port scan happens when a single source IP tries to connect to **many different destination ports or IPs** within a short time.

Data Collection

It searches firewall logs ,

```
index=firewall sourcetype="cisco:asa"
```

It looks for logs containing keywords like:

```
DPT=, dst_port, SRC=, proto=TCP, DROP, allowed tcp
```

These help identify traffic connection attempts.

```
("DPT=" OR "dst_port" OR "destination_port" OR "SIP=" OR "SRC=" OR "proto=TCP" OR "Connection refused" OR "allowed tcp" OR "DROP")
```

This is pre filter, not extraction, extraction means rex.

Why only pre filter destination port not destination ip or source ip

Attackers hit ports, not random fields.

Port scanning always involves destination ports.

Failed/allowed connections always contain port info.

Port formats are chaotic across vendors — so you need multiple variants (DPT=, dpt, dst_port, etc.) in the initial match.

That's why destination-port keywords are included in the PRE-FILTER:
They capture the type of traffic you're trying to detect.

Source IP and Destination IP appear in almost every firewall log line.

If you filter for:

SRC=
DST=
source
dest

You will end up matching:

Routing logs
NAT logs
VPN session logs
System messages
Health checks
Heartbeats
ARP logs
Status updates

These all contain SRC or DST patterns.

That means your search gets flooded with noise.

Destination port keywords are specific to connection events.
SRC= and DST= are everywhere, and ruin your filter.

Extracting Key Fields (with rex)

Source/Destination IP extraction is done *properly* later with regex

This is the correct workflow:

Pre-filter → reduce data

Rex → extract the real fields

You NEVER use IP patterns in pre-filter because:

- They match too much junk
- They hit too many log types
- They reduce performance
- They return irrelevant events
- They make your detection noisy

Destination-port keywords help **narrow the data** *before* extraction.

(rex) that pull out the key network fields (src_ip, dest_ip, dest_port) from raw log text.

The rule uses regex to extract:

src_ip → attacker/source IP
dest_ip → destination target
dest_port → port numbers being scanned.

Extracting Source IP

| rex max_match=0 "(?:SRC|source|src|saddr)[=:](?<src_ip>\b(?:\d{1,3}\.){3}\d{1,3}\b)"

rex → tells Splunk to use a regex to extract data from _raw logs.

max_match=0 → allows extracting multiple matches (useful if one event has multiple IPs).

(?:SRC|source|src|saddr) → matches any of these common field labels used by firewalls and save it as a source ip.

SRC=192.168.1.10

source=192.168.1.10

src:192.168.1.10

[=:] → means the value may follow = or : or a space.

(?<src_ip>\b(?:\d{1,3}\.){3}\d{1,3}\b) → captures the actual IP address (like 192.168.1.10) into a field named src_ip.

created a new field src_ip = 192.168.1.10

Extracting Destination Port

| rex max_match=0 "(?:DPT|dst_port|dport|destination_port|dst_port=|dpt)[= :]*(?<dest_port>\d{1,5})"

(?:DPT|dst_port|dport|destination_port|dst_port=|dpt) → matches any of these common field labels used by firewalls.save it as a destination Port.

DPT=443, dst_port:443, destination_port=80, etc.

[=:] → means the value may follow = or : or a space.

(?<dest_port>\d{1,5}) → captures a port number

dest_port = 443

Extracting Destination IP

| rex max_match=0 "(?:DST|dest|daddr)[=:](?<dest_ip>\b(?:\d{1,3}\.){3}\d{1,3}\b)"

(?:DPT|dst_port|dport|destination_port|dst_port=|dpt) → matches any of these common field labels used by firewalls.and save it as a destination ip.

DST=192.168.1.10

dest=192.168.1.10

daddr:192.168.1.10

[=:] → means the value may follow = or : or a space.

(?<dest_ip>\b(?:\d{1,3}\.){3}\d{1,3}\b) → captures a destination ip

dest_ip = 8.8.8.8

Time Window

| bin _time span=5m

Splunk processes raw machine data by dividing it into individual events.Each event has a timestamp.The bin command is used to group these individual events into a consistent time period, like 5 minutes.

Groups all events into 5-minute time windows.

This lets Splunk count how many connection attempts happened from one source to many destinations within 5 minutes time period.

Build a Behavior Snapshot

| stats count as events dc(dest_port) as unique_ports values(dest_port) as ports first(sourcetype) as sample_sourcetype by src_ip, dest_ip, _time

count as events
Counts the number of log events in each group.

Renames the count field as events.
Example: If 192.168.1.10 connected to 8.8.8.8 30 times → events = 30

dc(dest_port) as unique_ports

dc() = distinct count → counts how many unique destination ports were contacted.
Example:
If connections were to ports 22, 80, 443, 445 → unique_ports = 4

values(dest_port) as ports

Collects and lists all the destination ports seen in that group.
Example:
ports = [22, 80, 443, 445]

first(sourcetype) as sample_sourcetype

Captures the first seen log source (like cisco:asa, pan:log, fortinet:utm).
Used just for context — helps analysts know what device the data came from

by src_ip, dest_ip, _time

Groups all the above stats by:

src_ip → who initiated the connection,

dest_ip → who was targeted,

_time → time window (from the previous | bin _time span=5m command).

So each source → destination pair within a 5-minute window becomes one summarized record.

Filters suspicious hosts

| where unique_ports > 10 OR (unique_ports >= 5 AND events > 20)

Filters only suspicious hosts.

Triggers if:

The host touched more than 10 unique ports, or

It touched ≥5 unique ports with >20 total events.

Purpose: catch scanning behavior

Assigns severity level

| eval severity=case(...)

Assigns a severity level based on how many ports were probed:

≥100 → Critical

≥50 → High

≥20 → Medium

10 → Low

Else → Info

Purpose: prioritize which scans are more aggressive.

Creates a readable note field

| eval note=sample_sourcetype . " observed; hits=" . events . " ports=" . unique_ports

Creates a readable note field summarizing what was seen, like:
"pan:log observed; hits=80 ports=35"

Purpose: quick human-readable summary for alert details.

Displays only the useful columns

| table _time, src_ip, dest_ip, events, unique_ports, ports, sample_sourcetype, severity, note

Displays only the useful columns for the analyst to review.

Sorts the results

| sort - unique_ports, -events

Sorts the results — the most suspicious (most ports/events) appear at the top.

Detect suspicious port-scanning behavior

| where unique_ports > 10 OR (unique_ports >= 5 AND events > 20)

This line identifies hosts that contacted many different ports or made lots of connection attempts in a short time.

Where

where in Splunk is a filtering command — it removes rows that do not meet a condition.
where keeps only the results that satisfy a logical condition (true).
Everything else is dropped

unique_ports > 10

If a host touches **more than 10 different destination ports** in a short time (5-minute window),
→ **Likely a port scan**

(unique_ports ≥ 5 AND events > 20)

If a host tried to connect 5 or more different ports, and
generated more than 20 total connection attempts.

Give severity levels

```
| eval severity=case(unique_ports >= 100, "Critical",
                   unique_ports >= 50, "High",
                   unique_ports >= 20, "Medium",
                   unique_ports > 10, "Low",
                   1==1, "Info")
```

eval → creates the field
case() → decides the value based on conditions.

A field is simply a name–value pair inside each event.src_ip, user, status, unique_ports is a field.
After eval, every event gets a new field called severity with a value (“Critical”, “High”, “Medium”, “Low”, or “Info”)

Your events already had a field called unique_ports:
unique_ports = 120
unique_ports = 55
unique_ports = 23

case() looks at unique_ports, tests your conditions, and sets a new field:
severity = Critical
severity = High
severity = Medium
severity = Low
severity = Info

Example (for clarity)

Event before eval:
src_ip = 10.1.1.5
unique_ports = 120

After running your eval:
src_ip = 10.1.1.5
unique_ports = 120
severity = Critica
severity is the field created.

“If nothing else matched, assign severity = Info.”
Example scenario
unique_ports = 3
Not ≥ 100
Not ≥ 50
Not ≥ 20
Not > 10
So… fall back to 1==1 → TRUE

severity = "Info”

creates a human-readable summary line

```
| eval note = sample_sourcetype . " observed; hits=" . events . " ports=" . unique_ports
```

This line creates a new field called note and builds a text string by joining multiple values together.

Splunk uses the dot (.) for string joining — meaning it sticks text pieces together.

note =

You are creating a new field called note.

sample_sourcetype

This is an existing field in your events (e.g., linux_secure, firewall, apache_access).

" observed; hits="

A fixed text string (literal text).

events

Another field you already calculated earlier.

" ports="

Another fixed text.

unique_ports

A numeric field created earlier.

Putting it all together

The dot (.) simply means “join these together”

Example:-
sample_sourcetype . " observed; hits=" . events

Real example:-
sample_sourcetype = linux_secure
events = 45
unique_ports = 12

Becomes a single long string.

note = "linux_secure observed; hits=45 ports=12”

Useful for dashboards, reports, email alerts, etc.

Create a table format

| table _time, src_ip, dest_ip, events, unique_ports, ports, sample_sourcetype, severity, note

Splunk throws away every other field and shows ONLY these:in table format.

_time
src_ip
dest_ip
events
unique_ports
ports
sample_sourcetype
severity
note

Before table:
The event has 20+ fields

After table:
The event has 9 fields only, arranged cleanly as a table.

Show the most suspicious sources first

sort - unique_ports, -events

This command sorts your final results based on two fields:

unique_ports → in descending order (because of the -)

events → also in descending order (because of the -)

Descending = highest values first.

“First, sort everything by unique_ports from biggest to smallest.”

“If two rows have the same number of unique ports, then sort those by events from biggest to smallest.”

Full version

```
index=firewall sourcetype="cisco:asa”

("DPT=" OR "dst_port" OR "destination_port" OR "SIP=" OR "SRC=" OR "proto=TCP" OR "Connection refused" OR "allowed tcp" OR "DROP")

| rex max_match=0 "(?:SRC|source|src|saddr)[=: ](?<src_ip>\b(?:\d{1,3}\.){3}\d{1,3}\b)”


| rex max_match=0 "(?:DPT|dst_port|dport|destination_port|dst_port=|dpt)[= :]*(?<dest_port>\d{1,5})”

| rex max_match=0 "(?:DST|dest|daddr)[=: ](?<dest_ip>\b(?:\d{1,3}\.){3}\d{1,3}\b)”

| bin _time span=5m

| stats count as events dc(dest_port) as unique_ports values(dest_port) as ports first(sourcetype) as sample_sourcetype by src_ip, dest_ip, _time

| where unique_ports > 10 OR (unique_ports >= 5 AND events > 20)

| eval severity=case(unique_ports >= 100, "Critical",
                     unique_ports >= 50, "High",
                     unique_ports >= 20, "Medium",
                     unique_ports > 10, "Low",
                     1==1, "Info")

| eval note=sample_sourcetype . " observed; hits=" . events . " ports=" . unique_ports

| table _time, src_ip, dest_ip, events, unique_ports, ports, sample_sourcetype, severity, note

| sort - unique_ports, -events
```

———————————————

This rule detects possible port-scanning activity by searching recent firewall logs from multiple vendors (Cisco ASA, Palo Alto, Fortinet, or any log source containing “firewall”). It looks for connection events that include destination-port and source-IP information.

It extracts:

the source IP attempting the connections,

the destination IP,

and all destination ports that source attempted to reach.

The rule groups these events into 5-minute time windows, then counts:

how many total events came from each source IP → events

how many unique ports were targeted → unique_ports

A source IP is flagged when:

it hits more than 10 different ports, OR

it hits 5 or more ports AND generates more than 20 events

— patterns commonly seen in horizontal or vertical port scans.

It then assigns a severity level based on how many unique ports were scanned:

Critical if ≥100 ports

High if ≥50

Medium if ≥20

Low if >10