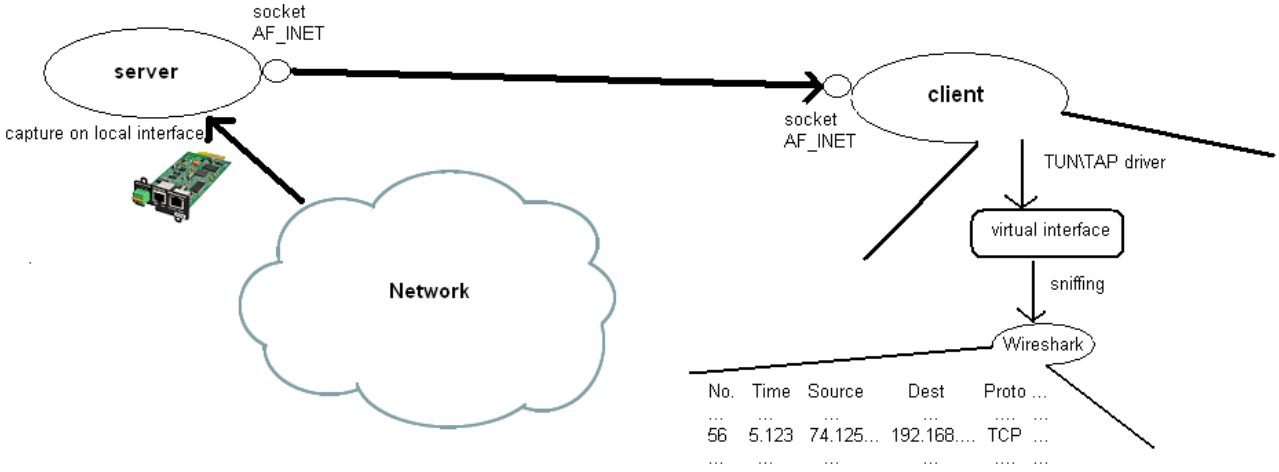
Gestione di Rete
a.a. 2009/2010 “Implementazione di uno sniffer remoto via TUN\TAP”

COLUCCI MARCO
426354
corso A

1. CARATTERISTICHE GENERALI

L'applicazione sfrutta il modello client-server, in particolare non era richiesto un server multithreading di conseguenza è stato previsto un solo client(non era la programmazione concorrente il tema principale del corso).
Il processo remoto(server) attende la richiesta di connessione da parte del client, cattura pacchetti da una specifica interfaccia di rete locale e li spedisce al client il quale a sua volta li trasferisce sull'interfaccia di rete virtuale opportunamente creata utilizzando i driver TUN\TAP[1].
Lo scopo principale è quello di utilizzare tale interfaccia di rete virtuale mediante un software di analisi dei pacchetti(Wireshark[2] in questo caso) al fine di realizzare uno sniffer remoto.



2. DESCRIZIONE MODULI

L'applicazione complessivamente consiste di due file sorgenti che descrivono rispettivamente client e server(*client.c*, *server.c*) con l'aggiunta di un file(*utility.c* e relativo header) che realizza una libreria minimale per implementare lo scambio di messaggi tra le due entità.
Viene data una descrizione generale del contenuto e del funzionamento di ciascun modulo, per i dettagli si rimanda ai commenti presenti nei rispettivi file.

2.1 utility.h

File header che definisce il tipo del messaggio scambiato tra client e server e le primitive per inviare e ricevere. In particolare viene definito il tipo struttura *packet_t* formato da due campi: lunghezza del messaggio e puntatore al buffer che lo contiene.
All' interno della funzione *sendPacket* viene inviata prima la lunghezza del messaggio, in modo da permettere al client di conoscerne la dimensione esatta, e solo dopo viene inviato il messaggio vero e proprio(trattasi del pacchetto catturato dal server). Infatti nella *receivePacket* viene controllato che il messaggio ricevuto sul socket abbia esattamente la lunghezza attesa. Se così non fosse si andrebbero a trasferire un numero di byte errati sull'interfaccia virtuale e di conseguenza l'analizzatore di pacchetti riscontrerebbe un pacchetto malformato nella fase di decodifica.

2.2 server.c

Il programma si aspetta come parametri d'ingresso il nome dell'interfaccia da cui catturare i pacchetti, il nome dell'interfaccia da utilizzare per comunicare con il client, il numero di porta da utilizzare per accettare connessioni ed il nome del file di log da creare. Nel caso il numero dei parametri sia scorretto oppure il numero della porta non sia valido viene visualizzato un opportuno messaggio di errore.

Successivamente si procede alla creazione del socket utilizzato per accettare richieste di connessione mediante la system call:

...socket*(AF_INET,SOCK_STREAM,0);*

con la quale si definisce un socket TCP, indispensabile per poter eseguire la listen successiva.
Viene creato il socket utilizzato per catturare i pacchetti mediante la system call:

...socket*(PF_PACKET,SOCK_RAW,htons(ETH_P_ALL));*

In questo modo tutti i pacchetti catturati dall'interfaccia di rete vengono consegnati direttamente all'applicazione in user-space scavalcando lo stack TCP/IP ed evitando così il demultiplexing.
Al fine di limitare la cattura dei pacchetti ad una singola interfaccia di rete viene eseguita una bind utilizzando il nome dell'interfaccia passato come parametro d'ingresso al programma.
Successivamente, una volta accettata la connessione da parte del client, inizia il ciclo infinito: il pacchetto catturato dall'interfaccia viene memorizzato nella struttura dati di tipo *packet_t* ed inviato al client senza operare alcun tipo di modifica. Inoltre per ogni pacchetto catturato e correttamente inviato viene aggiunta una riga nel file di log: identificativo pacchetto, byte catturati e byte spediti.
Al fine di garantire la corretta terminazione del processo server, i segnali SIGTERM e SIGINT vengono intercettati: il gestore di tali segnali provvede a liberare le risorse(memoria allocata e socket)e a stampare su stdout il numero di pacchetti correttamente catturati(il comportamento è analogo in caso di errore nel ciclo infinto).

2.3 client.c

Il programma si aspetta come parametri d'ingresso l'indirizzo IP del server e il numero di porta utilizzato per accettare connessioni. Nel caso il numero dei parametri sia scorretto oppure il numero della porta non sia valido viene visualizzato un opportuno messaggio di errore.
Si procede alla creazione dell'interfaccia virtuale utilizzando la funzione *tun_alloc*[3]: mediante TUN/TAP crea l'interfaccia virtuale(il cui nome prestabilito è “remote_interface”) e restituisce un numero intero che rappresenta il file descriptor speciale da utilizzare per comunicare con tale interfaccia. Va sottolineato che l'interfaccia creata è di tipo TAP in quanto si vuole lavorare con i frame Ethernet e che subito dopo la creazione l'interfaccia viene esplicitamente attivata(funzione *set_interface_up*).
Succesivamente viene creato il socket per comunicare con il server e relativa *connect*.
Stabilita la connessione inizia il ciclo infinito: il pacchetto ricevuto dal server viene trasferito sull'interfaccia virtuale utilizzando la system call *write* ed il file descriptor speciale precedentemente accennato.
Al fine di garantire la corretta terminazione del processo client, i segnali SIGTERM e SIGINT vengono intercettati: il gestore di tali segnali provvede a chiudere i socket aperti e stampare su stdout il numero di pacchetti catturati(il comportamento è analogo in caso di errore nel ciclo infinito).

3. PREREQUISITI

L'applicazione funziona in ambiente Linux. Il client necessita dell'installazione dei driver TUN\TAP(tipicamente già presenti nella maggior parte delle distribuzioni). Per caricare il relativo modulo utilizzare il comando : ' modprobe tun '.
Da sottolineare infine che dato il tipo di operazioni effettuate sia client che server per poter essere eseguiti necessitano dei diritti di amministratore.

4. ISTRUZIONI PER L'AVVIO

server
il server viene attivato da terminale mediante il comando:
./server *if_sniff if_client port file_log*
con
if_sniff : nome dell'interfaccia da cui catturare i pacchetti
if_client : nome dell'interfaccia da utilizzare per comunicare con il client
port : numero della porta su cui accettare connessioni
file_log : nome del file di log da creare(se non esiste)
(tutti i parametri devono essere specificati)

client
il client viene attivato da terminale mediante il comando:
./client *IP_server port*
con
IP_server : indirizzo ip del server
port : numero della porta su cui il server è in ascolto
(tutti i parametri devono essere specificati)

5. RIFERIMENTI

[1] http://vtun.sourceforge.net/tun/
[2] http://www.wireshark.org/
[3] http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt