TCP SYN (Stealth) Scan (-sS)

SYN scan is the default and most popular scan option for good reason. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by intrusive firewalls. SYN scan is relatively unobtrusive and stealthy, since it never completes TCP connections. It also works against any compliant TCP stack rather than depending on idiosyncrasies of specific platforms as Nmap's FIN/NULL/Xmas, Maimon and idle scans do. It also allows clear, reliable differentiation between `open`, `closed`, and `filtered` states.

SYN scan may be requested by passing the `-sS` option to Nmap. It requires raw-packet privileges, and is the default TCP scan when they are available. So when running Nmap as root or Administrator, `-sS` is usually omitted. This default SYN scan behavior is shown in Example 5.1, which finds a port in each of the three major states.

Example 5.1. A SYN scan showing three port states

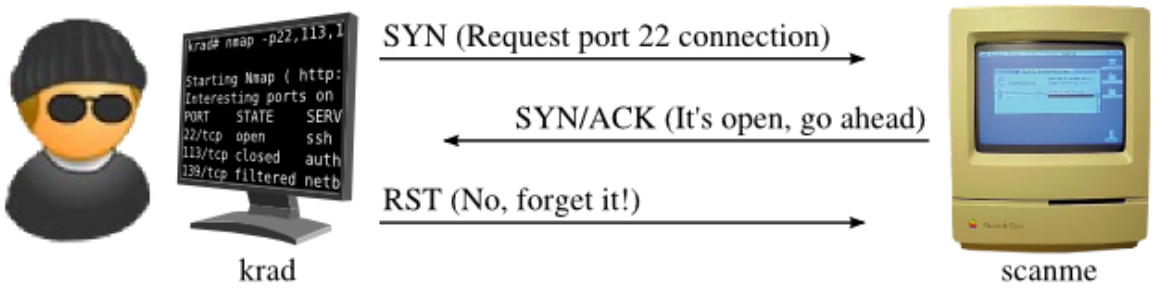
```
krad# nmap -p22,113,139 scanme.nmap.org

Starting Nmap ( https://nmap.org )
Nmap scan report for scanme.nmap.org (64.13.134.52)
PORT    STATE    SERVICE
22/tcp  open     ssh
113/tcp closed   auth
139/tcp filtered netbios-ssn

Nmap done: 1 IP address (1 host up) scanned in 1.35 seconds
```

While SYN scan is pretty easy to use without any low-level TCP knowledge, understanding the technique helps when interpreting unusual results. Fortunately for us, the fearsome black-hat cracker Ereet Hagiwara has taken a break from terrorizing Japanese Windows users to illustrate the Example 5.1 SYN scan for us at the packet level. First, the behavior against open port 22 is shown in Figure 5.2.

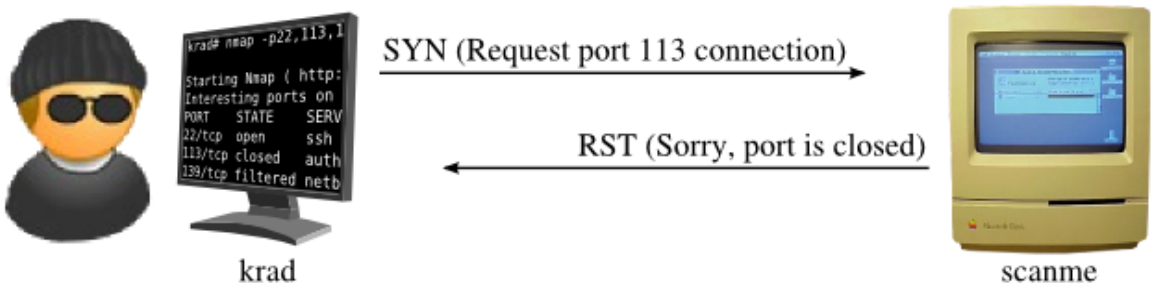
Figure 5.2. SYN scan of open port 22



As this example shows, Nmap starts by sending a TCP packet with the SYN flag set (see Figure 2, “TCP header” if you have forgotten what packet headers look like) to port 22. This is the first step in the TCP three-way handshake that any legitimate connection attempt takes. Since the target port is open, Scanme takes the second step by sending a response with the SYN and ACK flags back. In a normal connection, Ereet's machine (named krad) would complete the three-way handshake by sending an ACK packet acknowledging the SYN/ACK. Nmap does not need to do this, since the SYN/ACK response already told it that the port is open. If Nmap completed the connection, it would then have to worry about closing it. This usually involves another handshake, using FIN packets rather than SYN. So an ACK is a bad idea, yet something still has to be done. If the SYN/ACK is ignored completely, Scanme will assume it was dropped and keep re-sending it. The proper response, since we don't want to make a full connection, is a RST packet as shown in the diagram. This tells Scanme to forget about (reset) the attempted connection. Nmap could send this RST packet easily enough, but it actually doesn't need to. The OS running on krad also receives the SYN/ACK, which it doesn't expect because Nmap crafted the SYN probe itself. So the OS responds to the unexpected SYN/ACK with a RST packet. All RST packets described in this chapter also have the ACK bit set because they are always sent in response to (and acknowledge) a received packet. So that bit is not shown explicitly for RST packets. Because the three-way handshake is never completed, SYN scan is sometimes called half-open scanning.

Figure 5.3 shows how Nmap determines that port 113 is closed. This is even simpler than the open case. The first step is always the same—Nmap sends the SYN probe to Scanme. But instead of receiving a SYN/ACK back, a RST is returned. That settles it—the port is closed. No more communication regarding this port is necessary.

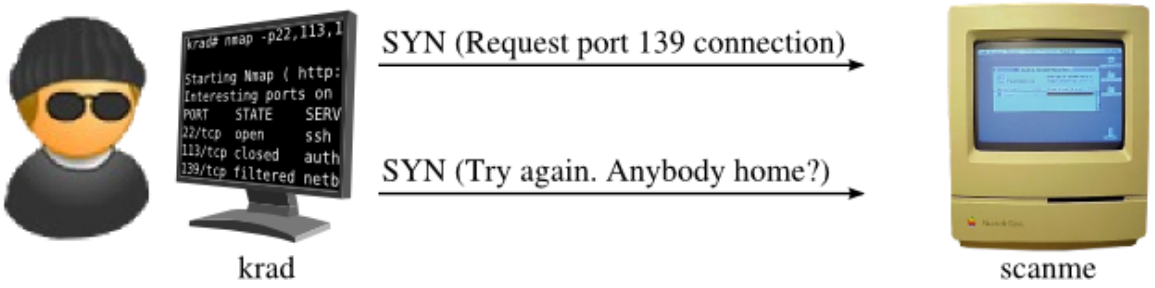
Figure 5.3. SYN scan of closed port 113



Finally, Ereet shows us how a filtered port appears to Nmap in Figure 5.4. The initial SYN is sent first, as usual, but Nmap sees no reply. The response could simply be slow. From previous responses (or timing defaults), Nmap knows how long to wait and eventually gives up on receiving one. A non-responsive port is

usually filtered (blocked by a firewall device, or perhaps the host is down), but this one test is not conclusive. Perhaps the port is open but the probe or response were simply dropped. Networks can be flaky. So Nmap tries again by resending the SYN probe. After yet another timeout period, Nmap gives up and marks the port `filtered`. In this case, only one retransmission was attempted. As described in the section called “Scan Code and Algorithms”, Nmap keeps careful packet loss statistics and will attempt more retransmissions when scanning less reliable networks.

Figure 5.4. SYN scan of filtered port 139



Nmap will also consider a port `filtered` if it receives certain ICMP error messages back. Table 5.2 shows how Nmap assigns port states based on responses to a SYN probe.

Table 5.2. How Nmap interprets responses to a SYN probe

Probe Response	Assigned State
TCP SYN/ACK response	`open`
TCP RST response	`closed`
No response received (even after retransmissions)	`filtered`
ICMP unreachable error (type 3, code 1, 2, 3, 9, 10, or 13)	`filtered`

While the pretty illustrations in this section are useful when you have them, Nmap reports exactly what it is doing at the packet level when you specify the `--packet-trace` option in addition to any other desired command-line flags. This is a great way for newbies to understand Nmap's behavior when Ereet is not around to help. Even advanced users find it handy when Nmap produces results they don't expect. You may want to increase the debug level with `-d` (or even `-d5`) as well. Then scan the minimum number of ports and hosts necessary for your purpose or you could end up with literally millions of output lines. Example 5.2 repeats Ereet's three-port SYN scan with packet tracing enabled (output edited for brevity). Read the command-line, then test yourself by figuring out what packets will be sent before reading on. Then once you read the trace up to “The SYN Stealth Scan took 1.25s”, you should know from the RCVD lines what the port state table will look like before continuing on to read it.

Example 5.2. Using `--packet-trace` to understand a SYN scan

```
krad# nmap -d --packet-trace -p22,113,139 scanme.nmap.org

Starting Nmap ( https://nmap.org )
SENT (0.0130s) ICMP krad > scanme echo request (type=8/code=0) ttl=52 id=1829
SENT (0.0160s) TCP krad:63541 > scanme:80 A iplen=40 seq=91911070 ack=99850910
RCVD (0.0280s) ICMP scanme > krad echo reply (type=0/code=0) iplen=28
```

```
We got a ping packet back from scanme: id = 48821 seq = 714 checksum = 16000
massping done:  num_hosts: 1  num_responses: 1
Initiating SYN Stealth Scan against scanme.nmap.org (scanme) [3 ports] at 00:53
SENT (0.1340s) TCP krad:63517 > scanme:113 S iplen=40 seq=10438635
SENT (0.1370s) TCP krad:63517 > scanme:22 S iplen=40 seq=10438635
SENT (0.1400s) TCP krad:63517 > scanme:139 S iplen=40 seq=10438635
RCVD (0.1460s) TCP scanme:113 > krad:63517 RA iplen=40 seq=0 ack=10438636
RCVD (0.1510s) TCP scanme:22 > krad:63517 SA iplen=44 seq=75897108 ack=10438636
SENT (1.2550s) TCP krad:63518 > scanme:139 S iplen=40 seq=10373098 win=3072
The SYN Stealth Scan took 1.25s to scan 3 total ports.
Nmap scan report for scanme.nmap.org (64.13.134.52)
PORT    STATE    SERVICE
22/tcp  open     ssh
113/tcp closed   auth
139/tcp filtered netbios-ssn

Nmap done: 1 IP address (1 host up) scanned in 1.40 seconds
```

SYN scan has long been called the stealth scan because it is subtler than TCP connect scan (discussed next), which was the most common scan type before Nmap was released. Despite that moniker, don't count on a default SYN scan slipping undetected through sensitive networks. Widely deployed intrusion detection systems and even personal firewalls are quite capable of detecting default SYN scans. More effective techniques for stealthy scanning are demonstrated in Chapter 10, *Detecting and Subverting Firewalls and Intrusion Detection Systems*.

◀ Prev
Chapter 5. Port Scanning Techniques and Algorithms

▲ Up
Chapter 5. Port Scanning Techniques and Algorithms

Home

Next ▶
TCP Connect Scan (-sT)