Portscanner

Mirko Bez Simon Targa

January 6, 2016

Contents

Introduction

The aim of this project was to implement a port scanner. The scanner should be written in C and support various scan methods (e.g. TCP connect scan, TCP SYN scan ...). The final result is a program that tries to simulate the behavior of nmap, which is one of the most used programs for port scanning. This document describes how the program and the implemented scan methods work. The focus of the first section is on the TCP connect scan which is the most simple port scan technique. The second section is about how the TCP SYN scan works and how it was implemented within the scope of this project. Chapter three describes the scan methods Xmas, TCP NULL and Fin scan and their implementations. The fourth and final section is dedicated on how port scanning attempts can be detected and blocked by an IT administrator.

1 TCP connect scan

1.1 Theory

The TCP connect scan is probably the most easy method to scan for open ports. It simply takes advantage of the system call *connect* of the underlying operating system, in order to establish a connection with the target machine and port. Afterwards the returned value of the system call is used to determine if the port to check is either closed or open at the target machine [1].

Extended description with details. Add advantages and disadvantages (table?)

1.2 Details of implementation

In order to use the system call connect the implementation uses C sockets of the type SOCK_STREAM. This type of socket allows us, to establish a tcp connection to the target machine.

```c
int mysocket;
mysocket=socket(AF_INET, SOCK_STREAM, 0);
```

Listing 1: C code to create a tcp socket in C

Additionally to the socket we also have to use a structure of the type sockaddr_in to connect to the target machine and port. The structure is needed to define the ip address of the target machine and the port to use for the connection. The listing shows the code of how to assign the ip address and port to a structure of the type sockaddr_in.

```c
struct sockaddr_in server;
struct hostent *hostname;
hostname = gethostbyname(p->host_name);
memcpy( (char *)&server.sin_addr, hostname->h_addr_list[0], hostname->h_leng
server.sin_family = AF_INET;
server.sin_port = htons(port);
```

Listing 2: C code to use the structure sockaddr_in

The last step is to use the created socket and sockaddr_in structure to connect to the target machine and port. If the connection could be established we know that the port is open. To check if the connection attempt was successful, we only have to check the return value of the connect() function. Upon successful completion, connect() shall return 0. The code to use the connect() function is shown in the listing.

```
if(connect(mysocket, (struct sockaddr *)&server, sizeof(server))>=0){
  printf("TCP - Port %d is open\n", i);
  close(mysocket);
  mysocket = socket(AF_INET, SOCK_STREAM, 0);
}
```

Listing 3: C code to use the connect() to check if port is open

2 TCP SYN scan

2.1 Theory

2.2 Details of implementation

In order to only send a syn request instead of open a full tcp connection (including handshake) the implementation uses raw sockets. Raw sockets allow to control every section of the packets that will be sent. The function socket(), as shown in listing, can be used to create a raw socket that uses the tcp protocol.

```
  int mysocket;
  mysocket=socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
```

Listing 4: C code to use the connect() to check if port is open

Before we can send a syn request to the target machine, we have to build the packet to be sent. To send packets with a raw socket the function sendto() is used. It's second parameter is a pointer to the message to be sent, which is the packet that we build. It consists of the tcp theader, the ip header and the data to be sent. As we only want to send a syn request we don't care about the data, therefore it is empty. The C code of listing is used to initialize a pointer to the message to be sent, with empty ip and tcp header.

```
//Datagram to represent the packet
char datagram[4096];

//IP header
struct iphdr *iph = (struct iphdr *) datagram;

//TCP header
struct tcphdr *tcph = (struct tcphdr *) (datagram + sizeof (struct ip));
```

Listing 5: C code to use the connect() to check if port is open

Afterwards the ip header must be filled in. We don't need optional fields therefore the we use the minimal size possible size of the ip header which is 160 Bits (5*32 Bits). We use the ip version 4, which is still the most widely used ip version. The length of our packet is the sum of the length of the ip header and the length of the tcp header. For the time to live we choose 64, which should be big enough fur our purpose. As transfer protocol we set the tcp protocol. The source address of the ip header is set to the ip address of the scanning system and the destination address is set to the address of the target system to scan. To have a complete ip header we also have to calculate its check sum.

```
//Fill in the IP Header
iph->ihl = 5;
iph->version = 4;
iph->tos = 0;
iph->tot_len = sizeof (struct ip) + sizeof (struct tcphdr);
iph->id = htons (54321); //Id of this packet
iph->frag_off = htons(16384);
iph->ttl = 64;
iph->protocol = IPPROTO_TCP;
iph->saddr = inet_addr ( source_ip );
iph->daddr = dest_ip.s_addr;
iph->check = csum(datagram, iph->tot_len >>1);
```

Listing 6: C code to fill in ip header

Before the packet can be sent, we also need to fill in the tcp header. In order to send a syn request we only set the syn flag to true and all the other flags to false.

```
tcph->fin=0;
tcph->syn=1;
tcph->rst=0;
tcph->psh=0;
tcph->ack=0;
tcph->urg=0;
```

Listing 7: C code to set flags in tcp header

The last step before we can send the packet is to set the destination port (the port to scan) in the tcp header and calculate its check sum.

```
tcph->dest = htons ( port );
tcph->check = csum(&psh, sizeof(struct pseudo_header))
```

Listing 8: C code to set port and calculate checksum in tcp header

To function sendto() is used to send the created packet to the target machine and port. If the sending fails the program terminates with an error, because then we cannot scan for open ports.

```
//Send the packet
if(sendto(s, datagram, packetsize, 0 , &dest, destsize)< 0)
{
perror("Error sending packet: ");
exit(0);
}
```

Listing 9: C code to set port and calculate checksum in tcp header

3 XMAS, TCP NULL and FIN scan

3.1 Theory

3.2 Details of implementation

4 Port Scan Detectors

References

[1] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009. ISBN: 0979958717. URL: http://https://nmap.org/book/.