Port Scanning Basics

While Nmap has grown in functionality over the years, it began as an efficient port scanner, and that remains its core function. The simple command **nmap *<target>*** scans 1,000 TCP ports on the host *<target>*. While many port scanners have traditionally lumped all ports into the open or closed states, Nmap is much more granular. It divides ports into six states: `open`, `closed`, `filtered`, `unfiltered`, `open|filtered`, or `closed|filtered`.

These states are not intrinsic properties of the port itself, but describe how Nmap sees them. For example, an Nmap scan from the same network as the target may show port `135/tcp` as open, while a scan at the same time with the same options from across the Internet might show that port as `filtered`.

The six port states recognized by Nmap

open

> An application is actively accepting TCP connections, UDP datagrams or SCTP associations on this port. Finding these is often the primary goal of port scanning. Security-minded people know that each open port is an avenue for attack. Attackers and pen-testers want to exploit the open ports, while administrators try to close or protect them with firewalls without thwarting legitimate users. Open ports are also interesting for non-security scans because they show services available for use on the network.

closed

> A closed port is accessible (it receives and responds to Nmap probe packets), but there is no application listening on it. They can be helpful in showing that a host is up on an IP address (host discovery, or ping scanning), and as part of OS detection. Because closed ports are reachable, it may be worth scanning later in case some open up. Administrators may want to consider blocking such ports with a firewall. Then they would appear in the filtered state, discussed next.

filtered

> Nmap cannot determine whether the port is open because packet filtering prevents its probes from reaching the port. The filtering could be from a dedicated firewall device, router rules, or host-based firewall software. These ports frustrate attackers because they provide so little information. Sometimes they respond with ICMP error messages such as type 3 code 13 (destination unreachable: communication administratively prohibited), but filters that simply drop probes without responding are far more common. This forces Nmap to retry several times just in case the probe was dropped due to network congestion rather than filtering. This slows down the scan dramatically.

unfiltered

> The unfiltered state means that a port is accessible, but Nmap is unable to determine whether it is open or closed. Only the ACK scan, which is used to map firewall rulesets, classifies ports into

this state. Scanning unfiltered ports with other scan types such as Window scan, SYN scan, or FIN scan, may help resolve whether the port is open.

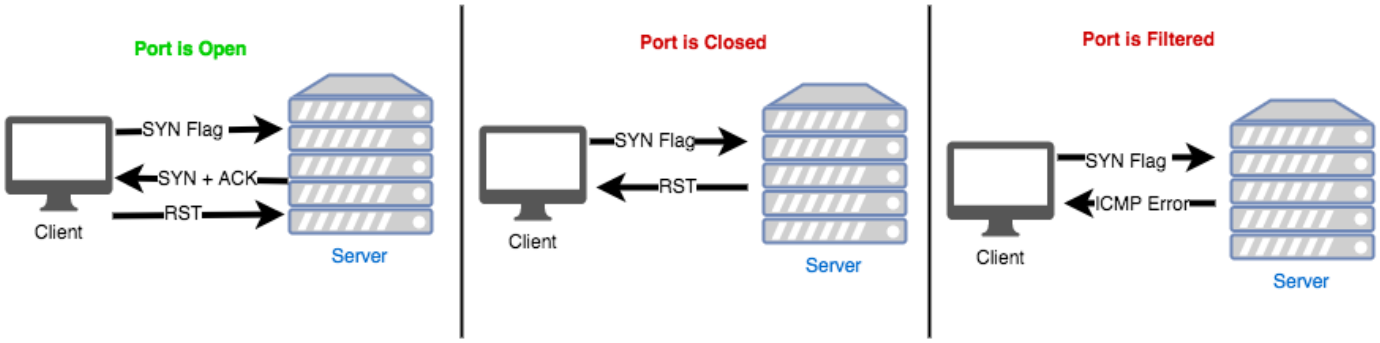
open|filtered

Nmap places ports in this state when it is unable to determine whether a port is open or filtered. This occurs for scan types in which open ports give no response. The lack of response could also mean that a packet filter dropped the probe or any response it elicited. So Nmap does not know for sure whether the port is open or being filtered. The UDP, IP protocol, FIN, NULL, and Xmas scans classify ports this way.

closed|filtered

This state is used when Nmap is unable to determine whether a port is closed or filtered. It is only used for the IP ID idle scan.

Stealth scanning



Stealth scanning is a form of TCP scanning. Here the port scanner creates raw IP packets and sends them to the host to monitor for responses. This type of scan is also known as half-open scanning, or SYN scanning, as it never opens a complete TCP connection. This type of scanner creates a SYN packet and sends it to the host. If the target port is open, the host will respond with a SYN-ACK packet. Then the client will respond with an RST packet to close the connection before completing the handshake. If the port is closed but unfiltered, the target will instantly respond with an RST packet.

To create a SYN scanner, we will use the Scapy module. It is a powerful interactive packet manipulation program and library.

We can create a SYN scanner with the following steps:

1. Create a new file called `syn-scanner.py` and open it in your editor.
2. As usual, import the required modules:

```
from scapy.all import *
```

This will import the `scapy` module

3. Now we can declare the variables, and we can also pass these variables as arguments, if required:

```
host = 'www.dvwa.co.uk'
ip = socket.gethostbyname(host)
openp = []
filterdp = []
common_ports = { 21, 22, 23, 25, 53, 69, 80, 88, 109, 110,
                 123, 137, 138, 139, 143, 156, 161, 389,       443, 445, 500, 546,
547, 587, 660, 995,       993, 2086, 2087, 2082, 2083, 3306, 8443,       10000 }
```

4. Now we can create a function to check whether the host is up or down:

```
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
        return True
```

We create and send an ICMP packet to the host. The host will respond if it is up.

5. Next, we can create a function to scan the port using SYN packets:

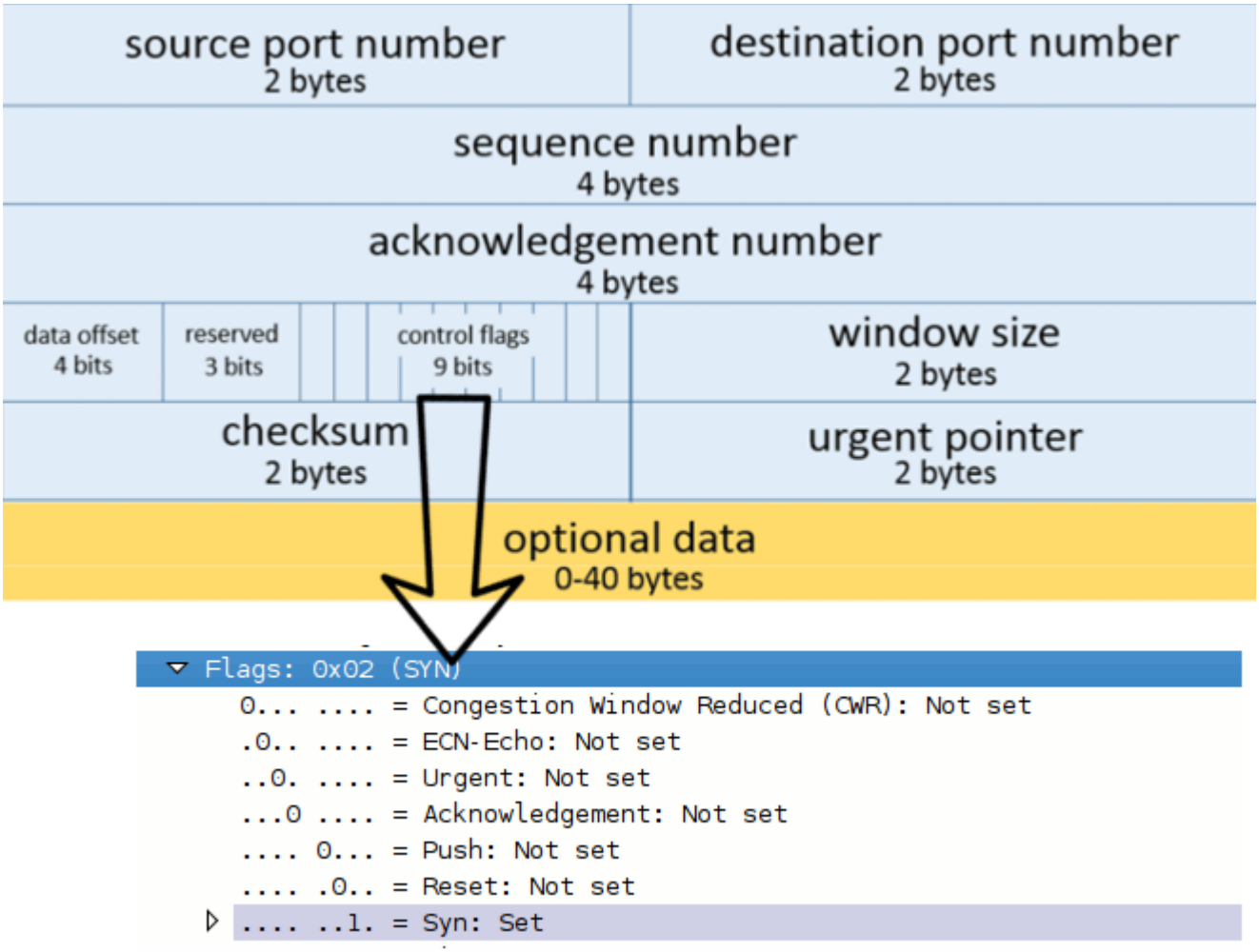
```
def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='F')
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 1
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x14:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and
int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 2
    except Exception as e:
        pass
    return result
```

Here we set a random port as the destination port, and then create a SYN packet with the source port, destination port, and destination IP. Then we will send the packet and analyze the response. If the response type is `None`, then the port is closed. If the response has a TCP layer, then we have to check

the flag value in it. The flag has nine bits, but we check for the control bits, which have six bits. They are:

- URG = 0x20
- ACK = 0x10
- PSH = 0x08
- RST = 0x04
- SYN = 0x02
- FIN = 0x01

The following is the header structure for the TCP layer:



So, if the flag value is 0x12, then the response has a SYN flag and we can consider the port to be open. If the value is 0x14, then the flag is RST/ACK, so the port is closed.

6. Then we will check whether the host is up or not, loop through the common ports list, and scan each port if the host is up:

```
if is_up(ip):
for port in common_ports:
    print (port)
```

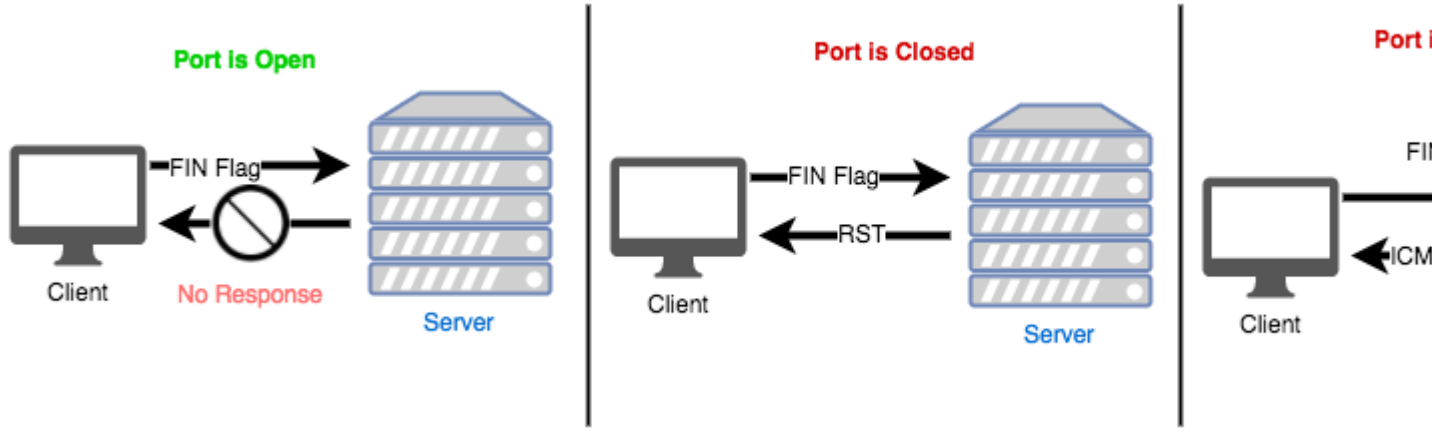
```
            response = probe_port(ip, port)
            if response == 1:
                openp.append(port)
            elif response == 2:
                filterdp.append(port)
        if len(openp) != 0:
            print ("Possible Open or Filtered Ports:")
            print (openp)
        if len(filterdp) != 0:
            print ("Possible Filtered Ports:")
            print (filterdp)
        if (len(openp) == 0) and (len(filterdp) == 0):
            print ("Sorry, No open ports found.!!")
    else:
        print ("Host is Down")
```

Each port from the common port list is scanned and the identified open ports are added to the open ports list, following the list that is printed

7. Make sure to run the script with `sudo`, as we are using Scapy and Scapy requires admin privileges:

```
sudo python3 syn-scanner.py
```

FIN scanning



SYN scanning can be blocked by firewalls. However, packets with the FIN flag set have the ability to bypass firewalls. Here is how it works--for a FIN packet, the closed ports reply with an RST packet, whereas the open ports ignore the packets. If it's an ICMP packet with type 3, and code 1, 2, 3, 9, 10, or 13, we may infer that the port is filtered and the port state cannot be found. We can use Scapy to create the FIN packet and scan the ports.

How to do it...

We can create a FIN scanner as following:

1. As we did in the previous recipe, we have to create another file, `fin-scanner.py`, and open it in our editor.
2. Then import the required module:

```
from scapy.all import *
```

3. As we did for the SYN scanner, set the variables and create the function to check whether the server is up or not:

```
host = 'www.dvwa.co.uk'
ip = socket.gethostbyname(host)
openp = []
filterdp = []
common_ports = { 21, 22, 23, 25, 53, 69, 80, 88, 109, 110,
                 123, 137, 138, 139, 143, 156, 161, 389, 443,
                 445, 500, 546, 547, 587, 660, 995, 993, 2086,
                 2087, 2082, 2083, 3306, 8443, 10000
                }
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
```

```
        return True
```

4. Now we can create the function to probe the ports as follows:

```
def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='F')
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 1
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x14:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and
int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 2
    except Exception as e:
        pass
    return result
```

Here we changed the flag to `F` for `FIN`, while creating the packet to send

5. Finally, we will check whether the host is up or not, loop through the common ports list, and scan each port if the host is up:

```
    if is_up(ip):
        for port in common_ports:
            print (port)
            response = probe_port(ip, port)
            if response == 1:
                openp.append(port)
            elif response == 2:
                filterdp.append(port)
        if len(openp) != 0:
            print ("Possible Open or Filtered Ports:")
            print (openp)
        if len(filterdp) != 0:
            print ("Possible Filtered Ports:")
            print (filterdp)
        if (len(openp) == 0) and (len(filterdp) == 0):
            print ("Sorry, No open ports found.!!")
    else:
        print ("Host is Down")
```

XMAS scanning

With XMAS scanning, we will send a TCP packet with a bunch of flags all at once (PSH, FIN, and URG). We will get an RST if the port is closed. If the port is open or filtered, then there will be no response from the server. It's similar to the FIN scan, other than the packet-creating part

How to do it...

Here are the steps to create a XMAS scanner with Scapy:

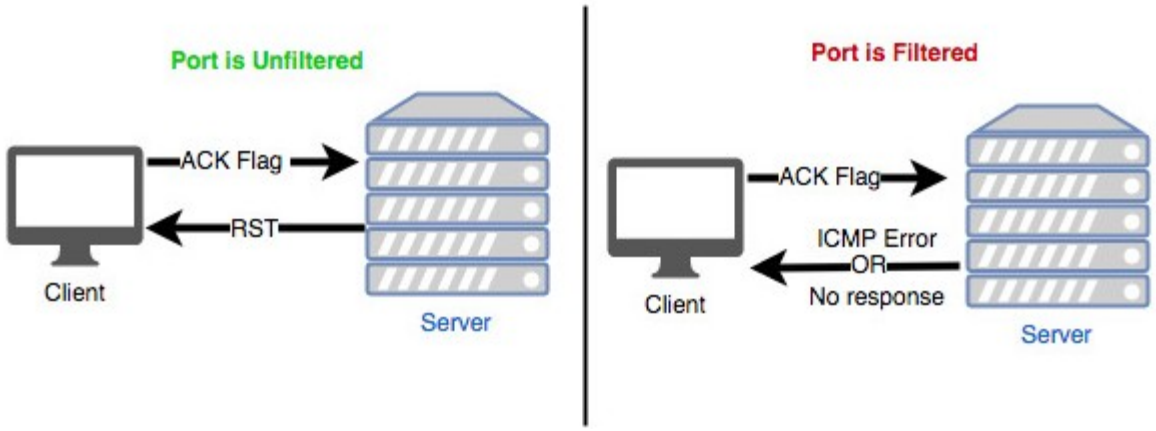
1. Create a copy of the file we created for the previous recipe (*FIN scanning*). As it is quite similar, we only need to change the packet-creation section.
2. To create and send a packet with PSH, FIN, and URG flags in it, update the packet-crafting section inside the `probe_port` method, as follows:

```
p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='FPU')
```

Only update the flags parameter. Here we set the flags as `FPU` for PSH, FIN, and URG combined.

TCP ACK scanning

ACK flag scanning is useful to verify whether the server is blocked with firewalls, IPS, or other network security controls. As in the FIN scan, we will send an TCP ACK packet. No response or an ICMP error indicates the presence of a stateful firewall, as the port is filtered, and if we get back an RST-ACK, then the stateful firewall is absent:



How to do it...

The steps to create a TCP ACK scanner with Scapy are as following:

1. As usual, import the required modules and set the variables. Also, define the method to check the status of the host:

```
from scapy.all import *
# define the host, port
host = 'rejahrehim.com'
ip = socket.gethostbyname(host)
port = 80
# define the method to check the status of host
def is_up(ip):
    icmp = IP(dst=ip)/ICMP()
    resp = sr1(icmp, timeout=10)
    if resp == None:
        return False
    else:
        return True
```

2. To send a TCP packet with the ACK flag, update the `probe_port` method in the previous recipe, as follows:

```
def probe_port(ip, port, result = 1):
    src_port = RandShort()
    try:
        p = IP(dst=ip)/TCP(sport=src_port, dport=port, flags='A', seq=12345)
        resp = sr1(p, timeout=2) # Sending packet
        if str(type(resp)) == "<type 'NoneType'>":
            result = 1
```

```
        elif resp.haslayer(TCP):
            if resp.getlayer(TCP).flags == 0x4:
                result = 0
            elif (int(resp.getlayer(ICMP).type)==3 and
int(resp.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                result = 1
    except Exception as e:
        pass
    return result
```

Here we create a TCP ACK packet and send it to the host

3. Finally, run the scanner, as follows:

```
    if is_up(ip):
            response = probe_port(ip, port)
            if response == 1:
                 print ("Filtered | Stateful firewall present")
            elif response == 0:
                 print ("Unfiltered | Stateful firewall absent")
    else:
        print ("Host is Down")
```

LanScan

LanScan is a Python 3 module that helps to scan a given local network. It can list all devices and their open ports. LanScan also helps to get the information about network interfaces and networks.

Getting ready

We can install `lanscan` using `pip`:

```
pip3 install lanscan
```

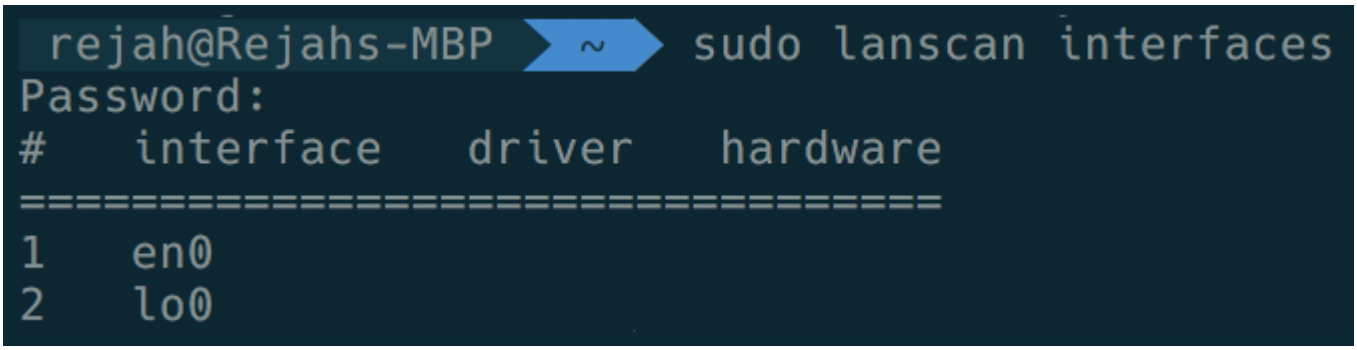
How to do it...

Here are some use cases for LanScan:

1. LanScan has some options that we can use for scanning the LAN. To get the details about the available interfaces in the system, we can use the `interfaces` option:

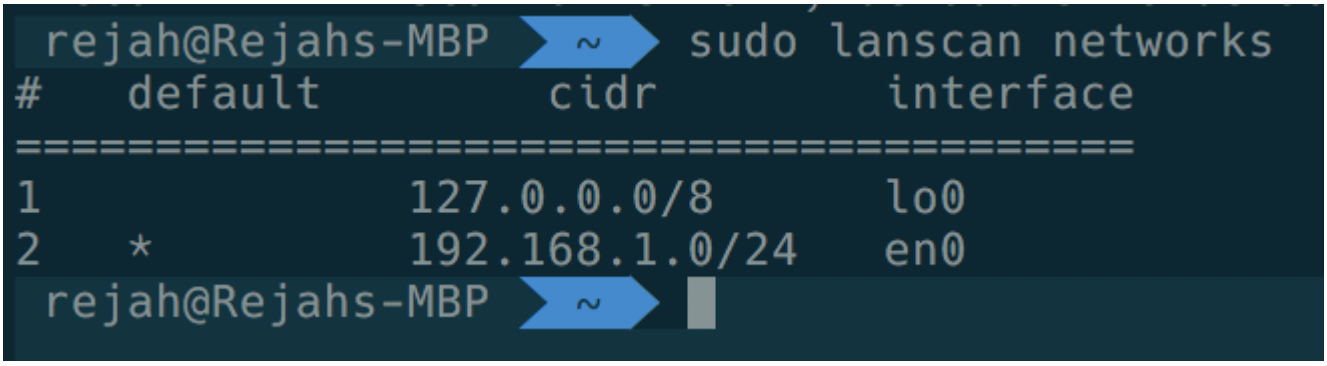
```
sudo lanscan interfaces
```

This will print the available interfaces, as follows:



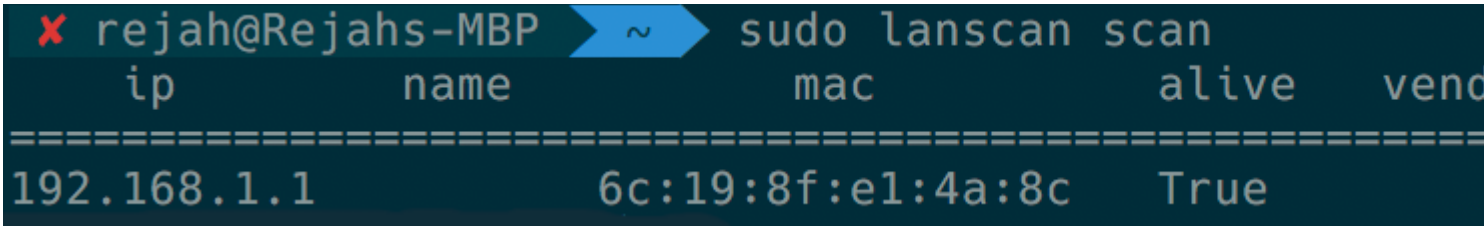
2. We can get the list of connected networks by using the network command:

```
sudo lanscan networks
```



3. We can start a local network scan from the terminal window. It requires admin privileges:

```
sudo lanscan scan
```



Here it will list the IP addresses in the LAN network and the open ports in each system