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.

```
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.

```
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

Continue to describe implementation

2 TCP SYN scan

2.1 Theory

2.2 Details of implementation

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/.