrlen);

fprintf(logfile , "UDP Header\n");

PrintData(Buffer + iphdrlen , sizeof icmph);

fprintf(logfile , "Data Payload\n");

//Move the pointer ahead and reduce the size of string

PrintData(Buffer + header\_size , (Size - header\_size) );

fprintf(logfile , "\n###########################################################");

}

void PrintData (const u\_char \* data , int Size)

{

//u\_char \*ptr=(u\_char \*)data;

// const char\* S1 = reinterpret\_cast<const char\*>(data);

// fprintf(logfile,"%s\n",S1);

int i , j;

for(i=0 ; i < Size ; i++)

{

if( i!=0 && i%16==0) //if one line of hex printing is complete...

{

fprintf(logfile , " ");

for(j=i-16 ; j<i ; j++)

{

if(data[j]>=32 && data[j]<=128)

fprintf(logfile , "%c",(unsigned char)data[j]); //if its a number or alphabet

else fprintf(logfile , "."); //otherwise print a dot

}

fprintf(logfile , "\n");

}

if(i%16==0) fprintf(logfile , " ");

fprintf(logfile , " %02X",(unsigned int)data[i]);

if( i==Size-1) //print the last spaces

{

for(j=0;j<15-i%16;j++)

{

fprintf(logfile , " "); //extra spaces

}

fprintf(logfile , " ");

for(j=i-i%16 ; j<=i ; j++)

{

if(data[j]>=32 && data[j]<=128)

{

fprintf(logfile , "%c",(unsigned char)data[j]);

}

else

{

fprintf(logfile , ".");

}

}

fprintf(logfile , "\n" );

}

}

}

//usage:

g++ filename.cpp -o file -lpcap

sudo ./file

-------------------------------------------------------------------------------------------------------------------------------

RPC

-------------------------------------------------------------------------------------------------------------------------------

.x file

struct arg{

};

program ADD\_PROG{

version ADD\_VERS{

return\_type fun(arg)=1;

}=1;

}=0x23451111;

usage:

rpcgen -a -C file.x

make -f Makefile.file

./file\_server

./file\_client localhost args

------------------------------------------------------------------------------------------------------------------------------------

ARP REPLY

------------------------------------------------------------------------------------------------------------------------------------

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netinet/in.h> // IPPROTO\_RAW, INET\_ADDRSTRLEN

#include <netinet/ip.h> // IP\_MAXPACKET (which is 65535)

#include <net/if.h>

#include <linux/if\_packet.h>

#include <net/ethernet.h>

struct arphdr

{

uint16\_t htype; //Hardware type

uint16\_t ptype; //Protocol type

uint8\_t hlen; //Hardware address length

uint8\_t plen; //Protocol address length

uint16\_t opcode; //Operation code

uint8\_t sender\_mac[6];

uint8\_t sender\_ip[4];

uint8\_t target\_mac[6];

uint8\_t target\_ip[4];

/\*

#define ARPOP\_REQUEST 1 // request to resolve address

#define ARPOP\_REPLY 2 // response to previous request

#define ARPOP\_REVREQUEST 3 // request protocol address given hardware

#define ARPOP\_REVREPLY 4 // response giving protocol address

#define ARPOP\_INVREQUEST 8 // request to identify peer

#define ARPOP\_INVREPLY 9 // response identifying peer

\*/

};

int sfd = socket(PF\_PACKET,SOCK\_RAW,htons(ETH\_P\_ALL));

char interface[40];

strcpy (interface, argv[5]);

if(sfd==-1)

{

perror("socket");

}

char\* buf = (char\*)malloc(1500);

uint8\_t src[6],dst[6];

//usr mac address

src[0] = 0x8C;

src[1] = 0x16;

src[2] = 0x45;

src[3] = 0xCE;

src[4] = 0x8B;

src[5] = 0xE4;

//local gateway mac address

dst[0] = 0x00;

dst[1] = 0x25;

dst[2] = 0x83;

dst[3] = 0x70;

dst[4] = 0x10;

dst[5] = 0x00;

memcpy(buf,dst,6\*(sizeof (uint8\_t)));

memcpy(buf+6\*(sizeof (uint8\_t)),src,6\*(sizeof (uint8\_t)));

buf[12] = ETH\_P\_ARP / 256;

buf[13] = ETH\_P\_ARP % 256;

struct arphdr\* arp = (struct arphdr\*)(buf+14);

arp->htype = htons(1); //because we use ethernet

arp->ptype = 8; // ETH\_P\_IP = 0x0800

arp->hlen = 6;

arp->plen = 4;

arp->opcode = htons(2); // ARP reply

memcpy(arp->sender\_mac ,src,6\*(sizeof(uint8\_t)));

memcpy(arp->target\_mac ,dst,6\*(sizeof(uint8\_t)));

// Friend's IP

arp->sender\_ip[0] = atoi(argv[1]);

arp->sender\_ip[1] = atoi(argv[2]);

arp->sender\_ip[2] = atoi(argv[3]);

arp->sender\_ip[3] = atoi(argv[4]);

//Gateway IP

arp->target\_ip[0] = 172;

arp->target\_ip[1] = 30;

arp->target\_ip[2] = 100;

arp->target\_ip[3] = 1;

memcpy(buf+14,arp,28);

int bytes;

struct sockaddr\_ll device;

memset (&device, 0, sizeof (device));

if ((device.sll\_ifindex = if\_nametoindex (interface)) == 0)

{

perror ("if\_nametoindex() failed to obtain interface index ");

exit (EXIT\_FAILURE);

}

printf ("Index for interface %s is %i\n", interface, device.sll\_ifindex);

device.sll\_family = AF\_PACKET;

memcpy (device.sll\_addr, dst, 6 \* sizeof (uint8\_t));

device.sll\_halen = 6;

while(1)

{

if ((bytes = sendto (sfd, buf,42, 0, (struct sockaddr \*) &device, sizeof (device))) <= 0)

{

perror ("sendto() failed");

exit (EXIT\_FAILURE);

}

}

-----------------------------------------------------------------------------------------------------------------------------------------

Own Ethernet Header( 14 bytes [6+6+2] )

-----------------------------------------------------------------------------------------------------------------------------------------

struct ether\_header

{

uint8\_t ether\_dhost[ETH\_ALEN]; /\* destination eth addr \*/

uint8\_t ether\_shost[ETH\_ALEN]; /\* source ether addr \*/

uint16\_t ether\_type; /\* packet type ID field \*/

};

char\* buf = (char\*)malloc(1500);

uint8\_t src[6],dst[6];

//usr mac address(6 bytes)

src[0] = 0x8C;

src[1] = 0x16;

src[2] = 0x45;

src[3] = 0xCE;

src[4] = 0x8B;

src[5] = 0xE4;

//destination mac address( 6 bytes )

dst[0] = 0x00;

dst[1] = 0x25;

dst[2] = 0x83;

dst[3] = 0x70;

dst[4] = 0x10;

dst[5] = 0x00;

memcpy(buf,dst,6\*(sizeof (uint8\_t)));

memcpy(buf+6\*(sizeof (uint8\_t)),src,6\*(sizeof (uint8\_t)));

//packet\_type (2 bytes)

buf[12] = ETH\_P\_ARP / 256;

buf[13] = ETH\_P\_ARP % 256;

-----------------------------------------------------------------------------------------------------------------------------------------

IP Header

-----------------------------------------------------------------------------------------------------------------------------------------

struct myipheader

{

#if \_\_BYTE\_ORDER == \_\_LITTLE\_ENDIAN

unsigned int ihl:4;

unsigned int version:4;

#elif \_\_BYTE\_ORDER == \_\_BIG\_ENDIAN

unsigned int version:4;

unsigned int ihl:4;

#else

# error "Please fix <bits/endian.h>"

#endif

uint8\_t tos;

uint16\_t tot\_len;

uint16\_t id;

uint16\_t frag\_off;

uint8\_t ttl;

uint8\_t protocol;

uint16\_t check;

uint32\_t saddr;

uint32\_t daddr;

/\*The options start here. \*/

};

struct myipheader \*ip;

char buffer[1000];

memset(buffer,'\0',sizeof(buffer));

ip=(struct myipheader\*)buffer;

ip->version=4;

ip->ttl=64;

ip->id=0;

ip->ihl=5;

ip->protocol=100; // value same as last argument(integer) of raw socket creation sys\_call

ip->saddr=inet\_addr("1.2.3.4");

ip->daddr=inet\_addr("127.0.0.2");

ip->tot\_len=1000;

printf("Msg:");

scanf("%s",buffer+60);

if(sendto(rsfd,buffer,1000,0,(struct sockaddr\*)&serv\_addr,sizeof(serv\_addr))<0)

{

perror("sendto");

exit(1);

}

------------------------------------------------------------------------------------------------------------------------------------------

TCP Header

------------------------------------------------------------------------------------------------------------------------------------------

struct tcphdr

{

\_\_extension\_\_ union

{

struct

{

uint16\_t th\_sport; /\* source port \*/

uint16\_t th\_dport; /\* destination port \*/

tcp\_seq th\_seq; /\* sequence number \*/

tcp\_seq th\_ack; /\* acknowledgement number \*/

# if \_\_BYTE\_ORDER == \_\_LITTLE\_ENDIAN

uint8\_t th\_x2:4; /\* (unused) \*/

uint8\_t th\_off:4; /\* data offset \*/

# endif

# if \_\_BYTE\_ORDER == \_\_BIG\_ENDIAN

uint8\_t th\_off:4; /\* data offset \*/

uint8\_t th\_x2:4; /\* (unused) \*/

# endif

uint8\_t th\_flags;

# define TH\_FIN 0x01

# define TH\_SYN 0x02

# define TH\_RST 0x04

# define TH\_PUSH 0x08

# define TH\_ACK 0x10

# define TH\_URG 0x20

uint16\_t th\_win; /\* window \*/

uint16\_t th\_sum; /\* checksum \*/

uint16\_t th\_urp; /\* urgent pointer \*/

};

struct

{

uint16\_t source;

uint16\_t dest;

uint32\_t seq;

uint32\_t ack\_seq;

# if \_\_BYTE\_ORDER == \_\_LITTLE\_ENDIAN

uint16\_t res1:4;

uint16\_t doff:4;

uint16\_t fin:1;

uint16\_t syn:1;

uint16\_t rst:1;

uint16\_t psh:1;

uint16\_t ack:1;

uint16\_t urg:1;

uint16\_t res2:2;

# elif \_\_BYTE\_ORDER == \_\_BIG\_ENDIAN

uint16\_t doff:4;

uint16\_t res1:4;

uint16\_t res2:2;

uint16\_t urg:1;

uint16\_t ack:1;

uint16\_t psh:1;

uint16\_t rst:1;

uint16\_t syn:1;

uint16\_t fin:1;

# else

# error "Adjust your <bits/endian.h> defines"

# endif

uint16\_t window;

uint16\_t check;

uint16\_t urg\_ptr;

};

};

};

// sample initialization

u\_char buffer[1500];

struct ether\_header\* eth;

struct iphdr\* ip;

struct tcphdr\* tcp;

tcp = (struct tcphdr\*)(buffer+14+(ip->ihl\*4));

tcp->dest = //destination port;

tcp->source = //source port;

tcp->doff = // length of tcp header

tcp->seq = // sequence number;

tcp->ack\_seq = // acknowledgement number

tcp->syn = // SYN bit

tcp->urg = // urgent bit

tcp->window = // window size

-------------------------------------------------------------------------------------------------------------------------------------------

UDP Header

------------------------------------------------------------------------

struct udphdr

{

\_\_extension\_\_ union

{

struct

{

uint16\_t uh\_sport; /\* source port \*/

uint16\_t uh\_dport; /\* destination port \*/

uint16\_t uh\_ulen; /\* udp length \*/

uint16\_t uh\_sum; /\* udp checksum \*/

};

struct

{

uint16\_t source;

uint16\_t dest;

uint16\_t len;

uint16\_t check;

};

};

};

#include<stdio.h>; //for printf

#include<string.h>; //memset

#include<sys/socket.h>; //for socket ofcourse

#include<stdlib.h>; //for exit(0);

#include<errno.h>; //For errno - the error number

#include<netinet/udp.h>; //Provides declarations for udp header

#include<netinet/ip.h>; //Provides declarations for ip header

/\*

96 bit (12 bytes) pseudo header needed for udp header checksum calculation

\*/

struct pseudo\_header

{

u\_int32\_t source\_address;

u\_int32\_t dest\_address;

u\_int8\_t placeholder;

u\_int8\_t protocol;

u\_int16\_t udp\_length;

};

/\*

Generic checksum calculation function

\*/

unsigned short csum(unsigned short \*ptr,int nbytes)

{

register long sum;

unsigned short oddbyte;

register short answer;

sum=0;

while(nbytes>;1) {

sum+=\*ptr++;

nbytes-=2;

}

if(nbytes==1) {

oddbyte=0;

\*((u\_char\*)&oddbyte)=\*(u\_char\*)ptr;

sum+=oddbyte;

}

sum = (sum>>16)+(sum & 0xffff);

sum = sum + (sum>>16);

answer=(short)~sum;

return(answer);

}

/Create a raw socket of type IPPROTO

int s = socket (AF\_INET, SOCK\_RAW, IPPROTO\_RAW);

if(s == -1)

{

//socket creation failed, may be because of non-root privileges

perror("Failed to create raw socket");

exit(1);

}

//Datagram to represent the packet

char datagram[4096] , source\_ip[32] , \*data , \*pseudogram;

//zero out the packet buffer

memset (datagram, 0, 4096);

//Datagram to represent the packet

char datagram[4096] , source\_ip[32] , \*data , \*pseudogram;

//zero out the packet buffer

memset (datagram, 0, 4096);

//IP header

struct iphdr \*iph = (struct iphdr \*) datagram;

//UDP header

struct udphdr \*udph = (struct udphdr \*) (datagram + sizeof (struct ip));

struct sockaddr\_in sin;

struct pseudo\_header psh;

//Data part

data = datagram + sizeof(struct iphdr) + sizeof(struct udphdr);

strcpy(data , "ABCDEFGHIJKLMNOPQRSTUVWXYZ");

//some address resolution

strcpy(source\_ip , "192.168.1.2");

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(80);

sin.sin\_addr.s\_addr = inet\_addr ("192.168.1.1");

//Fill in the IP Header

iph->ihl = 5;

iph->version = 4;

iph->tos = 0;

iph->tot\_len = sizeof (struct iphdr) + sizeof (struct udphdr) + strlen(data);

iph->id = htonl (54321); //Id of this packet

iph->frag\_off = 0;

iph->ttl = 255;

iph->protocol = IPPROTO\_UDP;

iph->check = 0; //Set to 0 before calculating checksum

iph->saddr = inet\_addr ( source\_ip ); //Spoof the source ip address

iph->daddr = sin.sin\_addr.s\_addr;

//Ip checksum

iph->check = csum ((unsigned short \*) datagram, iph->tot\_len);

//UDP header

udph->source = htons (6666);

udph->dest = htons (8622);

udph->len = htons(8 + strlen(data)); //udp header size

udph->check = 0; //leave checksum 0 now, filled later by pseudo header

//Now the UDP checksum using the pseudo header

psh.source\_address = inet\_addr( source\_ip );

psh.dest\_address = sin.sin\_addr.s\_addr;

psh.placeholder = 0;

psh.protocol = IPPROTO\_UDP;

psh.udp\_length = htons(sizeof(struct udphdr) + strlen(data) );

int psize = sizeof(struct pseudo\_header) + sizeof(struct udphdr) + strlen(data);

pseudogram = malloc(psize);

memcpy(pseudogram , (char\*) &psh , sizeof (struct pseudo\_header));

memcpy(pseudogram + sizeof(struct pseudo\_header) , udph , sizeof(struct udphdr) + strlen(data));

udph->check = csum( (unsigned short\*) pseudogram , psize);

//Send the packet

if (sendto (s, datagram, iph->tot\_len , 0, (struct sockaddr \*) &sin, sizeof (sin)) < 0)

{

perror("sendto failed");

}

//Data send successfully

else

{

printf ("Packet Send. Length : %d \n" , iph->tot\_len);

}

------------------------------------------------------------------------------------------------------------------------------------------

Pcap filter Code

------------------------------------------------------------------------------------------------------------------------------------------

s = socket (PF\_INET, SOCK\_RAW, IPPROTO\_TCP);perror("socket");

char \*device = "lo";

struct bpf\_program filter;

char error\_buffer[PCAP\_ERRBUF\_SIZE];

pcap\_t \*handle;

char filter\_exp[] = "src host 127.0.0.1 || src host 127.0.0.2";

bpf\_u\_int32 subnet\_mask, ip;

/\* Snapshot length is how many bytes to capture from each packet. This includes\*/

int snapshot\_length = 1024;

/\* End the loop after this many packets are captured \*/

int total\_packet\_count = 10;

u\_char \*my\_arguments = NULL;

pcap\_lookupnet(device, &ip, &subnet\_mask, error\_buffer) ;

handle = pcap\_open\_live(device, snapshot\_length, 0, 10000, error\_buffer);

pcap\_compile(handle, &filter, filter\_exp, 0, ip) ;

pcap\_setfilter(handle, &filter);

pcap\_loop(handle, total\_packet\_count, my\_packet\_handler, my\_arguments);

-------------------------------------------------------------------------------------------------------------------------------------------

Generic checksum calculation function

------------------------------------------------------------------------------------------------------------------------------------------

unsigned short csum(unsigned short \*ptr,int nbytes)

{

register long sum;

unsigned short oddbyte;

register short answer;

sum=0;

while(nbytes>;1) {

sum+=\*ptr++;

nbytes-=2;

}

if(nbytes==1) {

oddbyte=0;

\*((u\_char\*)&oddbyte)=\*(u\_char\*)ptr;

sum+=oddbyte;

}

sum = (sum>>16)+(sum & 0xffff);

sum = sum + (sum>>16);

answer=(short)~sum;

return(answer);

}

sample call

//Ip checksum

iph->check = csum ((unsigned short \*) data, iph->tot\_len);

------------------------------------------------------------------------------------------------------------------------------------

PCAP Monitor Mode

------------------------------------------------------------------------------------------------------------------------------------

char error\_buffer[PCAP\_ERRBUF\_SIZE];

pcap\_t \*handle = pcap\_create("wlan0", error\_buffer);

pcap\_set\_rfmon(handler, 1);

pcap\_set\_promisc(handler, 1); /\* Capture packets that are not yours \*/

pcap\_set\_snaplen(handler, 2048); /\* Snapshot length \*/

pcap\_set\_timeout(handler, 1000); /\* Timeout in milliseconds \*/

pcap\_activate(handle);

------------------------------------------------------------------------------------------------------------------------------------

PCAP My Handler

------------------------------------------------------------------------------------------------------------------------------------

void my\_packet\_handler(

u\_char \*args,

const struct pcap\_pkthdr \*header,

const u\_char \*packet

)

{

struct ether\_header \*eth\_header;

eth\_header = (struct ether\_header \*) packet;

if (ntohs(eth\_header->ether\_type) != ETHERTYPE\_IP) {

printf("Not an IP packet. Skipping...\n\n");

return;

}

const u\_char \*ip\_header;

const u\_char \*tcp\_header;

const u\_char \*payload;

int ethernet\_header\_length = 14; /\* Doesn't change \*/

int ip\_header\_length;

int tcp\_header\_length;

int payload\_length;

ip\_header = packet + ethernet\_header\_length;

ip\_header\_length = ((\*ip\_header) & 0x0F);

ip\_header\_length = ip\_header\_length \* 4;

printf("IP header length (IHL) in bytes: %d\n", ip\_header\_length);

u\_char protocol = \*(ip\_header + 9);

if (protocol != IPPROTO\_TCP) {

printf("Not a TCP packet. Skipping...\n\n");

return;

}

tcp\_header = packet + ethernet\_header\_length + ip\_header\_length;

tcp\_header\_length = ((\*(tcp\_header + 12)) & 0xF0) >> 4;

tcp\_header\_length = tcp\_header\_length \* 4;

printf("TCP header length in bytes: %d\n", tcp\_header\_length);

int total\_headers\_size = ethernet\_header\_length+ip\_header\_length+tcp\_header\_length;

printf("Size of all headers combined: %d bytes\n", total\_headers\_size);

payload\_length = header->caplen -

(ethernet\_header\_length + ip\_header\_length + tcp\_header\_length);

printf("Payload size: %d bytes\n", payload\_length);

payload = packet + total\_headers\_size;

printf("Memory address where payload begins: %p\n\n", payload);

/\*

if (payload\_length > 0) {

const u\_char \*temp\_pointer = payload;

int byte\_count = 0;

while (byte\_count++ < payload\_length) {

printf("%c", \*temp\_pointer);

temp\_pointer++;

}

printf("\n");

}

\*/

return;

}

------------------------------------------------------------------------------------------------------------------------------------