Protocol Design for TCP Port Scanning

Abdullah

June 12, 2021

Contents

0.1 Introduction

TCP is Transport layer connection oriented protocol, which is well known for its reliable data delivery.

This report includes a protocol design for scanning TCP ports on a server. Any number of TCP sockets running on a server/target device tells us the number of services being provided by the system. Our goal is to determine the number of TCP ports on a targeted machine. There are different methods involved in TCP port scanning, which tries to exploit the loopholes in TCP RFC but apart from those well known scanning techniques, we defined our own messaging protocol and response analysis to determine those running ports.

0.2 Functional Requirements

Requirements are mentioned below:

- Checking the status of Host (Live or not).
- Number of Ports running on the Host.
- Scan may be a single port or sequential.
- Bypass the Firewall (if any)

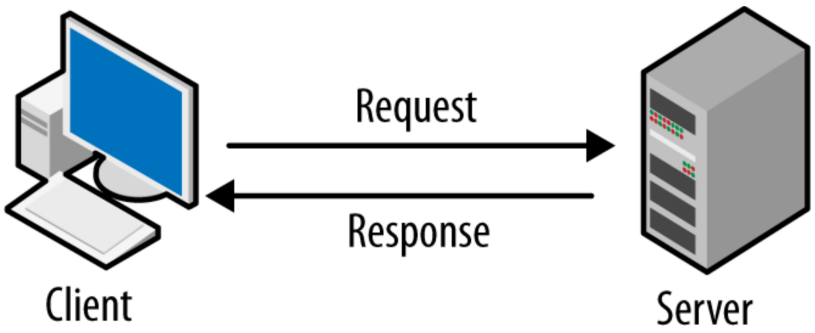
0.3 Nonfunctional Requirements

Requirements are mentioned below:

- Type of Services running on the host.
- Time Taken in Scanning.

0.4 Application Structure

The Application structure will be a client-server-based. Scanner will act as a client, while targeted machine will act as a sever.



Way of communication is described below

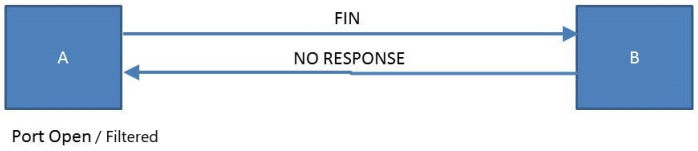
- Client port scanning application will send a TCP segment to a targeted machine.
- Server will respond to the client with an appropriate response message by looking request message and TCP RFC rules.

One thing to note that, we have nothing to do with data delivery, instead our work is just limited to connection making phase. That is why, there will not be any payload(data) attached to requesting segment.

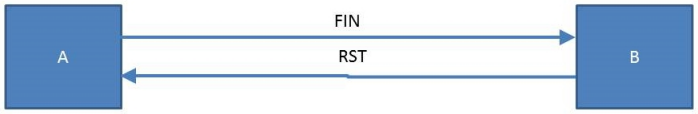
0.5 The Protocol

0.5.1 Message Types

These will be Request and Response messages, Below is the sequence of them:

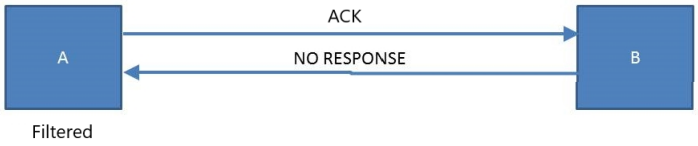


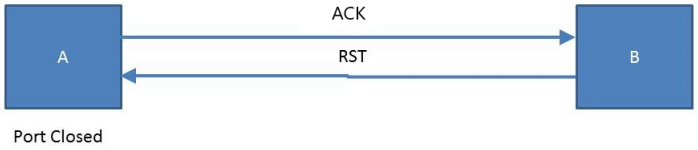
Port Open / Filtered



Port Closed

- Client will send TCP segments with only ”FIN flag set” to different ports on the server.
- Server will send the responses of the ports to client. (If not, then first packet will be send again)
- Server will send either TCP segment with ”RST flag set” means closed port or No response means either be opened or filtered.
- Client them send another TCP segment with ”ACK flag set” to same ports on the server.
- Server will respond with the ”RST flag set” means port is closed or No response means port is filtered.(If not, then second packet will be send again)
- Now, its time for Analysis part at client side.
- As we have two responses of each port, we will subtract the second response from the first-one and the result would be open ports.
- Now, we will match the port number of ’open ports’ in our data base and will fetch the corresponding service name to our application.
- In the mean time, we also noted the total request/respond time and that will be displayed in the client application.





For Example, from first response (FIN), we got 5 open/filtered ports. from second(ACK), we got 2 filtered ports. Since (ACK) provides us all filtered ports so we will remove the ambiguity of first response by subtraction and end-up with 3 open ports.

One thing to be Noted that, we are not implementing any protocol for checking the status of host (live or not) because our main concern is Port Scanning so for this, we will use an existing protocol ICMP. Its responses (Ping) will lets us know the aliveness of host.

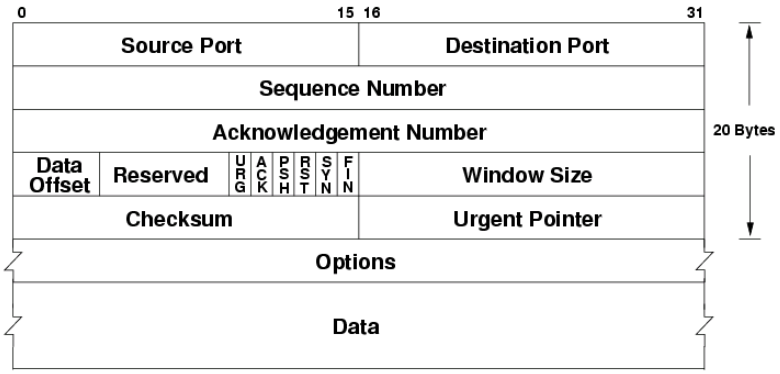
0.5.2 Message Format

Our Protocol is a binary protocol because our packet major portion is TCP segment which utilizes all values of bytes, as opposed to a text-based protocol, which only uses values corresponding to human-readable characters in ASCII encoding.

0.5.3 Message Syntax

We will be sending a basic TCP segment and our main focus will be on controlling flags rather on other fields. It is also to be noted that there will be no payload(data) in our protocol design as main objective is port scanning not Data delivery.

As mentioned in ”Message Type section”, Our first message will be sent with FIN flag set (bit set to 1) and response would be RST flag set or no response. Similarly, the second message will be sent with ACK flag set.



0.5.4 Message Semantics

We are using two control flags (FIN,ACK) to define our protocol. In TCP communication, FIN flag is used to end the transmission and ACK flag is used to acknowledge the successful delivery of packet.

But what happens is, if we try to send a FIN flag-set TCP segment before making any connection, So in accordance with TCP RFC rules, we may get RST flag-set segment or no response (means open or filtered). Similarly, ACK will also provide us with RST or Filtered status. One thing to note that, if you read TCP RFC, it mentions that ACK segment sent before any connection will respond RST of only those ports, which are only Filtered and no other choice of open/closed.

The Source port will contain client side application port number and destination will have the server side number. The Header length(data offset) will be containing the length of our TCP header. We will calculate it row by row in out TCP segment and its minimum length is always 20 bytes.

Fields like Sequence number and Acknowledgement number are set to 0 by default but they are 32bit in size. These are in no use because they are used in reliable data delivery after the connection is established. Window size is used by receiver that the client can accept that amount of Data. Checksum and Urgent pointer can be set to any value depending upon there size because its still have no use in our protocol design. Same is the case with Data, we are not attaching any Payload with our segments.

0.6 Additional Information

- In our Protocol, the firewall will be bypassed according to loophole in TCP RFC 793.
- The second ACK message is used to check, if there is a firewall blocking some packets, if there is not then our First FIN packet will provide us all open ports and the response of second packet will definitely be none.