ECE303 – Communication Network
TCP Port Scanner Project
Jongoh (Andy) Jeong
March 26, 2019

Table of Contents

Purpose

- This project deals with a simple TCP port scanner, which is a program that finds active(open) ports on a target host server by making client requests to a specific port. This was conducted under the assumption that the host follows RFC 793 (TCP) and the services are running on their default port addresses. In addition to detecting which port is open and what services it is used for, the program retrieves the TCP window size and IP TTL size from a received packet from the hosts of a number of ports, and tries to fingerprint the Operating System it runs on.

Usage

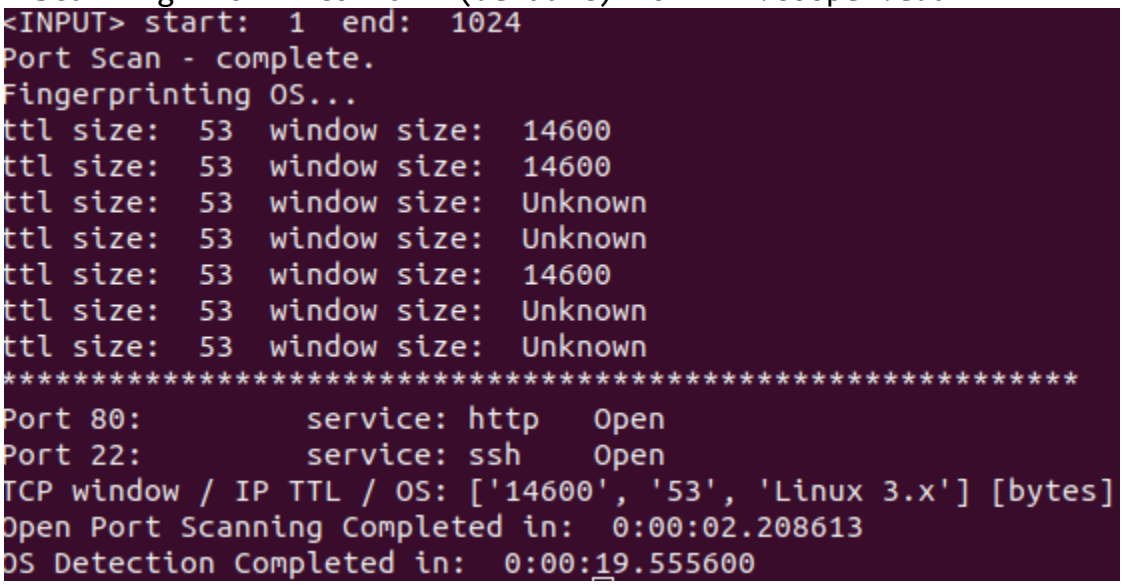
```
   USAGE:  python3 portscanner_osdetection.py [hostname] [-p 15:25]
              • It should work with the available version of python
                on the machine with the required libraries
   INPUT:  An IPv4 address (or hostname) and port range(optional;
           default: 1:1024)
FUNCTION:  1. [Port Scan]
           Enter an IP address and [optionally] a port (range), and
           the program will go through ports in the range, and show
           which of the port(s) is(are) open on the specified host by
           making a socket connection.

           2. [Fingerprinting OS]
           The program will go through a list of commonly open ports
           (as specified in the code) and check the OS type based on
           the characteristics of a successfully received packet
           through that port. If not successfully received, then it
           returns 'Unknown.'
  OUTPUT:  1. Status of OPEN ports [Port #: [OPEN]]
           2. For each commonly open ports,
                 [IP TTL Size / TCP Window Size / estimated OS type]
```

- Operating System fingerprinting is based on the set of known TCP window and IP TTL sizes for the following systems.

Operating System (OS)	IP Initial TTL [bytes]	TCP Window Size [bytes]
IOS 12.4 (Cisco Router)	255	4128
Google Linux	64	5720
Linux 2.4	64	5840
Linux 3.X	64	14600
Windows 7	128	8192
Windows Server 2003	128	16384
OpenBSD	64	16384
Windows XP	128	65535
FreeBSD	64	65535
Unknown	Unknown	Unknown

Experimental Output

- Scanning from 1 to 1024 (default) for www.cooper.edu



Code Listing

1. Selects starting and ending port based on the input arguments
2. Insert (host, port) pair into a Queue, from which each thread is created. This thread extends from Thread library, and opens a socket for the specified port. Upon successful connection through the socket, it returns a list of open ports.
3. The threads are joined, and each port output is printed.
4. For OS fingerprinting, it returns the system platform if localhost; if else, it sends a TCP packet with destination port and SYN flag, and upon successfully received packet, it returns its TCP window size and IP TTL size. After making a

set of packet requests, it finally prints the OS information from the lastly received packet, which is assumed to be its final estimate.

5. Necessary libraries: socket, subprocess, datetime, argparse, threading, queue, scapy(OS detection), logging, sys, os
6. Contingencies (possible failure): may require libgcc_s.so.1 for pthread_cancel to work

Jeong portscanner osdetection.py

```
# Author: Jongoh (Andy) Jeong
# Course: ECE303 - Communication Network
# Title: Port Scanner
#
#!/usr/bin/env python

import socket
import subprocess
from datetime import datetime
import argparse
from threading import Thread
from queue import Queue

from scapy.all import * #IP, TCP, sr1
import logging
import sys
import os


# turn off scapy warnings
logging.getLogger('scapy.runtime').setLevel(logging.ERROR)
os_info = []
open_ports = []; open_services = []
service, send_buf, rcv_buf, ttl = '',-1,-1,-1

# -------------Scanning ports------------------------------

class Scan(Thread):
    def __init__(self):
        super(Scan, self).__init__()
        # Thread.__init__(self)

    # override Thread.run(self)
    def run(self):
        while (not queue.empty()):
            host, port = queue.get()
            if port == -1:
                self.open(host, port, 1)
            else:
                self.open(host, port, 0)

    def open(self, host, port, default):
        # Using the range function to specify ports (here it will scans all ports
between 1 and 1024)
```

```
        # error handling for catching errors
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                # print("scanning:%d" % port)
                s.settimeout(2)
                result = s.connect_ex((host, port))
                if result == 0:
                    open_ports.append(port)
                    # get IP TTL and TCP Window Size (send/receive) from socket
                    # TCP window sizes are not accurate measures
                    # print("Port {}: \t Open".format(port))
                    service = ' '.join((socket.getservbyport(port)).split())
                    service = 'UNKNOWN' if service == '' else service
                    # ttl = s.getsockopt(socket.IPPROTO_IP, socket.IP_TTL)
                    open_services.append(service)
                    # print("host: [%s] port: [%d] [OPEN] service: [%s] TTL size: [%d
bytes]" % (host,port,service,ttl), sep='\t')
                s.close()
        except KeyboardInterrupt:
            print("ERROR: Keyboard Interrupt. Exiting...")
            sys.exit()

        except socket.gaierror:
            # print("ERROR: %s:%s not found." % (host,port))
            sys.exit()

        except socket.error:
            # print("ERROR: server connection to host %s:%s fail." % (host,port))
            sys.exit()


# -------------OS Detection-------------------------------

# -------------Common default port numbers---------------
ports_to_check = [
    80,     # http
    22,     # ssh
    21,     # ftp
    135,    # dcom-smc
    139,    # netbios
    143,    # imap
    1723,   # pptp
    3389,   # rdp
    25,     # smtp
    23,     # telnet
    53,     # dns
    443,    # https
    110,    # pop3
    445,    # ms-ds
    8080,   # tomcat
    4567,   # filenail (commonly open port for backdoors)
]
# determine OS by window size and TTL
def os_detect(window_size, ttl):
    # return name for OS based on TCP window size and TTL
```

```
    dict = {
        '4128': lambda ttl: 'IOS 12.4 (Cisco Router)',
        '5720': lambda ttl: 'Google Linux',
        '5840': lambda ttl: 'Linux 2.4',
        '14600': lambda ttl: 'Linux 3.x',
        '8192': lambda ttl: 'Windows 7',
        '16384': lambda ttl: ('Windows Server 2003' if int(ttl)
                              > 64 else 'OpenBSD'),
        '65535': lambda ttl: ('Windows XP' if int(ttl)
                              > 64 else 'FreeBSD'),
    }
    # default value for case window_size not specified in dictionary
    value_not_found = lambda ttl: 'Unknown'
    return dict.get(window_size, value_not_found)(ttl)

def get_os(host, port):
    # requires root prvilege
    if not os.geteuid() == 0:
        sys.exit('No root privilege. Acquire root access for OS fingerprinting.')
    print("Fingerprinting OS...")

    # loop through common ports of the host
    for port in ports_to_check:
        rcv_pkt = None
        # Assemble IP packet
        ip = IP(dst = host)
        #Assemble TCP with destination port and SYN flag
        tcp = TCP(dport = port, flags = 'S')
        # send a packet through, wait 2 (=timeout) for response
        # print("[OS detection] checking packet for port %s..." % port)
        try:
            if rcv_pkt: # if response already received,
                break
            rcv_pkt = sr1(ip / tcp, timeout=2, verbose=0)
            if rcv_pkt is not None:
                # retrieve TCP window size, IP TTL from the rcv_pkt
                window_size = rcv_pkt.sprintf('%TCP.window%')
                ttl = rcv_pkt.sprintf('%IP.ttl%')
                # make window_size and ttl size 'Unknown' if not found
                window_size = 'Unknown' if window_size == '??' else window_size
                ttl = 'Unknown' if ttl == '??' else ttl
                print("ttl size: ", ttl, " window size: ", window_size, end='\n')
                os_type = os_detect(window_size, ttl)
                os_info.append([window_size, ttl, os_type])
        except OSError:
            print("ERROR: send + receive packet error. port: %s" % tcp.dport)
            break

if __name__=='__main__':
    # Clear the screen
    subprocess.call('clear', shell=True)

    # Argument parse
    parser = argparse.ArgumentParser(description='Function: Scanning ports of the
entered host.')
```

```
    parser.add_argument("host", type=str, help='Enter a host name(address) to
scan...')
    parser.add_argument("-p", "--port", default='1:1024', help='Enter ports
(inclusive if range). Default is 1:1024.')
    args = vars(parser.parse_args())
    host = args['host']
    port = args['port']
    if len(port) == 0:
        ports = -1
    else:
        char_colon = args['port'].find(':')
        if (char_colon is not -1):
            a,b = args['port'].split(':')
        else:
            a = args['port']
            b = a
        startport, endport = int(a), int(b)
        print("<INPUT> start: ", startport, " end: ", endport)
        if (startport > endport):
            sys.exit("ERROR: Starting port is higher than the ending port")
        ports = list(range(startport, endport+1))

    # make a queue of (host, port) pair
    queue = Queue()
    for port in ports:
        queue.put((host, port))

    # Check what time the scan started
    t1 = datetime.now()

    # make threads
    threadlist = []
    numThreads = int(endport - startport) + 1
    for i in range(numThreads):
        thread = Scan()
        thread.start()
        threadlist.append(thread)

    # join all threads
    try:
        for thread in threadlist:
            thread.join()
    except KeyboardInterrupt:
        print("ERROR: Keyboard Interrupt. Exiting...")
        sys.exit()
    print("Port Scan - complete.")

    # Record time it has taken for port scan
    t2 = datetime.now()

    # *****OS detection****
    # check if localhost
    foundOS = []
    if host == "localhost" or host == "127.0.0.1":
        # test if window and ttl sizes return for the localhost machine
```

```
        # in general, not likely, so will return 'Unknown'
        for port in open_ports:
            rcv_pkt = None
            window = 'Unknown'; ttl = 'Unknown'
            ip = IP(dst = host)
            tcp = TCP(dport = port, flags = 'S')
            if rcv_pkt: # if response already received,
                break
            rcv_pkt = sr1(ip / tcp, timeout=2, verbose=0)
            print(rcv_pkt)
            if rcv_pkt is not None:
                # retrieve TCP window size, IP TTL from the rcv_pkt
                window = rcv_pkt.sprintf('%TCP.window%')
                ttl = rcv_pkt.sprintf('%IP.ttl%')
        foundOS.append([window, ttl, sys.platform])
    else:
        get_os(host, port)

    # ****print results****
    print("*"*60)
    for i in range(len(open_ports)):
        print("Port {}: \t service: {} \t Open".format(open_ports[i],
open_services[i]))

    i = 0
    windowsize, ttlsize = 0, 0
    while (i < len(os_info)):
        if (os_info[i][2] is not 'Unknown'):
            foundOS.append(os_info[i])
        else:
            windowsize, ttlsize = os_info[i][0], os_info[i][1]
        i += 1

    if len(foundOS):
        print("TCP window / IP TTL / OS: {} [bytes]".format(foundOS[len(foundOS)-1]))
    else:
        print("TCP window / IP TTL / OS: {} [bytes]".format([windowsize, ttlsize,
'Unknown']))

    # Record time it has taken for fingerprinting OS
    t3 = datetime.now()
    totalThread =  t2 - t1
    totalOSDetect = t3 - t2
    print('Open Port Scanning Completed in: ', totalThread)
    print('OS Detection Completed in: ', totalOSDetect)
```