AIM: To implement programs using raw sockets (like packet capturing and filtering).

ALGORITHM :

1. Start the program and to include the necessary header files
2. To define the packet length
3. To declare the IP header structure using TCPheader
4. Using simple checksum process to check the process
5. Using TCP \IP communication protocol to execute the program
6. And using TCP\IP communication to enter the Source IP and port number and Target IP address and port number.
7. The Raw socket () is created and accept the Socket ( ) and Send to ( ), ACK
8. Stop the program

//---cat rawtcp.c---

// Run as root or SUID 0, just datagram no data/payload

Program:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
// Packet length
#define PCKT_LEN 8192
```

```
// May create separate header file (.h) for all
// headers' structures
// IP header's structure
struct ipheader {
unsigned char iph_ihl:5, /* Little-endian */
iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned char iph_flags;
unsigned short int iph_offset;
unsigned char iph_ttl;
unsigned char iph_protocol;
unsigned short int iph_chksum;
unsigned int iph_sourceip;
unsigned int iph_destip;
};
/* Structure of a TCP header */
struct tcpheader {
unsigned short int tcph_srcport;
unsigned short int tcph_destport;
unsigned int tcph_seqnum;
unsigned int tcph_acknum;
unsigned char tcph_reserved:4, tcph_offset:4;
// unsigned char tcph_flags;
unsigned int
tcp_res1:4, /*little-endian*/
tcph_hlen:4, /*length of tcp header in 32-bit
words*/
tcph_fin:1, /*Finish flag "fin"*/
tcph_syn:1, /*Synchronize sequence numbers to
```

```
start a connection*/
tcph_rst:1, /*Reset flag */
tcph_psh:1, /*Push, sends data to the
application*/
tcph_ack:1, /*acknowledge*/
tcph_urg:1, /*urgent pointer*/
tcph_res2:2;
unsigned short int tcph_win;
unsigned short int tcph_chksum;
unsigned short int tcph_urgptr;
};
// Simple checksum function, may use others such as Cyclic
Redundancy Check, CRC
unsigned short csum(unsigned short *buf, int len)
{
unsigned long sum;
for(sum=0; len>0; len--)
sum += *buf++;
sum = (sum >> 16) + (sum &0xffff);
sum += (sum >> 16);
return (unsigned short)(~sum);
}
int main(int argc, char *argv[])
{
int sd;
// No data, just datagram
char buffer[PCKT_LEN];
// The size of the headers
struct ipheader *ip = (struct ipheader *) buffer;
struct tcpheader *tcp = (struct tcpheader *) (buffer +
sizeof(struct ipheader));
```

```
struct sockaddr_in sin, din;
int one = 1;
const int *val = &one;
memset(buffer, 0, PCKT_LEN);
if(argc != 5)
{
printf("- Invalid parameters!!!\n");
printf("- Usage: %s <source hostname/IP> <source port>
<target hostname/IP> <target port>\n", argv[0]);
exit(-1);
}
sd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
if(sd < 0)
{
perror("socket() error");
exit(-1);
}
else
printf("socket()-SOCK_RAW and tcp protocol is OK.\n");
// The source is redundant, may be used later if needed
// Address family
sin.sin_family = AF_INET;
din.sin_family = AF_INET;
// Source port, can be any, modify as needed
sin.sin_port = htons(atoi(argv[2]));
din.sin_port = htons(atoi(argv[4]));
// Source IP, can be any, modify as needed
sin.sin_addr.s_addr = inet_addr(argv[1]);
din.sin_addr.s_addr = inet_addr(argv[3]);
// IP structure
ip->iph_ihl = 5;
```

```
ip->iph_ver = 4;
ip->iph_tos = 16;
ip->iph_len = sizeof(struct ipheader) + sizeof(struct
tcpheader);
ip->iph_ident = htons(54321);
ip->iph_offset = 0;
ip->iph_ttl = 64;
ip->iph_protocol = 6; // TCP
ip->iph_chksum = 0; // Done by kernel
// Source IP, modify as needed, spoofed, we accept through
command line argument
ip->iph_sourceip = inet_addr(argv[1]);
// Destination IP, modify as needed, but here we accept
through command line argument
ip->iph_destip = inet_addr(argv[3]);
// The TCP structure. The source port, spoofed, we accept
through the command line
tcp->tcph_srcport = htons(atoi(argv[2]));
// The destination port, we accept through command line
tcp->tcph_destport = htons(atoi(argv[4]));
tcp->tcph_seqnum = htonl(1);
tcp->tcph_acknum = 0;
tcp->tcph_offset = 5;
tcp->tcph_syn = 1;
tcp->tcph_ack = 0;
tcp->tcph_win = htons(32767);
tcp->tcph_chksum = 0; // Done by kernel
tcp->tcph_urgptr = 0;
// IP checksum calculation
ip->iph_chksum = csum((unsigned short *) buffer,
(sizeof(struct ipheader) + sizeof(struct tcpheader)));
```

```
// Inform the kernel do not fill up the headers' structure,
we fabricated our own
if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))
< 0)
{
perror("setsockopt() error");
exit(-1);
}
else
printf("setsockopt() is OK\n");
printf("Using:::::Source IP: %s port: %u, Target IP: %s
port: %u.\n", argv[1], atoi(argv[2]), argv[3],
atoi(argv[4]));
// sendto() loop, send every 2 second for 50 counts
unsigned int count;
for(count = 0; count < 20; count++)
{
if(sendto(sd, buffer, ip->iph_len, 0, (struct sockaddr
*)&sin, sizeof(sin)) < 0)
// Verify
{
perror("sendto() error");
exit(-1);
}
else
printf("Count #%u - sendto() is OK\n", count);
sleep(2);
}
close(sd);
return 0;
}
```

RESULT :

Thus the Above programs using raw sockets TCP \IP (like packet capturing and filtering) was executed and successfully.

Outcome:

To understand packet filtering and capturing using raw sockets.