Implémentation d’un sniffer passif dans un réseau local

Algoritme :

\*fonction Creation\_socket(protocol\_to\_sniff:entier):entier

var

 rawsock:entier

debut

{si

((rawsock=socket(PF\_PACKET,SOCK\_RAW,htons(protocol\_to\_sniff)))=­-1)

   ecrire("Erreur de creation de socket!! « )}

retourner(rawsock)

fin

\*fonction liaison\_Socket\_à\_interface(device:chaine de caracteres, i:entier, protocol:entier)

type sockaddr\_ll=enregistrement

sll\_family:entier

sll\_protocole:entier

sll\_ifindex:entier

sll\_hatype:entier

sll\_pkttype:caractere

sll\_halen:caractere

sll\_addr[8]:chaine de caractere

fin

type ifreq=enregistrement

ifr\_name[]:chaine

type

type sockaddr=enregistrement

^sock

type ifreq=enregistrement

^ifr

debut

bzero(&sock, sizeof(sock))

bzero(&ifr, sizeof(ifr))

strncpy(ifr.ifr\_name, device, IFNAMSIZ)

   si((ioctl(rawsock, SIOCGIFINDEX, &ifr)) =-1)

    {ecrire ("Erreur d'obtenir l'index interface !!")

              }

sock.sock\_family = AF\_PACKET

sock.sock\_ifindex = ifr.ifr\_ifindex

sock.sock\_protocol = htons(protocol)

        si((bind(rawsock, (^sockaddr )&sock, sizeof(sll)))= -1)

          { ecrire("Erreur de lier socket à l'interface !! »)}

fin

\*procedure Analyser\_EthernetHeader(packet:chaine de caractere, a:entier)

type

type ethhdr=enregistrement

^ethernet\_header

debut

si(a > sizeof(ethhdr))

        {

                ethernet\_header = ( ethhdr )->packet

                PrintInHex("Destination MAC: ", ethernet\_header.h\_dest, 6)

                PrintInHex("Source MAC: ", ethernet\_header.h\_source, 6)

                PrintInHex("Protocol: « ,^ethernet\_header.h\_proto, 2)}

sinon

ecrire ("taille de packet est insuffisant!! »)

fin

\*procedure PrintPacketInHex( p:chaine de caractere,  a:entier)

{var

i:entier

debut

        ecrire("Packet debut")

        pour i de a à 0 (pas -1)

        {

                ecrire(p[i-a])

        }

        ecrire("packet terminé")

}

fin

\*procedure PrintInHex(ch:chaine de caractere, p:chaine,  a:entier)

{var

i:entier

debut

        ecrire(ch)

        pour i de a à 0 (pas -1)

        {

                ecrire(p[i-a])

        }

}fin

\*procedure strncpy(ch1:chaine,ch2:chaine,n:entier)

var

i:entier

debut

ch1="

pour i de 1 à n faire

ch1[i]=ch2[i]

fin

\*procedure Analyser\_IpHeader(packet:chaine de caracteres, a:entier)

type

type ethhdr=enregistrement

^ethernet\_header

type iphdr=enregistrement

^ip\_header

ethernet\_header = (^struct ethhdr )packet

debut

 si (ntohs(ethernet\_header->h\_proto) = ETH\_P\_IP){

 si (a >= (sizeof(struct ethhdr) + sizeof(struct iphdr))

{ip\_header = (^struct iphdr)(packet + sizeof(struct ethhdr))

 ecrire ("TTL: »,ip\_header.ttl)

                ecrire ("Adresse de Destination IP: ", inet\_ntoa( ^(struct in\_addr)&ip\_header->daddr))

                ecrire("Adresse de Source IP : ", inet\_ntoa( ^(struct in\_addr)&ip\_header->saddr))}

sinon

{ecrire("Le paquet IP n'a pas d'en-tête complet !! »)}

sinon

{ecrire(" Pas un paquet IP!! »)}

fin

\*procedure AnalyserTcpHeader(packet:chaine de caracteres , a:entier)

type

type ethhdr=enregistrement

^ethernet\_header

type iphdr=enregistrement

^ip\_header

type tcphdr=enregistrement

^tcp\_header

debut

si(a>= (sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct tcphdr)))

{ethernet\_header = (struct ethhdr \*)packet

si (ntohs(ethernet\_header.h\_proto) = ETH\_P\_IP)

 { ip\_header = (^struct iphdr )(packet + sizeof(struct ethhdr))

si (ip\_header.protocol =IPPROTO\_TCP)

{ecrire("----------------------------egment (TCP protocol num=" ,ip\_header.protocol)

 tcp\_header = (^struct tcphdr)(packet + sizeof(struct ethhdr) + ip\_header.ihl\*4 )

 ecrire("Source Port: ", ntohs(tcp\_header.source))

          ecrire("Destination Port: %d\n", ntohs(tcp\_header-.dest))

                        }

                        sinon

                        {

                                ecrire("Pas un paquet TCP")

                        }

                }

                sinon

                {

                        ecrire("Pas un paquet IP")

                }

        }

        sinon

        {

                ecrire("En-tête TCP non présent")

        }

fin

\*procedure AnalyserUdpHeader(packet:chaine de caracteres , a:entier)

type

type ethhdr=enregistrement

^ethernet\_header

type iphdr=enregistrement

^ip\_header

type udphdr=enregistrement

^udp\_header

debut

 si (a>= (sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct udphdr)))

 {ethernet\_header = (struct ethhdr \*)packet

                si (ntohs(ethernet\_header->h\_proto) =ETH\_P\_IP)

                {

                        ip\_header = (^iphdr )(packet + sizeof(struct ethhdr))

                        si (ip\_header.protocol = IPPROTO\_UDP)

                           {ecrire("UDP datagram (UDP num)",   ip\_header.protocol)

                                udp\_header = (^ udphdr)(packet + sizeof(struct ethhdr) + ip\_header->ihl\*4 )

                                ecrire("Source Port: ", ntohs(udp\_header.source))

                                ecrire("Destination Port:", ntohs(udp\_header.dest))

                        }

                        sinon

                        {

                                ecrire("Pas un paquet UDP!! ")

                        }

                }

                sinon

                {

                        ecrire("Pas un paquet IP")

                }

        }

        sinon

        {

                ecrire("UDP Header non présent ")

        }

}

\*fonction ParseData(packet:chaine de caractere,a:entier):entier

type

      type ethhdr=enregistrement

^ethernet\_header

type iphdr=enregistrement

^ip\_header

      type tcphdr=enregistrement

^tcp\_header

var

        data:chaine

        len:entier

debut

        si(a > (sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct tcphdr)))

        {

                ip\_header = ( iphdr^)(packet + sizeof(struct ethhdr))

                data = (packet + sizeof(struct ethhdr) + ip\_header.ihl\*4 +sizeof( tcphdr))

                len = ntohs(ip\_header->tot\_len) - ip\_header.ihl\*4 - sizeof( tcphdr)

                si(len){

   ecrire("Data Len :", len)

                        ecrire(data)

                        ecrire("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

                        PrintInHex("Data : ", data,len)

                        retourner 1

                }

                sinon

                {

                        ecrire("Aucune donnée dans le paquet!! ")

                        retourner 0

                }

        }

        sinon

        {

                ecrire("Aucune donnée dans le paquet!! ")

                retourner 0

        }

}

fin

\*fonction  IsIpAndTcpPacket(packet:chaine, len:entier):entier

type

         type ethhdr=enregistrement

^ethernet\_header

type iphdr=enregistrement

^ip\_header

        ethernet\_header = (^ ethhdr )packet

debut

        si (ntohs(ethernet\_header.h\_proto) = ETH\_P\_IP)

        {

                ip\_header = (^struct iphdr )(packet + sizeof( ethhdr))

                si(ip\_header.protocol =IPPROTO\_TCP)

                        retourner 1

                sinon

                        retourner -1

        }

        sinon

        {

                retourner -1

        }

}

fin

\*procedure main(argc:entier,argv:chaine)

type

sockaddr\_ll=enregistrement

packet\_info=enregistrement

var

 int raw;

        packet\_buffer[2048]:chaine

        len:entier

        packets\_to\_sniff:entier

     packet\_info\_size:entier

debut

 packet\_info\_size=sizeof(packet\_info

 si(argc < 3 )

        {

                ecrire("Usage: ./sniffer <Interface> <Nbr of packets to sniff>")

                retourner (EINVAL)

        }

 raw = Creation\_socket(ETH\_P\_IP)

        liaison\_Socket\_à\_interface(argv[1], raw, ETH\_P\_IP)

        packets\_to\_sniff = atoi(argv[2])

tant que(packets\_to\_sniff>0)

        {

                si((len = recvfrom(raw, packet\_buffer, 2048, 0, (sockaddr^)&packet\_info, (socklen\_t^)&packet\_info\_size)) = -1)

                {

                        ecrire("Recv de retour -1: ")

                }

              sinon

                {

                        PrintPacketInHex(packet\_buffer, len)

                        Analyser\_EthernetHeader(packet\_buffer, len)

                        Analyser\_IpHeader(packet\_buffer, len)

                        AnalyserTcpHeader(packet\_buffer, len)

                        AnalyserUdpHeader(packet\_buffer, len)

                        si(IsIpAndTcpPacket(packet\_buffer, len))

                        {

                                si(!ParseData(packet\_buffer, len))

                                        packets\_to\_sniff=packet\_to\_sniff+1

                        }

                }

packet\_to\_sniff=packet\_to\_sniff-1

        }fin tant que

fin