

# Network Scanner with Python

By

Sadia Parvin Ripa

Email: [sadiaripa504@gmail.com](mailto:sadiaripa504@gmail.com)



## INTRODUCTION:

A network scanner is an important tool to analyze the users which are available on network. Mainly, network scanner is an IP scanner used to scan networks that are interconnected to many devices. One big tool for research is the network scanner in the penetration test or a basic analysis. Because the hosts on the local network are available. There are plenty of resources available, but it is a best to write our own research and penetration testing tools based on what you want to do. In this project we can easily scan our network. So, in this project we are worked on –

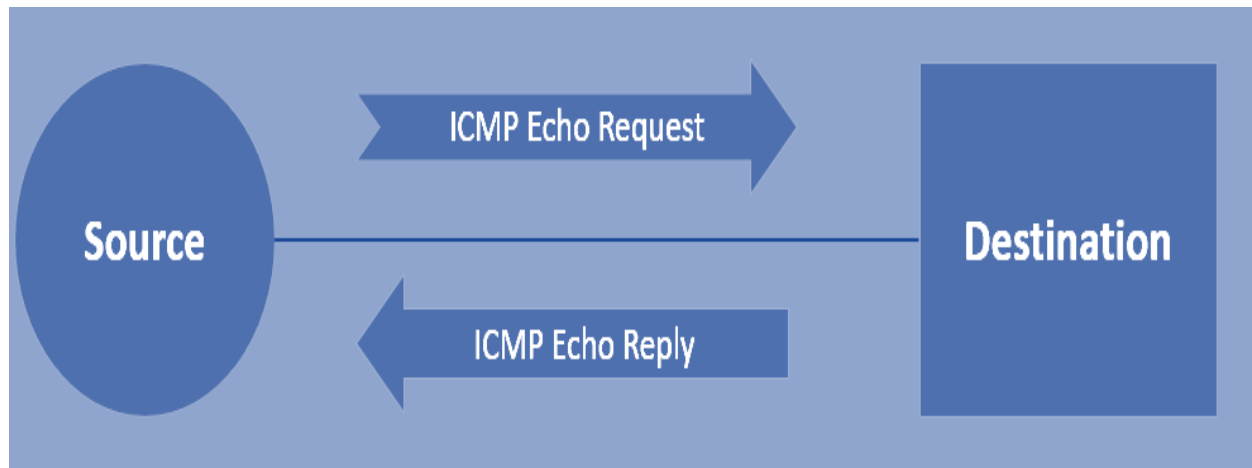
- Port Scanner Using Socket
- Port Scanner Using IPy
- Port Scanner Using Threading for efficiency
- Port Scanner Using ICMP
- Port Scanner Using TCP
- Network Scanner Using Scapy
- WIFI Scanner

Port scanning is a monitoring technique that is used on a specific host to identify the particular ports are accessible. This technique can be used by a network administrator, penetration tester or hacker. To get maximum information from the target device, we can configure the port scanner as per our requirements. A port scanner's concept is to run through a list of ports, to test whether they are available or not. With the port scanner, we will try to connect and do nothing else at different ports. If we're able to connect to open ports, then at least we know the port is open. For hackers and penetration testers, this is a type of "reconnaissance". Using IPy gives more faster result than using just socket module. But it is also time consuming.

Because of slow process of port scanner, we have to use threading. threading makes it possible to run various parts of the program simultaneously and can simplify design. it also helps speed up the program using threads. The threads can run on various processors, but they will run one at a time. it is a time saver process so improve the speed we can use threading.

The ICMP packet is sent through an IP address to a client. If the ICMP is received, the host is in online and the signals are received. ICMP is not a port scan. it is used to verify whether the host is up by pinging the remote host. When we have to search a number of hosts on a network, this

scan is useful. It includes sending an ICMP ECHO request to a host, and it will return an ICMP ECHO reply if that host is live. It is also known 'ping command'.



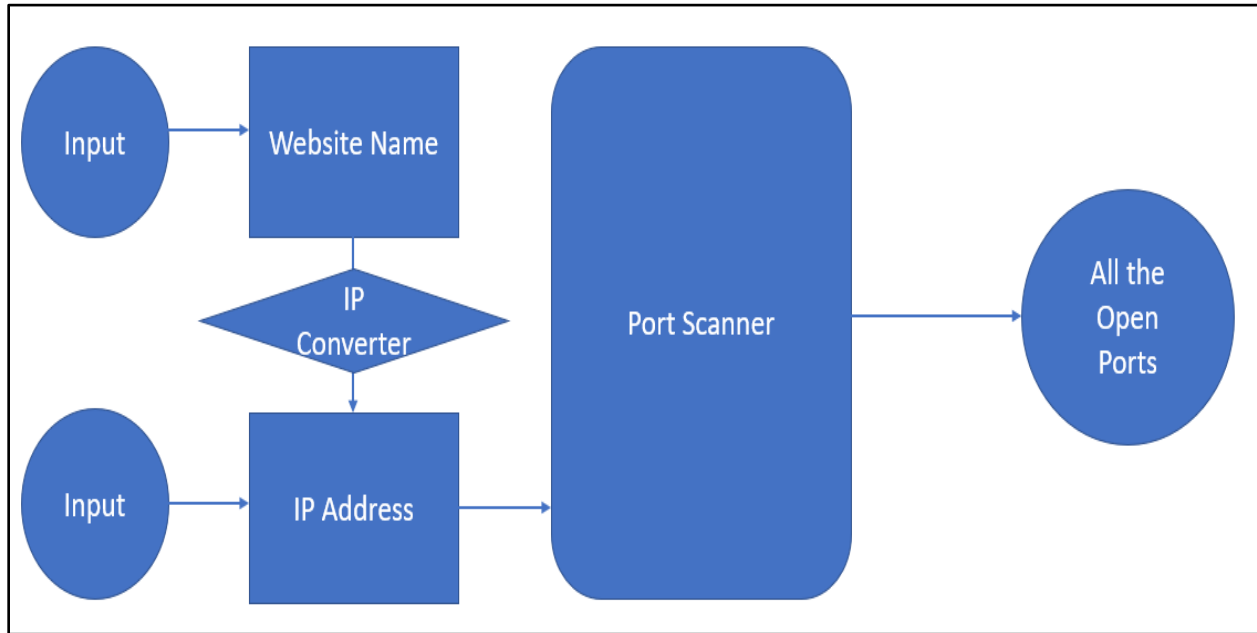
TCP connection, we have to perform a three-way handshake. We have to guess ports that could be available or open. This method is pre-assumed by the available ports of the hosts on the networks and we have to guess which port is open or not. The ports vary according to which operating system we are using. For Linux we can use 20, 21, 22, 23, 25, 80, 111, 443, 445, 631, 993 and 995 port. In windows operating system 135, 137, 138, 139 and 445 ports are using.

In three-way handshaking, our first step the client sent the Packet with SYN over the IP address to server. In the second step, server returns a packet with SYN and ACK. Thirdly, the SYN/ACK is obtained from the server by the client node and responds with an ACK packet.

An IP address is allocated from a pool of ISP managed addresses when a system attempts to connect to the Internet. Every time user launches a website, web app or email, user's behavior is monitored and registered using this numerical data as an identifier. It can be used to allow or block particular servers and services. It will scan all the WIFI network around our device.

## FRAMEWORK:

Our Port scanner will take the user input for both website and IP address. If we give a website name then our tools will convert the website address into a IP address and then it will scan all the ports.



**Diagram: Working of Port Scanner Tools**

We see in the diagram that all the open ports are on output section. But that not means that only open ports we scan. For simplicity we only print the open ports, otherwise it will hard to find from the output that which ports are open.

TCP scan all the live ports and our WIFI scanner scan all the network connected with our WIFI.

## PROCESS:

### Port Scanner using socket:

For port scanning tools we import socket and time module. We import time module to calculate the total time of the execution of the code. In our network programming course, we use socket module most of the time.

### Source Code with Description:

```
from socket import *
import time
startTime = time.time()

if __name__ == '__main__':
    target = input('Enter the host to be scanned: ')
    target_IP = gethostbyname(target)
    print('Started scanning on host: ', target_IP)

    for i in range(50, 500):
        s = socket(AF_INET, SOCK_STREAM)

        conn = s.connect_ex((target_IP, i))
        if(conn == 0) :
            print('Port %d: OPEN' % (i,))
        s.close()

    print('Time needed:', time.time() - startTime)
```

We take our target host as user input this host will convert into a IP address and if we give IP address then it will be same. Here, I scanned ports from 50 to 500. Here we made a socket instance and passed it two parameters. The first parameter is **AF\_INET** and the second one is **SOCK\_STREAM**. **AF\_INET** refers to the address family ipv4. The **SOCK\_STREAM** means connection-oriented TCP protocol. Now we can connect to a server using this socket. If connection is 0 that means port is open. Then we simply print all the open ports.

## Port Scanning Using IPy:

Port scanner with IPy is almost same as before. In this code we have to import a extra module IP from IPy. IPy can transform an IP address into a network address.

### **Source Code with Description:**

```
import socket
from IPy import IP
import time

start_time=time.time()

def scan(target):
    converted_ip = check_ip(target)
    print('\n' + '[scanning target]' + str(target))
    for port in range(1, 500):
        scan_port(converted_ip, port)

def check_ip(ip):
    try:
        IP(ip)
        return ip
    except ValueError:
        return socket.gethostbyname(ip)

def get_banner(s):
    return s.recv(1024)
```

At first, we define a function named scan with one parameter target. Inside this function we define a variable converted\_ip=check\_ip(target) that means it checks the IP of the single target. Then we use print statement. We did not fixed the port number that's why we use str(port). Then we use for loop for iterate certain amount of port. Then we call scan\_port(converted\_ip,port).Then we define another function check\_ip with parameter IP. Then we use try and exept.try and except rule

basically means that we are going to try this in case it doesn't work we are going to try something else which is under the except art. Inside this function we use IP module to convert the ip address and if we manage then return IP. If it manages to convert ip that means it is successful and specify the actual IP address to the target. If it gets value error that means the user specify the domain name and return socket.gethostbyname(ip).

Then we did another thing which is banner on open ports to discover which service is or which software is running on this open ports. Then we define function named banner with parameter s and inside it we return s.recv(1024) that we receive something from the port.

```
def scan_port(ipaddress, port):
    try:
        sock = socket.socket()
        sock.settimeout(0.5)
        sock.connect((ipaddress, port))
        try:
            banner=get_banner(sock)
            print('open port' + str(port) + ':' + str(banner))
        except:
            print('open port' + str(port))
    except:
        pass

if __name__ == '__main__':
    targets = input('enter the targets to scan:(split multiple targets with,):')

    if ',' in targets:
        for ip_add in targets.split(','):
            scan(ip_add.strip(' '))
    else:
        scan(targets)

end_time=time.time()
print("Total time need", end_time-start_time)
```

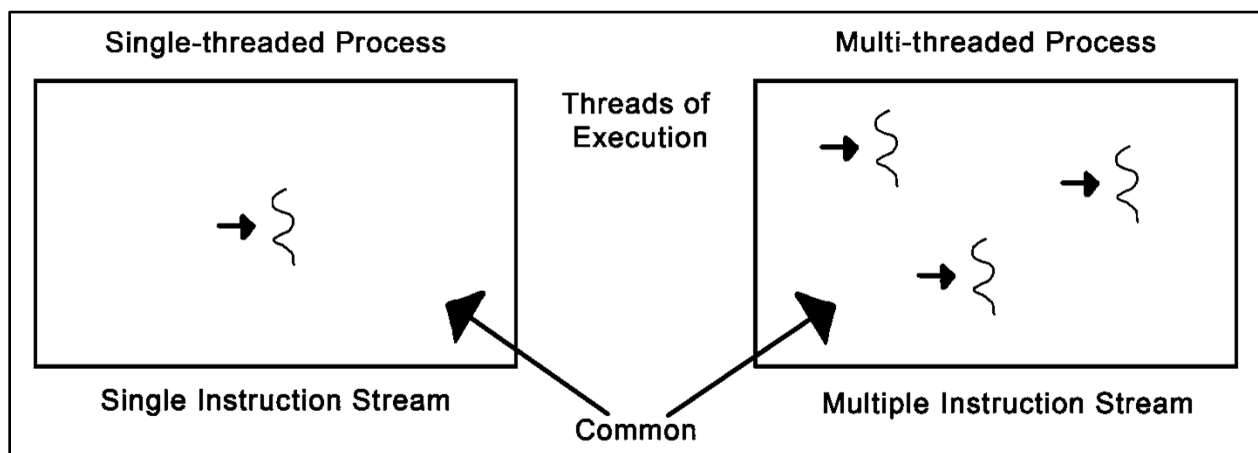
Then we create our main function `scan_port` with parameter `ipaddress` and `port`. Inside this function we use `try` and `except`. This function simply allows us to scan port. what they will do it will scan the port and define whether it is open or closed. Another thing we need to is we don't predetermine the port number so we remove the port number and also change the command in this code inside the function. In the print statement we add `str(port)`. First, we create socket object and assign it in `sock` variable also set timeout. `Socket.connect` means we are trying to connect to the port. Inside this `try` rule we use another `try` rule and use `get_banner` function with `sock` parameter. Then we print the open port and banner. If it is not work, we use `except` and print simply the open port. If this whole code is not work, we simply use `pass`.

We also take user input to know the target. Then we use `if` statement. In the `if` statement we use comma that means user try to specify multiple targets. Then we use `for` loop to split the target that means we want to split the IP address then we use `strip`. In the `else` statement we scan targets that means we call the scan function.

### **Port Scanner using threading:**

The above port scanner is quite slow. Using IPy port scanner becomes fast but still not that much fast. So, we use threading to make the port scanner faster.

For this we have to install the module `threading`. Threading can execute multiple command on a same time. Like it will scan multiple ports in a same time. So, it will take less time than other module or method.



**Diagram: Single Thread and Multi Thread process**



### Source Code with Description:

Here we see another new module queue. The queue module implements multi-producer, multi-consumer queues. It is especially useful in threaded programming when information must be exchanged safely between multiple threads. The Queue class in this module implements all the required locking semantics.

```
import socket
import time
import threading

from queue import Queue
socket.setdefaulttimeout(0.25)
print_lock = threading.Lock()

target = input('Enter the host for scanning: ')
target_IP = socket.gethostbyname(target)
print('Started scanning on host: ', target_IP)

def portscan(port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        connection = s.connect((target_IP, port))
        with print_lock:
            print(port, 'is open')
        connection.close()
    except:
        pass
```

We are using the thread locking concept, **thread\_lock = threading.Lock()** to avoid multiple modification at a time. We define one threader () function that will fetch the work (port) from the

worker for loop. Then the portscan() method is called to connect to the port and print the result. The port number is passed as parameter. Once the task is completed the q.task\_done() method is called. Rest of the procedure is almost same as the previous scanner.

```
def threader():
    while True:
        worker = q.get()
        portscan(worker)
        q.task_done()

q = Queue()
startTime = time.time()

for x in range(100):
    t = threading.Thread(target=__threader)
    t.daemon = True
    t.start()

for worker in range(1, 500):
    q.put(worker)

q.join()
print('Taken time:', time.time() - startTime)
```

### **Port Scanner Using ICMP:**

ICMP is not a port scan but it is used to ping the remote host to check if the host is up. Like in the command prompt of our operating system where there is a ping option. ICMP is almost similar. That's why it also called as ping sweep.

### **Source Code with Description:**

For this tool we import OS, Platform and Datetime module. The platform module includes tools to see the platform's hardware, operating. system, and interpreter version information where the program is running. The OS module in python provides functions for interacting with the operating system. Datetime module for importing date and time.

It first selects the range of IP address to ping sweep scan by splitting it into parts. This is followed by using the function, which will select command for ping sweeping according to the operating system, and last it is giving the response about the host and time taken for completing the scanning process.

```

import os
import platform

from datetime import datetime
net = input("Enter the Network Address: ")
net1= net.split('.')
a = '.'

net2 = net1[0] + a + net1[1] + a + net1[2] + a
st1 = int(input("Enter the Starting port Number: "))
en1 = int(input("Enter the Last port Number: "))
en1 = en1 + 1
oper = platform.system()

if (oper == "Windows"):
    ping1 = "ping -n 1 "
elif (oper == "Linux"):
    ping1 = "ping -c 1 "
else :
    ping1 = "ping -c 1 "
t1 = datetime.now()
print ("Scanning is in Progress:")

for ip in range(st1,en1):
    addr = net2 + str(ip)
    comm = ping1 + addr
    response = os.popen(comm)

    for line in response.readlines():
        if(line.count("TTL")):
            break
        if (line.count("TTL")):
            print (addr, "--> Live")

t2 = datetime.now()
total = t2 - t1
print ("Scanning completed...: ",total)

```

### Port Scanner Using TCP:

For TCP scan we import two module socket and datetime. But it's working process is different than port scanner using socket. This is actually three-way handshaking.



if we turn off the ICMP ECHO reply feature or using a firewall to ICMP packets then ping sweep scanner will not work and we need TCP scan.

### Source Code with Description:

We take a user input which is the IP address. We don't need to scan all the ports. So, user can define the port range as they want. IP address is split by '.'. Defining the time, we start our scanning. It selects the range of IP address to ping sweep scan by splitting it into parts. This is followed by using a function for scanning the address, which further uses the socket. Later, it gives the response about the host and time taken for completing the scanning process.

The result = s. connect\_ex((addr,135)) statement returns an error indicator. The error indicator is 0 if the operation succeeds, otherwise, it is the value of the errno variable. Here, we used port 135; this scanner works for the Windows system. Another port which will work here is 445 (Microsoft-DSActive Directory) and is usually open.

```
import socket
from datetime import datetime
net = input("Enter the IP address: ")
net1 = net.split('.')
a = '.'

net2 = net1[0] + a + net1[1] + a + net1[2] + a
st1 = int(input("Enter the starting port number: "))
en1 = int(input("Enter the last port number: "))
en1 = en1 + 1
t1 = datetime.now()
```

```
def scan(address):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    socket.setdefaulttimeout(1)
    result = s.connect_ex((address, 135))
    if result == 0:
        return 1
    else:
        return 0
```

```
def run1():  
    for ip in range(st1,en1):  
        address = net2 + str(ip)  
        if (scan(address)):
```

```
run1()  
t2 = datetime.now()  
total = t2 - t1  
print_ ("Scanning completed in: ", total)
```

### **Network Scanner Using Scapy:**

For network scanner we import scapy module. Network scanning refers to scanning of whole network to which we are connected and try to find out what are all the clients connected to our network.

**Scapy:** is a library supported by both Python2 and Python3. It is used for interacting with the packets on the network. It has several functionalities through which we can easily forge and manipulate the packet.

**ARP ():** This function defined in scapy module which allows us to create ARP packets (request or response). By default, if we are calling it, it will create an ARP request packet for us.

**summary ():** This method provides us the status of the packet that we have created. It does not provide the detailed information about the packet, it just gives us the basic idea like what is the type of packet, what is the destination of the packet etc.

**show () method:** This method is very similar to summary () method. It gives more detailed information about the packet. The usage of this function is also much similar to as summary () method.

### Source Code with Description:

First, we import scapy module then in request we call ARP method. In the print statement we print the summary of the packets then print the full packet details using show method. We determine the destination and then broadcast packet using Ether method. Combine ARP request packet and Ethernet frame using '/'. Send this to your network and capture the response from different devices.

```
import scapy.all as scapy

request = scapy.ARP()
print(request.summary())
print(request.show())

request.pdst = '192.168.0.1/24'
broadcast = scapy.Ether()

broadcast.dst = 'ff:ff:ff:ff:ff:ff'

request_broadcast = broadcast / request
clients = scapy.srp(request_broadcast, timeout = 1)[0]
for element in clients:
    print(element[1].psrc + " " + element[1].hwsrc)
```

### WIFI Scanner:

The WIFI Scanner will show us all the device connected under our WIFI network. It is very useful tool.

### Source Code with Description:

For this tool we import only one module called subprocess. The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

Netsh is a command-line scripting utility that allows you to display or modify the network configuration of a computer that is currently running.

WLAN commands that allow you to access the Wi-Fi profiles.

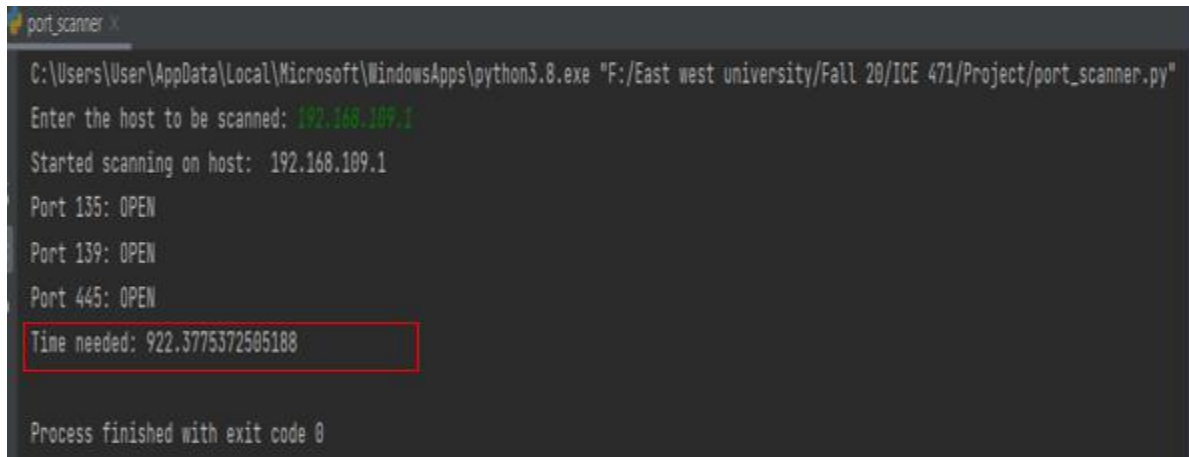
Network the unique name of wireless network

```
import subprocess
devices=subprocess.check_output(['netsh', 'wlan', 'show', 'network'])
devices=devices.decode('ascii')
devices=devices.replace("\r", "")
ssid=devices.split("\n")
ssid=ssid[4:]
ssids=[]
x=0
y=0
while x<len(ssid):
    if x%5 ==0:
        ssids.append(ssid[x])
    x+=1
while y< len(ssids):
    print(ssids)
    y+=1
```

## RESULT:

### **Port Scanner Using Socket:**

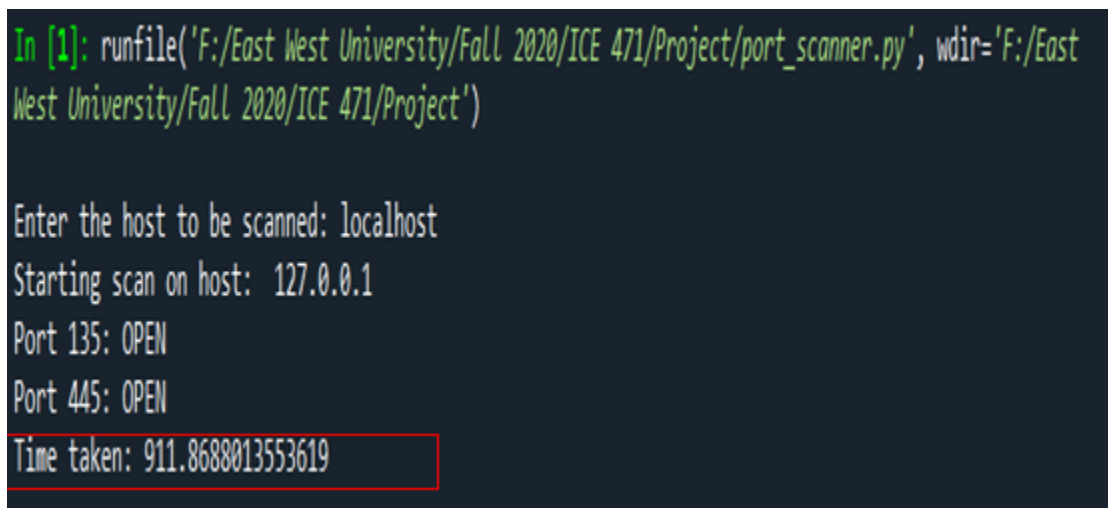
When we use socket module to port scanning it take one user input that is host name. We give host IP address and it will show us all the open ports of that host.



```
port_scanner X
C:\Users\User\AppData\Local\Microsoft\WindowsApps\python3.8.exe "F:/East west university/Fall 20/ICE 471/Project/port_scanner.py"
Enter the host to be scanned: 192.168.109.1
Started scanning on host: 192.168.109.1
Port 135: OPEN
Port 139: OPEN
Port 445: OPEN
Time needed: 922.3775372505188
Process finished with exit code 0
```

There is only one problem of using socket port scanner. This port scanner is slow and take too much time to execute. So, using this is little bit difficult.

Here we see another output of the program where instead of IP address we use host name then it covert it into IP address and give us the output.



```
In [1]: runfile('F:/East West University/Fall 2020/ICE 471/Project/port_scanner.py', wdir='F:/East
West University/Fall 2020/ICE 471/Project')

Enter the host to be scanned: localhost
Starting scan on host: 127.0.0.1
Port 135: OPEN
Port 445: OPEN
Time taken: 911.8688013553619
```



## Port Scanner Using IPy:

Port Scanner with just socket is very time consuming. So, we can use IPy module along with the socket module. It will take less than one third time from socket port scanner. So, using it give better efficiency.

```
C:\Users\User\AppData\Local\Microsoft\WindowsApps\python3.8.exe "F:/East west university/Fall 20/ICE 471/Project/portscanner.py"
enter the targets to scan:(split multiple targets with,):127.0.0.1

[scanning target]127.0.0.1
open port135
open port445
Total time need 266.07290482521057

Process finished with exit code 0
```

## Port Scanner Using Threading:

Though using IPy take less time than using socket but still it's not time efficient. In real we have to scan lots of port from multiple hosts then it will take long time. So, threading is the best solution. Using threading we can scan ports within very short time. Here, we can see the output that it takes almost like 1.5 second to finished the code.

So, using threading is the best solution.

```
threaded_scanner x
C:\Users\User\AppData\Local\Microsoft\WindowsApps\python3.8.exe "F:/East west university/Fall 20/ICE 471/Project/threaded_scanner.py"
Enter the host for scanning: 192.168.109.1
Started scanning on host: 192.168.109.1
139 is open
135 is open
445 is open
Taken time: 1.4064655303955078

Process finished with exit code 0
```

## Port Scanner Using ICMP:

ICMP port scanning actually work as a ping command of command prompt. The output is showing no live ports because the firewall is on and ICMP inbound settings are disabled too. After changing these settings, we can get the list of live ports in the range from 100 to 200 provided in the output.

```
In [1]: runfile('D:/python code/untitled0.py', wdir='D:/python code')  
  
Enter the Network Address: 127.0.0.1  
  
Enter the Starting port Number: 100  
  
Enter the Last port Number: 200  
Scanning is in Progress:  
Scanning completed...: 0:00:04.943467
```

## Port Scanner Using TCP:

If we turn off the ICMP ECHO reply feature or using a firewall to ICMP packets then ping sweep scanner will not work and we need TCP scan. For this program we have to give the IP address and range of the port number then we can see all the live TCP connection.

The simplest port scanners use the operating system's network functions and are generally the next option to go to when SYN is not a feasible option.

```
tcp_scanner x  
C:\Users\User\AppData\Local\Microsoft\WindowsApps\python3.8.exe "F:/East west university/Fall 20/ICE 471/Project/tcp_scanner.py"  
Enter the IP address: 127.0.0.1  
Enter the starting port number: 100  
Enter the last port number: 200  
127.0.0.100 is live  
127.0.0.101 is live  
127.0.0.102 is live  
127.0.0.103 is live  
127.0.0.104 is live  
127.0.0.105 is live  
127.0.0.106 is live  
127.0.0.107 is live  
127.0.0.108 is live  
127.0.0.109 is live  
127.0.0.110 is live  
127.0.0.111 is live  
127.0.0.112 is live  
127.0.0.113 is live  
127.0.0.114 is live  
127.0.0.115 is live  
127.0.0.116 is live  
127.0.0.117 is live  
127.0.0.118 is live  
127.0.0.119 is live  
127.0.0.120 is live  
127.0.0.121 is live
```

```
tcp_scanner x
127.0.0.179 is live
127.0.0.180 is live
127.0.0.181 is live
127.0.0.182 is live
127.0.0.183 is live
127.0.0.184 is live
127.0.0.185 is live
127.0.0.186 is live
127.0.0.187 is live
127.0.0.188 is live
127.0.0.189 is live
127.0.0.190 is live
127.0.0.191 is live
127.0.0.192 is live
127.0.0.193 is live
127.0.0.194 is live
127.0.0.195 is live
127.0.0.196 is live
127.0.0.197 is live
127.0.0.198 is live
127.0.0.199 is live
127.0.0.200 is live
Scanning completed in: 0:00:00.743615
Process finished with exit code 0
```

### Network Scanner Using Scapy:

It will show IP address along with the MAC address. Then it broadcast packets.

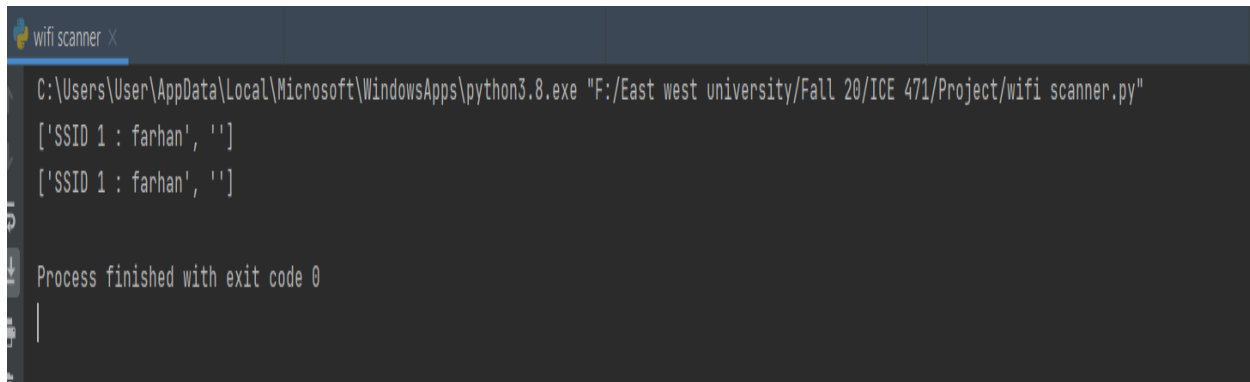
```
In [9]: runfile('F:/East West University/Fall 2020/ICE 471/Project/broadcasting.py',
West University/Fall 2020/ICE 471/Project')
ARP who has 0.0.0.0 says 192.168.0.109
###[ ARP ]###
  hwtype   = 0x1
  ptype    = IPv4
  hwlen    = None
  plen     = None
  op       = who-has
  hwsrc    = f8:34:41:77:36:4d
  psrc     = 192.168.0.109
  hwdst    = 00:00:00:00:00:00
  pdst     = 0.0.0.0

None
Begin emission:
Finished sending 256 packets.

Received 261 packets, got 3 answers, remaining 253 packets
192.168.0.1      70:4f:57:22:0c:b6
192.168.0.101    4c:02:20:11:6c:f9
192.168.0.100    70:0b:c0:bb:5b:e0
```

## WIFI Scanner:

This show us the WIFI information and username of all currently online users.



```
wifi scanner x
C:\Users\User\AppData\Local\Microsoft\WindowsApps\python3.8.exe "F:/East west university/Fall 20/ICE 471/Project/wifi scanner.py"
['SSID 1 : farhan', '']
['SSID 1 : farhan', '']
Process finished with exit code 0
```

## CONCLUSION:

Network Scanner is very important tool. As a network engineer we should have the basic knowledge of it. In our lab and classes, we learnt about this type of tools and worked with this type of module. With the knowledge of our network programming, we understand this tool and work with it easily.

**If** we want to investigate any remote or local network for security considerations, we need port scanner. Used by programmers and network administrators, these **port scanners** are used to mainly know what **ports** are being used by what applications, so we can identify malicious programs right away. We see port scanners with different module and see that using threading is the best option for port scanning. Because it needs less time to execute the program. Then we do ICMP scanning, TCP scanning, network scanning with scapy and broadcast packets. We also make a WIFI scanner tool.