# Um Tutorial sobre Sockets – Parte I

*Por Antonio Marcelo*

## Iniciando

As grandes ferramentas utilizadas por especialistas de segurança, hackers e crakers tem como base a linguagem C ANSI ou C ++. Muitos dos scanners, sniffers, backdoors, etc. , exploram um recurso muito conhecido na programação-cliente servidor : os sockets.

Um socket é nada mais nada menos que um programa, ou rotinas de programas que permitem a comunicação, interligação e troca de dados entre aplicações. Por exemplo quando é feita uma conexão de FTP, um socket é estabelecido entre a origem e o destino. Até mesmo os famosos exploits utilizam sockets para estabelecer comunicação.

Nós iremos explorar nestes nossos tutoriais os mais variados tipos de socketes, inclusive o *RAW SOCKETS*, que é o mais interessante de todos.

O que você precisa para começar :

a) Compilador C – Iremos explorar nosso tutorial em ambiente Linux e por isso utilizaremos o compilador GCC. Esta decisão foi tomada por porque o GNU Linux além de ser um sistema gratuito é o mais utilizado e explorados pelos especialistas de segurança para o desenvolvimento de ferramentas.
b) Uma rede com TCP/IP – Apesar de ser um *acessório* importante, podemos simular com um micro com uma placa de rede um ambiente d etrabalho.
c) Sistema Operacional Linux – Por ser robusto, confiável e ter tudo para o desenvolvimento de aplicações baseadas em sockets.
d) Paciência e perseverança – Isto é muito importante, pois não se aprende do dia para noite.

## Primeiros Passos :

Basicamente um socket pode ser declarado mediante três headers básicos :

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

Estes três headers permitem que utilizemos as funções  para a montagem de uma conexão. A definição de um socket é feita da seguinte maneira em C :

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

main(){
```

```
int e_socket;
...
}
```

Com isto começamos o nosso trabalho.Vamos começar utilizando os dois tipos de sockets, mais utilizados em aplicações, baseados no o protocolo TCP (Stream Sockets) e os que utilizam o protocolo UDP (Datagram Sockets). Estes sockets também são conhecidos como "SOCK_STREAM" e "SOCK_DGRAM", respectivamente.

A estrutura padrão em C de um socket pode ser definida da seguinte maneira :

```
struct sockaddr_in {
        short int sin_family;
        unsigned short int sin_port;
        struct in_addr sin_addr;
        unsigned char sin_zero[8];
        }
```

Cada item destas linhas possuem uma característica importante, são elas :

`short int sin_family;` - Tipo de família do socket, sendo que os padrões mais comuns seriam os seguintes :

a) AF_INET - ARPA INTERNET PROTOCOLS
b) AF_UNIX - UNIX INTERNET PROTOCOLS
c) AF_ISSO - ISO PROTOCOLS
d) AF_NS - XEROX NETWORK SYSTEM PROTOCOLS

`unsigned short int sin_port;` - Número da porta TCP ou UDP a ser utilizada para a comunicação dos programas.

`struct in_addr sin_addr;` - Endereço IP do host destino. Pode ser colocado de maneira direta ou por uma entrada d edados.

`unsigned char sin_zero[8];` - Zera a estrutura do socket. Vamos ver mais a fernte isto.

A declaração do socket é feita da seguinte maneira :

e_socket = socket(sin_family, tipo_do_socket_desejado,número_do_protocolo);

Traduzindo para o C ANSI ficaria assim :

e_socket = socket(AF_INET,SOCK_STREAM,0)

Onde o 0 é o número do protocolo e pode ser substituído pelo seguinte :

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

Vamos a um exemplo mais completo agora :

```
main(){
int e_socket;
struct sockaddr_in destino;

e_socket = socket(AF_INET,SOCK_STREAM,0);
if(e_socket < 0)
{
perror("Socket");
exit(1);
}
destino.sin_family = AF_INET;
destino.sin_port = htons(2048);
destino.sin_addr.s_addr = inet_addr("10.0.0.1");
bzero(&(destino.sin_zero),8);
...
}
```

Nosso programa está começando a ser delineado e para finalizarmos esta primeira parte falaremos de uma função básica para finallizar nossa primeira parte do tutorial.

**A Função `CONNETC()`**

Eis a função responsável por executar a conexão em uma porta propriamente dita. Quando um programa vai se comunicar com outro a função `connect()` do socket é utilizada para testar a conexão e iniciar o *processo de comunicação*.

O protótipo da função é o seguinte :

int connect(socket,(struct sockaddr * )&destino, sizeof(destino));

E agora o nosso programa ficará o seguinte com esta função :

```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>


int e_socket;
struct sockaddr_in destino;
int conexao;

main()
{
```

```
e_socket = socket(AF_INET,SOCK_STREAM,0);
if(e_socket < 0)
 {
  perror("ERRO !");
  exit(1);
 }

destino.sin_family = AF_INET;
destino.sin_port = htons(22);
destino.sin_addr.s_addr = inet_addr("10.0.0.20");
bzero(&(destino.sin_zero),8);

conexao = connect(e_socket,(struct sockaddr * )&destino,
sizeof(destino));
if(conexao < 0) {
    perror("Porta fechada !\n");
    close(e_socket);
    exit(1);
  }
   printf("A PORTA 22 DO SSH ESTA ABERTA !\n");
   close(e_socket);
  }
```

Eis o nosso programa que testa se aporta 22 está aberta. Ele funciona da seguinte maneira :

```
int e_socket;
struct sockaddr_in destino;
int conexao;
```

Declaração das variáveis do sockets.

```
e_socket = socket(AF_INET,SOCK_STREAM,0);
if(e_socket < 0)
 {
  perror("ERRO !");
  exit(1);
 }
```

Em seguida vamos declarar um socket do tipo TCP (SOCK_STREAM) e testamos se as funções de sockets estão ativas .

```
if(e_socket < 0)
 {
  perror("ERRO !");
  exit(1);
 }
```

Neste ponto declaramos  o tipo de socket (AF_INET) a porta que quermos testar se está aberta (destino.sin_port = htons(22);) o endereço do host que quermos testar (destino.sin_addr.s_addr = inet_addr("10.0.0.20");) e zeramos a estrutura (`bzero(&(destino.sin_zero),8);`)

```
destino.sin_family = AF_INET;
destino.sin_port = htons(22);
destino.sin_addr.s_addr = inet_addr("10.0.0.20");
bzero(&(destino.sin_zero),8);
```

E no final do programa, testamos se a conexão está ativa ou não, utilizando a função CONNECT().

```
conexao = connect(e_socket,(struct sockaddr * )&destino,
sizeof(destino));
if(conexao < 0) {
    perror("Porta fechada !\n");
    close(e_socket);
    exit(1);
  }
   printf("A PORTA 22 DO SSH ESTA ABERTA !\n");
   close(e_socket);
  }
```

Vamos compilar o programa para testarmos, no prompt de seu Linux digite :

```
oldmbox# gcc –o ex1 ex1.c
```

Com o programa compilado digite :

```
oldmbox# ./ex1
```

Se sua porta 22 estiver aberta, a resposta será o seguinte :

```
oldmbox# A PORTA 22 DO SSH ESTA ABERTA !
```

Caso contrário a resposta será negativa.

## Considerações finais :

Com estes modestos passos iniciais podemos já começarmos a especular algumas coisas. Este programa acima é o princípio **muito remoto** de um scanner TCP de portas, pois estamos testando se a porta 22 está ativa ou não. Podemos propor o seguinte : criar um scanner TCP que teste um range de portas ( por exemplo de 1 – 1080). Num segundo momento, escrever o resultado em um arquivo.

Na próxima lição de nosso tutorial, iremos explorar novas funções e escrever um scanner de portas !!!! Até a próxima.

Antonio Marcelo é especialista em segurança e diretor de tecnologia e negócios da empresa BufferOverflow Informática (http://www.bufferoverflow.com.br). É autor de 4 livros sobre Linux, entre eles Linux Ferramentas Anti hackers, publicado pela editora Brasport. Seu email de contato é amarcelo@bufferoverflow.com.br.