THE

SPRAWL

-
-
-

Port Scanning

Port Scanning is a process of identifying listening ports on a networked system. It reveals a wealth of information about the target including running services, operating system, presence of a firewall.

Table of Contents

Introduction

Just as with other reconnaissance techniques, port scanning must be both efficient and reliable in order to be useful. Several techniques were developed to deal with firewalls, intrusion detection systems and other filtering devices while still being able to complete the scan in a timely manner. Today's arsenal of port scanning techniques has moved far beyond the classic TCP Connect Scan to include a number of stealth and more efficient scans like SYN, ACK, and others that are described below.

TCP Port Scanning

TCP port scanning targets services utilizing Transmission Control Protocol (TCP). Such services include Web Servers, SSH, FTP, and others that require reliable communication.

TCP Connect Scan

TCP Connect Scan is the original form of port scanning which attempts to establish a complete connection with a range of ports. A connection is established to the target port with a complete *three-way handshake* exchange (SYN -> SYN/ACK -> ACK). A successful connection indicates an open port.

The exchange that happens when attempting to establish a TCP connection is described in RFC 793 as follows:

> The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.
>
> 1) A --> B SYN my sequence number is X 2) A <-- B ACK your sequence number is X 3) A <-- B SYN my sequence number is Y 4) A --> B ACK your sequence number is Y
>
> Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

If a port is closed, the host will respond with a RST (Reset) packet indicating a closed port. The behavior for closed ports is described in RFC 793 as follows:

> 1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

There is also a possibility that no response will be received at all indicating that the scanned port is filtered by a firewall.

Since a complete connection is established with a target host, TCP Connect Scan is easily detected by most firewalls and IDSs.

Packet Trace

Below is a complete packet trace of a typical TCP Connect scan.

```
# sending initial SYN of the three-way handshake to google.com on port 80 (www):
0.514699 192.168.1.100 -> 72.14.207.99 TCP 58851 > www [SYN] Seq=0 Len=0 MSS=1460 TSV=18536702 TSER=0 WS=2
# receiving SYN/ACK from google indicating an open port port 80 (www):
0.603326 72.14.207.99 -> 192.168.1.100 TCP www > 58851 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
# we complete the three-way handshake by sending ACK back to google with a received sequence number:
0.603362 192.168.1.100 -> 72.14.207.99 TCP 58851 > www [ACK] Seq=1 Ack=1 Win=5840 Len=0
# at last we are sending RST back to google to close the connection:
0.603629 192.168.1.100 -> 72.14.207.99 TCP 58851 > www [RST, ACK] Seq=1 Ack=1 Win=5840 Len=0
```

Here is a packet trace of a scanner attempting to connect to a closed port:

```
# sending initial SYN to port 666 of a local 192.168.1.104 machine
0.163866 192.168.1.100 -> 192.168.1.104 TCP 59079 > 666 [SYN] Seq=0 Len=0 MSS=1460 TSV=18703363 TSER=0 WS=2
# we received RST back so we know that the port is closed
0.163956 192.168.1.104 -> 192.168.1.100 TCP 666 > 59079 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
```

TCP SYN Scan

TCP SYN Scan (aka half-open scan) is a smarter version of the TCP Connect Scan. Since we know that the target host will respond with RST if the port is closed or SYN/ACK if the port is open, there is no need to complete the three-way handshake with the final ACK response. Thus only half of the connection is established to the target port to deduce port's state. As a precaution to finalize the TCP SYN Scan a RST is sent to the target host to avoid a SYN Flooding Denial of Service attack.

TCP SYN Scan is more stealthy than TCP Connect approach because a complete connection is never established thus reducing a risk of detection.

Packet Trace

Below is a complete packet trace of a TCP SYN scan.

```
# Once again we are sending initial SYN to port 80 on google.com
0.695056 192.168.1.100 -> 72.14.207.99 TCP 59002 > www [SYN] Seq=0 Len=0 MSS=1460
# Response is SYN/ACK so we can conclude that the port is indeed open
0.844707 72.14.207.99 -> 192.168.1.100 TCP www > 59002 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
# Since we do not want to complete the connection, we send RST to google
0.844736 192.168.1.100 -> 72.14.207.99 TCP 59002 > www [RST] Seq=1 Len=0
```

TCP FIN, Null, Xmas Scans

TCP FIN Scan was originally introduced by Uriel Maimon and later expanded into other similar variants. In this scan type a FIN packet is sent to the target port with an expectation of a RST reply packet if the port is closed. If the port is any other state no response is sent back. The basis for this scan is described in RFC 793 as follows:

> If the state is CLOSED (i.e., TCB does not exist) then
>
> all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

Using this approach we can only conclude that the port is closed, but not whether it is opened or filtered. Similar approach will work for a packet with any combination of FIN, PSH, and URG flags set as long as SYN, RST, and ACK bits are not set. In fact Null and Xmas scans use this exact approach. Where TCP Null Scan sends a packet with all flags off to the target port and works just like TCP FIN Scan by expecting RST for closed ports. TCP Xmas Scan sends a packet with FIN, PSH, and URG flags.

TCP FIN, Null, and Xmas Scans are even more stealthy compared to SYN Scan. Unfortunately many operating systems send RST to even open ports (Windows, Cisco, BSDI, and others) thus reducing scan's reliability.

Packet Trace

Below is a complete packet trace of a TCP FIN scan. This type of scan generates a response only when the port is both unfiltered and closed an attempt will be made to scan a known closed port 666.

FIN Scan:

```
# sending FIN packet to the target host
0.103592 192.168.1.100 -> 192.168.1.104 TCP 41669 > 666 [FIN] Seq=0 Len=0
# since port is closed we receive RST
0.103689 192.168.1.104 -> 192.168.1.100 TCP 666 > 41669 [RST, ACK] Seq=0 Ack=0 Win=0 Len=0
```

NULL Scan:

```
# sending NULL packet to the target host
0.548606 192.168.1.100 -> 72.14.207.99 TCP 50033 > 666 [] Seq=0 Len=0
# we've got no response from google. we will try again and conclude that the port
# is either open or filtered
1.016353 192.168.1.100 -> 72.14.207.99 TCP 50034 > 666 [] Seq=0 Len=0
```

Xmas Scan:

```
# sending Xmas packet to the target host
0.131860 192.168.1.100 -> 192.168.1.104 TCP 50984 > 666 [FIN, PSH, URG] Seq=0 Urg=0 Len=0
# we've got RST back so we can conclude the port is closed
0.131959 192.168.1.104 -> 192.168.1.100 TCP 666 > 50984 [RST, ACK] Seq=0 Ack=0 Win=0 Len=0
```

TCP ACK Scan

TCP ACK Scan sends an ACK packet to the target port in order to determine whether the port is filtered or unfiltered. For unfiltered ports a RST reply packet will be sent for both open and closed ports. Filtered ports will result in either no response or an ICMP destination unreachable reply packet.

This approach is useful to get through stateless firewalls which block incoming connections by blocking incoming SYN packets, but allowing ACK packets to get through to allow internal hosts to communicate with the rest of the internet. TCP ACK will not work with stateful firewalls.

Packet Trace

Unfiltered response:

```
# Sending ACK packet to the target on port 80
0.425238 192.168.1.100 -> 216.34.181.45 TCP 63851 > www [ACK] Seq=0 Ack=0 Win=2048 Len=0
# We have received  RST back meaning the port is unfiltered
0.459511 216.34.181.45 -> 192.168.1.100 TCP www > 63851 [RST] Seq=0 Len=0
```

Filtered response

```
# Sending ACK packet to the target on port 666
1.728128 192.168.1.100 -> 216.34.181.45 TCP 46985 > 666 [ACK] Seq=0 Ack=0 Win=4096 Len=0
# We have received no response so we try one more time and give up.
1.908035 192.168.1.100 -> 216.34.181.45 TCP 46986 > 666 [ACK] Seq=0 Ack=0 Win=3072 Len=0
```

NOTE: google.com is sending responses contrary to the expected RST packet for filtered ports (666) and no response for unfiltered ports (80) indicating the use of a stateful firewall.

TCP Window Scan

Originally introduced by Uriel Maimon, TCP Window Scan attempts to deduce whether the port is open or closed based on the Window Size and ttl returned by the target host. A positive window size serves as an indicator for an open port while a zero size window indicates a closed port. At the same time if the returning packet's ttl is lower compared to the rest of the received RST packets the port is also likely to be open. Although this applies to the majority of machines on the internet, a small number of systems will actually return the reverse - zero for open ports and positive number for closed ports.

Packet Trace

It was a lot harder to find systems which would identify open ports with positive window sizes; however, there were plenty of systems which returned RST for closed ports or no response at all indicating a filtered port.

```
# Sending ACK packet
0.101139 192.168.1.100 -> 192.168.1.104 TCP 33272 > www [ACK] Seq=0 Ack=0 Win=4096 Len=0
# Window size is 0 indicating a closed port
0.101222 192.168.1.104 -> 192.168.1.100 TCP www > 33272 [RST] Seq=0 Len=0
```

TCP Idlescan

As suggested by Antirez in buqtraq, *idle scan* relies on the predictability of IPID sequence numbers. It works be continuously sampling the current IPID sequence number on a zombie system while forging packets from a zombie to an actual target.

Zombie systems can be chosen based on their low traffic and predictable IPID sequence numbers. The attack begins when a SYN packet is forged on behalf of a zombie system and sent to the port on a target system. When the target system receives a forged SYN packet it will reply with SYN/ACK to the zombie host. Since zombie host did not expect the incoming SYN/ACK packet, it will generate a RST packet back to the target host thus incrementing its IPID by one. On the other hand, if the port on the target host is closed a RST packet will be sent to the zombie host which will be quietly dropped and will not result in an increment.

At last we have to sample the zombie system once again to determine by how much IPID number was incremented while taking into account that our sample request will result in an additional IPID increase. In our final sample if the zombie's IPID counter was incremented by two the target port is indeed open; however, if the IPID counter was incremented only by one the target port is closed.

Idlescan offers the ultimate solution in stealthy port scanning since no packets are transmitted between the scanner machine and the target. In addition to stealth capabilities, this technique allows for mapping of trusted machines useful in bypassing firewall rules.

Packet Trace

In this packet trace we will use a local machine 192.168.1.104 as a zombie to scan google.com on port 80.

```
# First we send SYN/ACK multiple times to the zombie machine to confirm IPIDs are indeed predictable
# in this case they are because for every response they increment by one (2330,2331,2332,2333)
0.454789 192.168.1.100 192.168.1.104 TCP 53007 > www [SYN, ACK] Seq=0 Ack=0 Win=3072 Len=0 MSS=1460 ID=15461
0.454911 192.168.1.104 192.168.1.100 TCP www > 53007 [RST] Seq=0 Len=0 ID=23330
0.486616 192.168.1.100 192.168.1.104 TCP 53008 > www [SYN, ACK] Seq=0 Ack=0 Win=1024 Len=0 MSS=1460 ID=39449
0.486702 192.168.1.104 192.168.1.100 TCP www > 53008 [RST] Seq=0 Len=0 ID=23331
0.522368 192.168.1.100 192.168.1.104 TCP 53009 > www [SYN, ACK] Seq=0 Ack=0 Win=2048 Len=0 MSS=1460 ID=17487
0.522449 192.168.1.104 192.168.1.100 TCP www > 53009 [RST] Seq=0 Len=0 ID=23332
0.554631 192.168.1.100 192.168.1.104 TCP 53010 > www [SYN, ACK] Seq=0 Ack=0 Win=4096 Len=0 MSS=1460 ID=25705
0.554718 192.168.1.104 192.168.1.100 TCP www > 53010 [RST] Seq=0 Len=0 ID=23333

# Next we forge a packet destined for the target host that appears to be coming from the
# zombie machine.
1.082869 192.168.1.104 64.233.167.99 TCP www > www [SYN] Seq=0 Len=0 MSS=1460 ID=38548

# At the same time we will try to sample the current IPID on the zombie which is 2334
1.134610 192.168.1.100 192.168.1.104 TCP 53128 > www [SYN, ACK] Seq=0 Ack=0 Win=4096 Len=0 MSS=1460 ID=34914
1.134696 192.168.1.104 192.168.1.100 TCP www > 53128 [RST] Seq=0 Len=0 ID=23334

# Google.com is now responding to the zombie machine with TCP/ACK attempting to complete a three-way handshake
1.155557 64.233.167.99 192.168.1.104 TCP www > www [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460 ID=8303
1.155656 64.233.167.99 192.168.1.104 TCP www > www [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460 ID=8303

# Since zombie machine has never sent out the SYN request it responds with RST while
# at the same time incrementing its counter by one.
1.155730 192.168.1.104 64.233.167.99 TCP www > www [RST] Seq=1 Len=0 ID=23335
1.155769 192.168.1.104 64.233.167.99 TCP www > www [RST] Seq=1 Len=0 ID=23335

# Once again we sample the current IPID on the zombie machine and determine that it is 2336 meaning that it was
# incremented by 2 since our last recorded sample was 2334. This means that the zombie machine has responded
# with RST to SYN/ACK coming from an open port on the target machine.
1.270610 192.168.1.100 192.168.1.104 TCP 53185 > www [SYN, ACK] Seq=0 Ack=0 Win=3072 Len=0 MSS=1460 ID=16667
1.270698 192.168.1.104 192.168.1.100 TCP www > 53185 [RST] Seq=0 Len=0 ID=23336
```

FTP Bounce Scan

FTP Bounce Scan relies on a weakness in FTP protocol which allows requests made to FTP servers to establish other FTP connections on behalf of the client. This allows ftp client to scan other hosts by first bouncing requests from a vulnerable FTP Server using PORT command. For ports on the vulnerable system that are open we will get "Transfer OK" response while for closed ports we will receive "Can't open data connection" response. This is explained in detail in RFC959:

> DATA PORT (PORT)
>
> The argument is a HOST-PORT specification for the data port to be used in data connection. There are defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command is used, the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be:
>
> PORT h1,h2,h3,h4,p1,p2
>
> where h1 is the high order 8 bits of the internet host address.
>
> ...
>
> In another situation a user might wish to transfer files between two hosts, neither of which is a local host. The user sets up control connections to the two servers and then arranges for a data connection between them. In this manner, control information is passed to the user-PI but data is transferred between the server data transfer processes.

This scanning approach is stealthy although it requires the presence of a vulnerable FTP Server.

Packet Trace

In the example below the following machines will be used to simulate FTP Bounce Scan:

- 192.168.1.100 is the scanner
- 192.168.1.250 is a vulnerable FTP Server allowing proxy connections
- 192.168.1.1 is the target system

Below is a sample attack:

```
# Initial connection is established with a vulnerable FTP Server
0.066851 192.168.1.100 192.168.1.250 TCP 46625 > ftp [SYN] Seq=0 Len=0 MSS=1460 TSV=2166439 TSER=0 WS=2
0.067014 192.168.1.250 192.168.1.100 TCP ftp > 46625 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
0.067090 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSV=2166439 TSER=0

# We login into FTP server using anonymous account
0.069071 192.168.1.250 192.168.1.100         FTP      Response: 220 Welcome to WinFtp Server.
0.069159 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=1 Ack=32 Win=5840 Len=0 TSV=2166439 TSER=97308
7.066605 192.168.1.100 192.168.1.250         FTP      Request: USER anonymous
7.067807 192.168.1.250 192.168.1.100         FTP      Response: 331 Password required for anonymous
7.067897 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=17 Ack=69 Win=5840 Len=0 TSV=2168189 TSER=97378
7.068426 192.168.1.100 192.168.1.250         FTP      Request: PASS -wwwuser@
7.068577 192.168.1.250 192.168.1.100         FTP      Response: 230 Logged on
7.106561 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=33 Ack=84 Win=5840 Len=0 TSV=2168199 TSER=97378

# Now we request a proxy connection to be made for us to the target 192.168.1.1 system
# on port 443 (1,187 translates to 256 + 187 = 443)
9.066781 192.168.1.100 192.168.1.250         FTP      Request: PORT 192,168,1,1,1,187
9.068027 192.168.1.250 192.168.1.100         FTP      Response: 200 Port command successful
9.068169 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=57 Ack=113 Win=5840 Len=0 TSV=2168689 TSER=97398
9.068700 192.168.1.100 192.168.1.250         FTP      Request: LIST

# FTP Server is now attempting to establish a connection to the target system on port 443
9.070580 192.168.1.250 192.168.1.1 TCP 1249 > https [SYN] Seq=0 Len=0 MSS=1460
9.071345 192.168.1.1   192.168.1.250 TCP https > 1249 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
9.071380 192.168.1.250 192.168.1.1 TCP 1249 > https [ACK] Seq=1 Ack=1 Win=17520 Len=0

# Connection was successfully established so FTP attempts to communicate with the target system
9.071491 192.168.1.250 192.168.1.100         FTP      Response: 150 Opening data channel for directory list.
9.071581 192.168.1.250 192.168.1.1           SSL      Continuation Data
9.071618 192.168.1.250 192.168.1.1 TCP 1249 > https [FIN, ACK] Seq=137 Ack=1 Win=17520 Len=0
9.072143 192.168.1.1   192.168.1.250 TCP https > 1249 [ACK] Seq=1 Ack=137 Win=5840 Len=0
9.074239 192.168.1.1   192.168.1.250 TCP https > 1249 [RST, ACK] Seq=1 Ack=138 Win=5840 Len=0
9.110608 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=63 Ack=159 Win=5840 Len=0 TSV=2168700 TSER=97398

# FTP Server confirms to us that the port is indeed open with the Transfer ok response
9.110674 192.168.1.250 192.168.1.100         FTP      Response: 226 Transfer ok
9.110759 192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=63 Ack=176 Win=5840 Len=0 TSV=2168700 TSER=97399

# FTP Server is now attempting to establish a connection to the target system on port 442
9.118678 192.168.1.100 192.168.1.250         FTP      Request: PORT 192,168,1,1,1,186
9.118953 192.168.1.250 192.168.1.100         FTP      Response: 200 Port command successful
9.126485 192.168.1.100 192.168.1.250         FTP      Request: LIST

# Connection to the target system failed because we have received RST response
9.127948 192.168.1.250 192.168.1.1 TCP 1250 > 442 [SYN] Seq=0 Len=0 MSS=1460
9.128427 192.168.1.1   192.168.1.250 TCP 442 > 1250 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
9.313303 192.168.1.250 192.168.1.100 TCP ftp > 46625 [ACK] Seq=205 Ack=93 Win=17428 Len=0 TSV=97401 TSER=2168703
9.641438 192.168.1.250 192.168.1.1 TCP 1250 > 442 [SYN] Seq=0 Len=0 MSS=1460
9.641925 192.168.1.1   192.168.1.250 TCP 442 > 1250 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
10.078939   192.168.1.250 192.168.1.1 TCP 1250 > 442 [SYN] Seq=0 Len=0 MSS=1460
10.079476   192.168.1.1   192.168.1.250 TCP 442 > 1250 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0

# FTP Server lets us know that it couldn't establish the connection to the closed port
10.080471   192.168.1.250 192.168.1.100      FTP      Response: 425 Can't open data connection.
10.118633   192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=93 Ack=238 Win=5840 Len=0 TSV=2168952 TSER=97409
10.231082   192.168.1.100 192.168.1.250 TCP 46625 > ftp [FIN, ACK] Seq=93 Ack=238 Win=5840 Len=0 TSV=2168980 TSER=97409
10.231157   192.168.1.250 192.168.1.100 TCP ftp > 46625 [ACK] Seq=238 Ack=94 Win=17428 Len=0 TSV=97410 TSER=2168980
10.231944   192.168.1.250 192.168.1.100 TCP ftp > 46625 [FIN, ACK] Seq=238 Ack=94 Win=17428 Len=0 TSV=97410 TSER=2168980
10.232022   192.168.1.100 192.168.1.250 TCP 46625 > ftp [ACK] Seq=94 Ack=239 Win=5840 Len=0 TSV=2168980 TSER=97410
```

UDP

UDP port scanning targets services utilizing User Datagram Protocol (UDP). Such services include DNS, VoIP, TFTP, and others that do not require reliable communication.

UDP ICMP port unreachable scanning

This technique sends a UDP datagram to a target host and waits for ICMP_PORT_UNREACH message for closed ports. If such message never arrives we can deduce that the port is open.

Packet Trace

Below is the packet trace for UDP scan of a machine on port 666.

```
# Sending UDP datagram to port 666 on 192.168.1.104
0.094182 192.168.1.100 -> 192.168.1.104 UDP Source port: 42949  Destination port: 666
# Received ICMP port unreachable response indicating the port is closed
0.094270 192.168.1.104 -> 192.168.1.100 ICMP Destination unreachable (Port unreachable)
```

Other

IP Protocol Scan

Originally introduced by Gerhard Rieger, IP Protocol Scan sends a number of IP packets to the target host with varying values in the protocol field of the IP datagram. The target system should reply with ICMP Destination Protocol Unreachable packet for protocol types which are not supported by the system. If there is no response we can assume the protocol is supported. Such scan is useful in search of systems that support protocols beyond the usual TCP and UDP such as routers. This behavior is described in RFC 792:

> If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

External Links

- RFC 791
- RFC 792
- RFC 793
- RFC 959
- IANA Assigned Port Numbers
- Phrack 49 Port Scanning without the SYN flag
- Idle Scanning and Related IPID Games
- The Art of Port Scanning
- TRUE Blind ip spoofed portscanning

Published on December 22nd, 2008 by iphelix