SOCKET : Ovvero paragone tra i vari linguaggi & C.

by

Michael Bakunin

<bakunin@meganmail.com>

Un po' di teoria.... condita di qualche esempio!
Gia' perche' l'Hacking non e' solo pratica, ma anche (e forse soprattutto) teoria.

L'articolo in cui oggi cerchero' di cimentarmi riguarda la spiegazione dei socket, il confronto dell'uso di essi nei vari linguaggi, esempi e idee varie. Ovviamente nei vari linguaggi intendo solo quelli che ritengo prevalentemente atti allo scopo e questi sono il C, il Visual Basic e il TCL. Perche' prendo questi in esempio? Diciamo che la ragione principale e' che questi linguaggi hanno caratteristiche molto diverse fra loro, ma molto in comune con altri. Quindi ritengo inutile mostrarvi anche la programmazione dei socket in PERL o il altri linguaggi per SCRIPT, ma solo il TCL. Oppure ritengo inutile spiegarmi anche il programmazione dei socket in C++ perche' il C gia' ne mostra un esempio chiarificatore. Con cio' non intendo dire che non ci siano differenze tra un linguaggio e l'altro (tra C e C++ come tra PERL e TCL), ma ritendo esserci più somiglianze che differenze.

Ma direi subito di iniziare spiegando cosa sono i SOCKET!

--> Un socket e' un modo per far comunicare due programmi da remoto <--

Questo vuol dire che un socket e' un meccanismo (che puo' variare a seconda del linguaggio) che permette di mettere in comumicazione 2 programmi distinti attraverso un determinato protocollo (TCP/IP o UDP). Questi due programmi per incontrasi hanno bisogno di un indirizzo comune. Questo e' l'IP del SERVER.

Rincominciamo questo punto cercando di essere più chiari! Quando si decide di metterci in contatto con una persona, bisogna:

1 Sapere come comunicare con lei. Supponiamo di voler comunicare con il telefono (...o con un determinato PROTOCOLLO Es: TCP/IP)

2 Sapere il suo numero di telefono (...o il suo IP)

3 Chiamarlo (...metterci in contatto con il SERVER)

********** SERVER: Nel nostro caso un telefono nella casa **********
********** di chi vogliamo chiamare, capace di attendere **********
********** chiamate.... (oppure uno dei due programmi che **********
********** aspetta una connessione) **********

4 Comunicare

5 Terminare la comunicazione

Per quanto riguarda la programmazione dei socket quindi si necessita di 2 programmi:

1 SERVER: che aspetti connessioni su una determinata porta e che, se interpellato da un CLIENT, capace di interagire con lui.

2 CLIENT: il programma che si mettera' in contatto con il SERVER e che invii dati e che riceva la loro elaborazione del server

Per spiegare semplicemente un esempio di CLIENT e SERVER che comunemente avete sott'occhi, basta pensare al vostro Browser per internet.
Questo non fa altro che collegarsi alla porta 80 dell'indirizzo DNS che voi mettete.

********** DNS: Non e' altro con un altro modo per scrivere **********
********** l'indirizzo IP. Al posto di scrivere 212.234.3.54 **********
********** scrivete un indirizzo alfanumerico tipo: **********
********** www.spippolatori.it **********

********** Sta di fatto che www.spippolatori e 212.234.3.54 **********
********** e' assolutamente uguale! EX CLARO? **********
********** PS: 212.234.3.54 non e' l'IP del sito degli SPP :)**********

e mandare come dati:

GET /

a questo punto avrete il sorgente della pagina HTML del sito che vi interessa. Sara' poi compito del Browser visualizzarla secondo le direttive dei comandi HTML.

Quindi...

1 il SERVER (in questo caso non un vero e proprio programma ma un qualcosa di più complesso) attende connessioni all'IP del vostro sito sulla porta 80.

2 il CLIENT (il vostro BROWSER) si collega alla porta 80 e invia come dati la frase "GET /"

3 l'elaborazione del dato da parte del SERVER sara' la pagina HTML.

Se non ci credete aprite una sessione di TELNET con l'IP (numerico o alfanumerico) di un sito e scrivete "GET /". Vedrete!!

Un altro esempio di CLIENT e' il vostro programma per IRC (Mirc per Windoz o xchat per Linux sono i più famosi) e scoprirete che non fa altro che connettersi alla porta 6667 del server (ad esempio irc.azzurra.it) e inviare e ricevere dati.

Questo e' identico con telnet alla porta 6667 del solito server irc.azzurra.it Una volta collegato in CLIENT irc (o voi con telnet) vi registra con i comandi:

NICK vostro_nick

USER uzver b c:vostro_reale_nome

(PS: Questa e' una delle tante pratiche di inizializzazione di un utente)

Dopo di che con opportuni comandi si puo' comunicare. Ma questo non riguarda il tema di oggi!

Un'ennesimo esempio di comunicazione attraverso l'uso di socket e affini sono i trojan come BO o NETBUS.

Come funzionanto? Nulla di più semplice!

1 La vittima (consenziente o no :)) installa un programma SERVER sulla sua macchina il quale ascolta una determinata porta (Back Orifice 31337, NetBus 12345) in attesa di comunicazioni da parte di un CLIENT.
2 Il CLIENT (ovvero il programma in mano al "CATTIVO") si connette all'IP della vittima e alla porta in cui sa che il SERVER ascolta.
3 Invia opportuni comandi e aspetta che il server gli invii l'elaborato
4 Il Server ricevuti i dati compie sulla macchina della vittima quello che i comandi gli dicono di fare e poi manda il risultato delle operazioni al CLIENT
5 Il CLIENT stacca la comunicazione
6 Il SERVER si rimette in ascolto aspettando la prossima comunicazione da parte del CLIENT.

EX CLARO?

Direi che se non avete capito ancora il funzionamento di comunicazione SERVER - CLIENT allora e' grave :)

Ora parliamo della vera e propria programmazione!

SCELTA DEL LINGUAGGIO

Prima di buttarci a capifitto su un editor di testi per scrivere righe e righe di comandi, e' opportuno scegliere che tipo di linguaggio. Per fare questo bisogna vagliare una serie di punti centrali.

1 Che caratteristiche ha la macchina in cui vogliamo installare il SERVER o il CLIENT

E' infatti totalmente inutile programmare un SERVER in Visual Basic quando poi devo usare quel SERVER su una macchina *nix o Linux!

Uguamente inutile programmare un SERVER in TCL quando poi lo vogliamo piazzare su una macchina con Windozzolo! Esiste si' un programma capace di rendere eseguibili gli SCRIPT in TCL su piattaforme diverse da LINUX e simili, ma occupa 2,8MB! Sai che trojan!!

2 Utilizzo del rapporto SERVER - CLIENT

Questo vuol dire:
COSA VOGLIAMO FARGLI FARE A QUESTO SERVER o A QUESTO CLIENT?

Se vogliamo creare un Browser avremo certe esigenze, se vogliamo creare un CLIENT IRC altre, un TROJAN altre ancora! Dipende tutto da cio' che vogliamo fare!

3 Direi che non c'e' nessun terzo punto :)

METTIAMO A CONFRONTO LA PROGRAMMAZIONE DEI SOCKET NEI VARI LINGUAGGI:

IL C

Il C e' un linguaggio multipiattaforma (+ o -) se si regolano le varie librerie. E' snello e occupa poco. Programmare i socket in C vuol dire niente meno che creare un descrittore di file. Solo che, al posto di essere un FILE e' un SOCKET :)
Non e' facilissimo (ma nemmeno difficilissimo) creare un prog in C capace di

comunicare ed e' sicuramente la scelta migliore se si necessita di stabilita' e sicurezza (e' compilato). Di suo non e' grafico... poi e' tutto una questione di librerie

Il VISUAL BASIC

Marcatamente per WINDOZ. E' semplice, compilato, stabile. Non necessita di conoscenze approfondite sulla programmazione (anzi...). Il SOCKET (che per lui diventa "winsock") e' un qualcosa di gia' configurato a cui basta dare 2 o 3 direttive per essere brillantemente funzionante. E' grafico.

Il TCL

E' uno SCRIPT, cio' comporta che se uno avesse voglia, e capisse di programmazione... saprebbe capire cosa fa. E' leggero (sempre se escludiamo il fatto che necessita per win di un programma aggiuntivo bello massiccio). Il socket e' più o meno visto come il C, quindi come una variabile ben definita. Non e' grafico, ma lo puo' diventare con estrema semplicita' grazie ad un supporto grafico chiamato TK. Di solito girano abbinati con nome TCL/TK.

PREMESSA ALLA PARTE PRATICA: Nulla ci vieta di creare un server in C e un client in Visual Basic! Anzi! Spesso e' più congeniale se abbiamo esigenze particolari tipo 2 diversi sistemi operativi. Vedremo in seguito.

PRATICA (ovvero qualche esempio per confrontare meglio!):

RICORDO CHE QUEST'ARTICOLO NON È UN TUTORIAL SULLA PROGRAMMAZIONE QUINDI NON MI SOFFERMO SUI VARI COMANDI.

Incominciamo con il server in C:

Un server molto semplice in C capace di ascoltare una determinata porta e

capace di ascoltare una determinata porta (nell'esempio la 12345). E' programmato con librerie per LINUX, non so se le stessa valgono anche per win... per win io uso librerie diverse che poi vi indico.

------INIZIO----------

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <stdio.h>
//Queste solo le librerie necessarie per un server per LINUX

int creazione(int porta) {
  int sock,errore;
  struct sockaddr_in temp;

  // Creazione del SOCKET:
  sock=socket(AF_INET,SOCK_STREAM,0);

  //Specifico l'indirizzo:
  temp.sin_family=AF_INET;
  temp.sin_addr.s_addr=INADDR_ANY;
  temp.sin_port=htons(porta);

  //Imposto il SOCKET non bloccante:
  errore=fcntl(sock,F_SETFL,O_NONBLOCK);

  //Bind:
  errore=bind(sock, (struct sockaddr*) &temp,sizeof(temp));

  //Faccio accettare solo 3 chiamate:
  errore=listen(sock,3);

  return sock;
}
```

```
void chiusura(int sock) {

  //La chiusura del socket... rapida ed indolore!
  close(sock);
  return;
}

int main() {
  char buffer[512];
  int descrittore,nuovo;
  int exitCond=0;
  int Quanti;

  descrittore=creazione(12345);
  printf("Attendo connessioni...");

  //Questo ciclo continua fino a quando non avviene la connessione:
  while (!exitCond) {
        if ((nuovo=accept(descrittore,0,0)) !=-1) {

                //Lettura dati ricevuti:
                if ((Quanti=read(nuovo,buffer,sizeof(buffer)))<0) {
                        printf("Impossibile leggere il messaggio. \n");
                        chiusura(nuovo);
                } else {
                buffer[Quanti]=0;

                //Elaborazione dei dati:
                if (strcmp(buffer,"exit")==0)
                        exitCond=1;
                else printf("%s\n",buffer);
                }
                //Chiusura temporanea:
                chiusura(nuovo);
```

```
        }
  }

  //Chiusura definitiva del socket
  chiusura(descrittore);
  printf("Terminato.\n");

  return 0;
}
```

--------FINE----------

Una rapida spiegazione puo' essere questa:

```
int socket(int dominio, int tipo, int protocollo)
```

Questo crea il socket. Se da' errore pari a -1 azz non e' andato a buon fine la crezione del socket!

DOMINIO: indica il tipo di protocollo da utilizzare. Puo' essere:
        AF_UNIX (protocolli interni UNIX)
        AF_ISO (protocolli ISO)
        AF_INET (protocolli per INTERNET) --> quelli che servono a noi!

TIPO: indica il modo in cui avviene la comunicazione. Ce ne sono vari tipi. I
      più importanti sono:
      SOCK_STREAM connessione permanente e bidirezionale che si basa sul
                  flusso continuo di byte. E' probabilmente il più sicuro
      SOCK_DGRAM scambio di dati attraverso pacchetti di byte di lunghezza
                 massima fissata. Non e' affidabile al massimo.

PROTOCOLLO: indica il tipo di protocollo effettivo da utilizzare. Puo' essere
            nullo il valore in modo tale che venga usato quello di default
            regolato dal DOMINIO. Le altre possibilita' possono essere varie.
            Chi ha linux puo' andare a leggere il file: /etc/protocols,
            oppure leggere il mio :) :

```
---------------------------------------------------------------------------

ip      0       IP              # internet protocol, pseudo protocol
number icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # Internet Group Management
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
hmp     20      HMP             # host monitoring protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
rdp     27      RDP             # "reliable datagram" protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol class 4
xtp     36      XTP             # Xpress Tranfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp       39      IDPR-CMTP       # IDPR Control Message Transport
ipv6    41      IPv6            # IPv6
ipv6-route      43      IPv6-Route      # Routing Header for IPv6
ipv6-frag       44      IPv6-Frag       # Fragment Header for IPv6
ipv6-crypt      50      IPv6-Crypt      # Encryption Header for IPv6
ipv6-auth       51      IPv6-Auth       # Authentication Header for IPv6
ipv6-icmp       58      IPv6-ICMP       # ICMP for IPv6
ipv6-nonxt      59      IPv6-NoNxt      # No Next Header for IPv6
ipv6-opts       60      IPv6-Opts       # Destination Options for IPv6
rspf    73      RSPF            #Radio Shortest Path First.
vmtp    81      VMTP            # Versatile Message Transport
ospf    89      OSPFIGP         # Open Shortest Path First IGP
ipip    94      IPIP            # Yet Another IP encapsulation
encap   98      ENCAP           # Yet Another IP encapsulation

---------------------------------------------------------------------------
```

Necessario sapere e' che il socket deve essere non bloccato!
Comunque questo riguarda la programmazione... cosa che io non voglio toccare qui!

Il SERVER creato sopra attende un messaggio per stamparlo sul monitor. Ora mi mostro un client in C capace di comunicare con il socket di prima.

------INIZIO-----------

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void chiusura(int sock) {

        //semplice come bere un bicchiere d'acqua!
        close(sock);
        return;
}

int creazione(char* destinazione, int porta) {
        struct sockaddr_in temp;
        struct hostent *h;
        int sock;
        int errore;

        temp.sin_family=AF_INET;
        temp.sin_port=htons(porta);

        //gethostbyname indica l'IP con DNS
        h=gethostbyname(destinazione);
        if (h==0) {
                printf("IP alfanumerico fallito\n");
```

```
                exit(1);
        }
        bcopy(h->h_addr,&temp.sin_addr,h->h_length);

        //Creazione del socket:
        sock=socket(AF_INET,SOCK_STREAM,0);

        //Connessione del socket:
        errore=connect(sock, (struct sockaddr*) &temp, sizeof(temp));
        return sock;
}

void comunicazione(int sock, char* messaggio) {
        printf("Il messaggio inviato e' %s\n",messaggio);

        //Si usa write come se fosse un file!
        if (write(sock,messaggio, strlen(messaggio))<0) {
                printf("Impossibile comunicare col SERVER\n");
                chiusura(sock);
                exit(1);
        }
        printf("Messaggio spedito\n");
        return;
}

int main(int argc, char* argv[]) {
        int descrittore;

        //Creo e connetto:
        descrittore=creazione("127.0.0.1",12345);

        if((argc==2)&&(strcmp(argv[1],"exit")==0))
                comunicazione(descrittore,"exit");
        else
                comunicazione(descrittore,"Un semplice messaggio");
```

```
        chiusura(descrittore);

        return 0;
}
```

---------FINE----------

Come potete vedere il C anche qui non si smentisce! Tratta tutto come un file!
Io invio messaggi come se scrivessi sopra il file "SOCKET" e lo leggo con
read()!

Esiste anche un altro modo che si basa su SEND e RECV.

int send(int descrittore_socket, const void* buffer, in lunghezza, unsignet int opzioni)

ove
BUFFER contiene il messaggio. Deve avere una dimensione superiore alla
       lunghezza del messaggio.
LUNGHEZZA e' appunto la lunghezza del messaggio
OPZIONI mediamente si lascia 0.

Per RECV, praticamente uguale:

int recv(int descrittore_socket, const void* buffer, in dimensione_buffer, unsignet int opzioni)

Per quanto rigurada win
bisogna inserire questa libreria:

#include <winsock.h>

Per le piccole modifiche di stesura del codice... STUDIATE!

Data che io sono un bravo ragazzo vi posso dire che:

SERVER
sorgente -->   1521 byte
compilato -->   12826 byte

CLIENT
sorgente -->   1361 byte
compilato -->   13063 byte

questo per quanto riguarda il mio linux... win, "i don't know!" :)
ma penso non ci sia molta differenza.

Finito con il C, passiamo al VISUAL BASIC!

Per creare un server in VISUAL BASIC bisogna ricordarsi che comunque sara'
solo per Windozzolo.

La procedura e' abbastanza semplice:

1 Si crea un progetto Standard EXE
2 Si aggiunge il componente mswinsck.ocx che corrisponde a
  microsoft winsock controls 5.0
3 Si inserisca il winsocke nel form e lo si nomini tcpServer
4 Si inerisca un textBox e gli si dia nome arrivo
5 Si inserisca questo codice:

----INIZIO---------

```
Private Sub Form_Load()
    'imposto la porta
    tcpServer.LocalPort = 12345
    'imposto e lo metto in ascolto:
    tcpServer.Listen
End Sub
```

```
Private Sub tcpServer_ConnectionRequest _
(ByVal requestID As Long)
    If tcpServer.State <> sckClosed Then _
    tcpServer.Close
    tcpServer.Accept requestID
End Sub

Private Sub tcpServer_DataArrival _
(ByVal bytesTotal As Long)
    'imposto che i dati arrivati vengano mostrati nella textbox arrivo
    Dim strData As String
    tcpServer.GetData strData
    arrivo.Text = strData
End Sub
```

----FINE--------

Semplicissimo ma efficace!

Possiamo vedere come qui il SOCKET non e' toccato... e' un'oggetto a se stante che va solo indirizzato alla porta (tcpServer.LocalPort = 12345) e ad ascoltare (tcpServer.Listen).

Ma si puo' dire certo se questo e' un vantaggio o uno svantaggio? Sicuramente no! Poiche' tutto ruota in base alle finalita'! Questo va bene per semplici comunicazioni o un semplice trojan, ma forse per cose relealemtne più tecniche il C da più liberta' a scapito forse della leggerezza di codice e di semplicita'.

In VB creare un banale SERVER vuol dire 10 righe (max), in C abbiamo visto prima. Più di 30! Il visual Basic lo porto il linux e non ho possibilita' di utilizzo. Il C basta cambiare 2 righe e tutto diventa uguale sia in win che in linux.

Ora vediamo un client capace di mandare un messaggio, sempre in visual BASIC:

Per creare un client si faccia:

1 Si apra sempre uno Standard EXE.
2 Si inserisca sempre un mswinsck.ocx come prima e gli si dia il nome
  tclClient
3 Si aggiunga un textBox e gli si dia il nome messaggio
4 Si metta un CommandButton e gli si dia il nome connetti
5 Si aggiunga il seguente codice:

-----INIZIO-------

```
Private Sub Form_Load()
    tcpClient.RemoteHost = "127.0.0.1"
    tcpClient.RemotePort = 12345
End Sub

Private Sub connetti_Click()
    tcpClient.Connect
End Sub

Private Sub txtSendData_Change()
    tcpClient.SendData messaggio.Text
End Sub
```

-------FINE-----

Il client e' ancor più inpressionante come semplitica'. Due comandi indicano porta e Host (tcpClient.RemoteHost = "127.0.0.1" e tcpClient.RemotePort = 12345) e il resto sono 2 righe: il pulsante che avvia la comunicazione (tcpClient.Connect) e l'invio dei dati (tcpClient.SendData messaggio.Text).

Tutto questo con valori pressoche' identici al C:

SERVER:
sorgente --> form1.FRM: 1.525 byte / progetto.VBP 651

eseguibile --> 10.240 byte

CLIENT:
sorgente --> form1.FRM: 1.460 / progetto.VBP 651
eseguibile --> 10.240 byte

Non manca che un esempio di socket basato su script. Ho preso come vi ho gia' detto il TCL, ma di per se' e' uguale prendere il PERL o altri.

Cosa c'e' gia' da dire in anticipo:

1 Non e' compilato! Quindi basta che qualcuno faccia un bel "vi nome_file.tcl" che vede il codice sorgente. Voi direte che non e' nulla! Ma sappiate che la presenza di bug e' dietro l'angolo!

C'e' sempre la possibilita' di rischiare di dare la propria macchina nelle mani di uno sconosciuto hacker e se c'e' un SERVER in TCL, o la programmazione e' iper sicura o si rischia di fare danni.
ESEMPIO: un server che esegua determinati comani di una shell e poi dia il risultato al CLIENT. Noi siamo sicuri perche' il CLIENT da per far suo, solo determinati comandi e (essendo noi molto ingenui) non ipotiziamo che qualcuno possa fare altro da quello concesso dal CLIENT. Arriva un hacker che mi entra nella macchina come untente generico e vuole cercare un bug. Trova questo server! Un conto e' trovarlo compilato e un conto trovarlo come script. Se e' compilato e' possibile che ci rinunci e non si metta a ricercare il sorgente (dubito che si metta pure a decompilarlo!). Ma se e' uno script e conosce un minimo il TCL, capisce al volo come funziona il gioco e se e' capace puo' crearsi un CLIENT e diventare ROOT di quella macchina.
EX CLARO l'esempio (particolarmente stupido)?

2 Non e' grafico ma ci vuole molto poco a farlo diventare :)
Questo e' un profondo vantaggio! Poiche' e' possibile creare un SERVER non grafico ma un client programmato nello stesso linguaggio grafico!

3 E' particolamente versatile! Non prevete le strette norme del C, ma allo

stesso tempo e' configurabile quasi come lui. Non e' cosi' semplice come il VB ma e' capace di fare altrettanto!

4 E' multipiattaforma, cio' vuol dire che sul mio linux mi creo un CLIENT per IRC e lo posso usare col relativo supporto anche in WINDOWS

5 E' usato anche come script per INTERNET (come il PERL e altri).

6 Comunica solo in TCP/IP. Non comunica ancora in UDP.

Tutto cio' va messo su una bilancia! Vanno fatte giuste considerazioni e poi si deve scegliere cosa e' meglio per le nostre esigenze.

Facciamo quindi subito un esempio di un SERVER in TCL:

--------INIZIO--------

```
#!/usr/bin/tcl

#variabile per la porta:
set porta 12345

proc esamina {sock} {

set l [gets $sock]

#questa procedura legge in socket
#e se non trova il comando EOF
#cioe' di fine, stampa:

if {[eof $sock]} {
close $sock
} else {

#Stampa il tutto:
```

```
puts $l
}
}

proc accetto {sock addr port} {

#regola gli enventi del socket:
fileevent $sock readable [list esamina $sock]

#Lo configura non bloccato (vedi C):
fconfigure $sock -buffering line -blocking 0

}

#crea il socket e aspetta gli eventi:
socket -server accetto $porta
vwait events

---------FINE---------
```

Cosa c'e' da dire? E' la via di mezzo fra uno e l'altro esempio. Dico una via di mezzo perche':

1 il socket non e' qualcosa di pre confezionato (creazione del socket: socket -server accetto $porta, configurazione: fconfigure $sock -buffering line -blocking 0)
2 Non e' quella serie di comandi stani per regolare le porte e simili. Ho messo una variabile contenete la porta e gliel'ho fatta leggere (potevo pero' anche non farlo).
3 E' semplice e stabile ed e' testuale. Avrei potuto farlo diventare grafico cosi':

```
---------INIZIO---------

#!/usr/bin/wish
```

```
#possiamo vedere che non e' più TCL ma WISH, poiche' voglio la grafica
#e quindi il supporto TK

#creo un'etichetta:
label .a -text "Attendo connessioni..."
pack .a

set porta 12345

proc esamina {sock} {
set l [gets $sock]
if {[eof $sock]} {
close $sock
} else {

#modifico l'etichetta di prima con una nuova scritta:
.a configure -text "Il messaggio trasmesso e'\n$l"
}
}

proc accetto {sock addr port} {

fileevent $sock readable [list esamina $sock]

fconfigure $sock -buffering line -blocking 0

.a configure -text "Connessione avvenuta..."
}

socket -server accetto $porta
vwait events

----------FINE----------
```

Una grafica scarna solo per necessita', ma e' possibile creare applicazioni della stessa complessita' del VISUAL BASIC.
Analizziamo ora il CLIENT in TCL:

--------INIZIO----------

```
#!/usr/bin/tcl

set s [socket 127.0.0.1 12345]
fconfigure $s -buffering line
puts $s "Un semplicissimo messaggio"
```

--------FINE-----------

Mi direte: ma e' più corto di VB! Si' e' vero solo perche' non ho messo l'opzione
di scelta del messaggio come avevo fatto in visual basic. Se cosi' avessi fatto sarebbe diventato questo:

-------INIZIO-----------

```
#!/usr/bin/wish

label .messaggio -title "Inserisci nello spazio sottostante\nun messaggio da\
                         inviare al server"

entry .spazio -textvariable msg

button .invia -text "ok" -command spedisci
pack .messaggio .spazio .invia

proc spedisci {} {
global $msg

set s [socket 127.0.0.1 12345]
fconfigure $s -buffering line
```

```
puts $s $msg

}

------FINE--------------
```

Semplice no?
Vi ricordo che tutti gli esempi qui riportati sono solo base. Gia' in queso avrei dovuto fare un controllo della presenza del messaggio, poi avrei dovuto mettere un titolo alla finestra e simili. Ma queste sono solo sottigliezze.

SERVER:
testuale --> 298 byte
grafico --> 606 byte

CLIENT:
testuale --> 114 byte
grafico --> 342 byte

I programmi più piccoli mai esistiti sulla faccia della terra!

Mettiamo a confronto i 4 programmi:

	C	VISUAL BASIC	TCL	TCL/TK
SERVER sorgente	1521	1525+651	298	606
SERVER compilato	12826	10.240		
CLIENT sorgente	1361	1460+651	114	342
CLIENT compilato	13063	10240		

Possiamo quindi tranquillamente affermare che se il nostro problema e' lo spazio il TCL e' il massimo.
Se il nostro problema e' la grafica possiamo optare per il VB o il TCL/TK
Per quanto riguarda altri aspetti... spero di essere stato abbastanza chiaro!
Se il nostro problema e' la sicurezza gia' sappiamo!

La mia panoramica sui SOCKET direi che finisce qui! Ho esaminato tre linguaggi che mi sembrano esempi di 3 tipologie distinte di linguaggi mostrandone i pregi e i difetti.

Per aiuti, critiche o commenti sappiate che io sono reperibile sempre al mio indirizzo email! Ciao e buona programmazione!

----------FINE----------

Michael Bakunin
<bakunin@meganmail.com>