Reconhecimento/Enumeração

Implementação de um port scanner em C

Henrique Hepp
Universidade Federal do Paraná
hhepp@inf.ufpr.br

ABSTRACT

Esse trabalho relata a implementação de um port scanner em C, os testes realizados e seus resultados. O port scanner funciona apenas para endereços IPv4 e identifica apenas serviços em portas que usam o protocolo TCP.

Keywords

Port Scanner, C

1. INTRODUÇÃO

Nesse trabalho foi implementado um port scanner de um servidor com endereço IPv4 que identifica serviços com o protocolo TCP. Existem duas principais formas de escanear uma porta TCP: O SYN scan e connect scan. O SYN scan é o mais popular, nesse caso é feita uma "meia-conexão" entre o cliente e o servidor, envia-se um pacote SYN ao servidor e sendo recebida uma resposta a conexão é fechada. Já no caso do connect scan, estabelece-se uma conexão completa entre o cliente e o servidor usando o system call connect().[1]

Nesse trabalho, foi implementada a segunda opção. O programa, escrito na linguagem c, cria um socket e usa a função connect para conecta-lo ao serviço em uma porta do servidor. Se houve conexão estabelecida, o programa aguarda a primeira mensagem do serviço que geralmente contém a sua identificação e a imprime, caso não receba a mensagem, constata-se apenas que a porta está aberta.

Nas próximas seções serão vistos o código implementado, os testes feitos e os resultados obtidos.

2. O CÓDIGO FONTE

O programa pode ser dividido em duas partes, a primeira é a preparatória, que lê os argumentos, os ips e as portas que devem ser escaneadas; a segunda, que cria o socket, faz a conexão e identifica o serviço.

A primeira parte, lê os argumentos: endereço IPv4 (ou faixa separada por "-") e, opcionalmente, uma porta (ou intervalo separado por "-"). Por exemplo, o endereço 192.168.1.1-10 e as portas 1-1023. Após a leitura, copia-se a primeira parte, até o terceiro ponto, do endereço IP para uma string "IP", os números após o terceiro ponto são copiadas para duas variáveis "ipIn" e "ipFin", se foi usado apenas um IP no argumento, "ipFin" será igual a "ipIn". As portas também são copiadas para duas variáveis "portIn" e "portFin", se elas não forem declaradas o programa assume que elas correspondem às portas 1 e 65535, que são a menor e a maior portas possível. Essas variáveis são usados em dois fors (veja o código abaixo) que iteram todos os ips e portas a serem escaneados pela segunda parte do programa.

```
for (i=ipIn;i<=ipFin;i++){
        //forma uma string com o endereco
            IP completo
        sprintf(endIP,"%s%d",IP,i);
        for (j=portIn;j<=portFin;j++){
                segunda parte, escaneia
                    cada ip e porta
        }
}
```

A segunda parte, basicamente, cria um socket, tenta fazer a conexão com o IP e a porta, caso a conexão sejá estabelecida espera-se uma mensagem do serviço com sua descrição. Ao serem realizados os testes constatou-se para a última parte a necessidade de estabelecer um tempo limite para receber a mensagem, pois há serviços que não enviam nennhuma descrição. Por último fecha-se o socket.

Para criar o socket usa-se:

```
sock = socket(AF_INET , SOCK_STREAM , 0);
//AF_INET == endereco IPv4
//SOCK_STREAM == conexao tipo TCP
// 0 == protocolo padrao do socket, nesse
    caso, o IP
```

Para se conectar ao IP usa-se a função "connect", como entrada ela recebe uma struct que contém o IP e a porta:

```
//cria a struct
struct sockaddr_in servidor;
//coloca o endereco do IP na struct
servidor.sin_addr.s_addr = inet_addr(endIP)
//declara ser o endereco tipo IPv4
servidor.sin_family = AF_INET;

//coloca a porta na struct
servidor.sin_port = htons(j);

//conecta com o servidor
```

ACM ISBN xxx-xxxx-xx-xxx.
DOI: xx.xxxx/xxxxx

```
c = connect(sock,(struct sockaddr *)&
    servidor,sizeof(servidor));
```

Caso a conexão suceda, a porta está aberta, para saber qual serviço está usando essa porta, espera-se uma mensagem do serviço com sua identificação. Espera-se receber essa mensagem por 60 segundos, se houver ela é impremida, caso contrário, apenas a porta é impressa. Por último, fecha-se o socket.

```
//estabelece um time out para receber uma
    mensagem do servidor
struct timeval tv;
tv2.tv_sec = 60;
tv2.tv_usec = 0;
setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (
    char *)&tv,sizeof(struct timeval));

//le a mensagem enviada pelo servico
read(sock,buffer,sizeof(buffer));
printf("%s\t%d\t%s",endIP,j,buffer);

close(sock);
```

3. TESTES E RESULTADOS

Para testar o programa implementado foram escaneados os IPs 200.238.144.20-29 no dia 1/9/2016, para verificar se os testes retornavam valores corretos, também foi feito um escaneamento desses IPs com o programa Nmap. Nesse dia apenas os IPs 200.238.144.27-29 estavam disponíveis. Os resultados do escaneamento desses IPs seguem abaixo:

Saída do Nmap:

Starting Nmap 7.01 (https://nmap.org) at 2016-09-01 15:56 BRT
Nmap scan report for 200.238.144.27
Host is up (0.032s latency).
Not shown: 992 closed ports
PORT STATE SERVICE
21/tcp open ftp
22/tcp open ssh
25/tcp filtered smtp
111/tcp open rpcbind
135/tcp filtered msrpc
139/tcp filtered netbios-ssn
445/tcp filtered microsoft-ds
32768/tcp open filenet-tms

Starting Nmap 7.01 (https://nmap.org) at 2016-09-01 15:56 BRT
Nmap scan report for 200.238.144.28
Host is up (0.040s latency).
Not shown: 993 closed ports
PORT STATE SERVICE
22/tcp open ssh
25/tcp filtered smtp
135/tcp filtered msrpc
139/tcp filtered netbios-ssn
445/tcp filtered microsoft-ds
5001/tcp open commplex-link
8080/tcp open http-proxy

Starting Nmap 7.01 (https://nmap.org) at 2016-09-01 15:56 BRT
Nmap scan report for 200.238.144.29
Host is up (0.041s latency).
Not shown: 987 closed ports
PORT STATE SERVICE
21/tcp open ftp
22/tcp open ssh
25/tcp filtered smtp
42/tcp open nameserver
80/tcp open http
135/tcp filtered msrpc
139/tcp filtered netbios-ssn
443/tcp open https
445/tcp filtered microsoft-ds
1433/tcp open ms-sql-s
3306/tcp open mysql
5060/tcp open sip
5061/tcp open sip-tls

Saída do programa implementado:

Varredura iniciada em Thu 01 Sep 2016 22:30:26 BRT
IP: 200.238.144.27
Portas: 1-65535
200.238.144.27 21 220 redhood FTP server (Version wu-2.6.1-18) ready.
200.238.144.27 22 SSH-1.99-OpenSSH 2.9p2
200.238.144.27 111
200.238.144.27 32768

Varredura iniciada em Thu 01 Sep 2016 19:05:27 BRT
IP: 200.238.144.28
Portas: 1-65535
200.238.144.28 22 SSH-2.0-OpenSSH 6.6.1p1 Ubuntu-2ubuntu2.6
200.238.144.28 5001
200.238.144.28 8080

Varredura iniciada em Thu 01 Sep 2016 18:32:02 BRT
IP: 200.238.144.29
Portas: 1-65535
200.238.144.29 21 220 Welcome to the ftp service
200.238.144.29 22 SSH-2.0-OpenSSH 5.9p1 Debian-5ubuntu1.9
200.238.144.29 42
200.238.144.29 80
200.238.144.29 443
200.238.144.29 1433
200.238.144.29 3306
200.238.144.29 5060
200.238.144.29 5061

Comparando-se ambas as saída pode-se verificar que todas as portas abertas de acordo com o Nmap foram reconhecidas pelo programa implementado. Os serviços SSH e FTP enviaram mensagens, de modo que puderam ser identificados. Com as demais portas, não foram recebidas mensagens de identificação, precisaria-se enviar mensagens para essas portas para tentar identificá-las, o que não foi implementado.

4. CONCLUSÃO

Nesse trabalho foi implementado um port scanner de um servidor com endereço IPv4 que identifica serviços com o protocolo TCP. O programa implementado na linguagem c usa a função connect que conecta o socket com o serviço em uma porta do servidor. Se houve conexão estabelecida, o programa aguarda a primeira mensagem do serviço que é a

sua identificação e a imprime, caso não receba a mensagem apenas é constatado que a porta está aberta. Foram feitos alguns testes que foram comparados com um escaneamento feito pelo programa nmap, constatou-se que o programa implementado reconheceu corretamente as portas e os serviços disponíveis.

Verificou-se, também, que o escaneamento das portas é bastante lento, em torno de meia hora para escanear todas as portas possíveis de um endereço IP, por isso sugere-se implementar futuramente um port scanner que tenha um time out mais rápido ao tentar estabelecer uma conexão com alguma porta.

5. REFERENCES

[1] G. F. Lyon. *Nmap Network Scanning*, 2009.