Detection of Operating System Types

(Final EECE658-Project Report)

Alem Fitwi
Electrical & Computer Egineering Department
State University of New York at Binghamton
(Ph.D. Student)
Binghamton, New York, USA
afitwi1@binghamton.edu

Abstract— **The Internet has amazingly revolutionized every aspect of human life. The revolution of the Internet can be traced back to the development of the Advanced Research Projects Agency Network (ARPANET) in the early 1970s by the Department of Defense of the USA based on which the Internet, the TCP/IP protocol, that has eventually become the most widely used network interconnection protocol, was developed. However, insufficient security concerns at the beginning of the design have left a lot of weakness that could be exploited by attackers to compromise computing systems. Particularly, the vulnerabilities of the Internet protocols and Technologies are widely exploited by both internal and external users and attackers. Hence, there has been a tremendously pressing quest for the design, development, and deployment of less-resource hungry, effective intranet security solutions that ensure secure, authentic, and authorized access to enterprise resources. The prime goal of this course project is to explore the preludes to the design and implementation of strong Intranet Security Management System. Specifically, Operating Systems Type detection, a part of an intranet security solution, will be implemented using such technologies as NMAP (Network Mapper) and a powerful python network module called Scapy. It focuses on the implementation of an algorithm that will function in a way similar to network mapping tool to detect the type and version of operating systems running on local area network (LAN) hosts where the Raspberry Pi is used to run the python script which is responsible for the detection of the OS via packet analysis, port scanning, port filtering, and based on predefined peculiar attributes of an OS like Time To Live (TTL) and Window Size. Plus, this Scapy based scanning is stealthy for its packets are not flagged unlike NMAP where the scanning packets are flagged and can be easily identified.**

Keywords— Internet, ARPANET, TCP/IP, Intranet security, Raspberry PI, security policies, TTL, NMAP, Scapy, & LAN

I. INTRODUCTION

The Internet has alarmingly revolutionized the way we communicate, the way we do business, and the way we live. The start of the revolution of the Internet can be traced back to the development of the Advanced Research Projects Agency Network (ARPANET) in the early 1970s by the Department of Defense of the USA. The ARPANET was an early packet switching network and the first network to implement the protocol suite TCP/IP, which later both technologies became the technical foundations of the Internet. In other words, along with the rapid development of the Internet, the TCP/IP protocol has become the most widely used network interconnection protocol [3]. However, due to insufficient security concerns during the design phase, the protocol has a lot of security risks. It is to be recalled that the Internet was first applied in a research environment for a few trusted user groups. Therefore, network security problems were not the major concerns at that time which has left big security holes in the TCP/IP protocol stacks. Some of the security weakness associated with each layer of the TCP/IP protocol suite are portrayed in figure 1. The vast majority protocols do not provide the necessary security mechanisms. Various studies show that many of the vulnerabilities of any network, be it Intranet, Extranet or Internet are compromised or affected by intentional or unintentional attacks directed from Terminals of internal users where the edge security devices like firewalls and antivirus alone can't solve [4, 5, 6, 7, 8, 9].

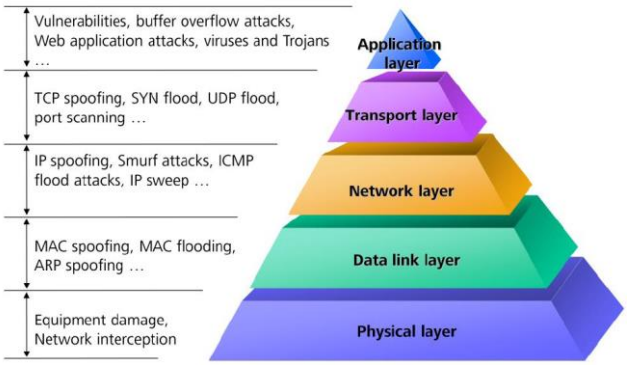

Figure 1: TCP/IP Common Security Problems

One of the solutions is building a strong Intranet management system capable of fully implementing an enterprise's security policy that can safeguard the intranet from possible attacks by enforcing Identity and security checks or authentications. This project focuses on one of the major preludes or prerequisite for strong Intranet security solution– the detection of operating systems and their versions. One of the major functions that any Intranet security solution should possess is security authentication. Security Authentication refers to the process of detecting the type of OS running in hosts along with their versions, and scanning for vulnerabilities so as to recommend timely correct patching and version updates. This is the main push

factor that aroused my interest to work on "Operating Systems Type Detection" as part of my Hardware Based Security course project hoping that it will serve as a stepping-stone or prelude to some of my future research works.

In this project work, a review of many related papers and technologies was made. Then, in what ensues, five more sections are presented. Section II briefly presents the survey results, Section III presents the main objectives of the project, Section IV explores the major tasks and the results produced. Then, section V briefly outlines the possible future works that could be done as extension of what has been done in this project followed by a conclusion presented in section VI.

II. Literature Survey

I have made a survey on how operating systems are differentiated from one another in the eyes of NMAP and the Scapy python module for network security. What is more, I have tried my level best to explore how the NMAP and the Scapy operate, and what special features they comprise. The major surveys I have accomplished are tersely presented in what ensues in three sub-sections.

A. Unique Attributes of Operating Sysetms

I have made a thorough survey in search of the attributes or features that could be used to distinguish one operating system from another. So far, I have found out that operating systems could be identified based on some contents of the packet they send. Specifically such attributes of a packet as TTL values and Window size could be employed to identify what type of an operating system a given network host is running

Time to live (TTL) or hop limit is a mechanism that limits the lifespan or lifetime of data in a computer or network. TTL could be implemented as a counter or timestamp attached to or embedded in the data. Once the prescribed event count or timespan has elapsed, data is discarded. In computer networking, TTL prevents a data packet from circulating indefinitely. In computing applications, TTL is used to improve performance of caching. TCP window size is one of the most popular options for network troubleshooting or performing an application baseline. The TCP window size, a.k.a. the TCP receiver window size, is simply an advertisement of how much data (in bytes) the receiving device is willing to receive at any point in time. The receiving device can use this value to control the flow of data, or as a flow control mechanism. Some operating systems will use a multiple of their maximum segment size (MSS) to calculate the maximum TCP window size [10].

Table 1: OS Distinguishing Features/Attributes

S/N	***Operating System***	***TTL***	***Window Size***
1	Chrome OS/Android	64	5720
2	CISCO IOS 12.4	255	4128
3	FreeBSD	64	65535
4	Linux Kernel 2.x	64	5840
5	Linux Kernel 3.x	64	14600
6	MAC OS	64	65535
7	Windows 2000/ Server 2003	128	16384
8	Windows 7/Vista/ Server 2008	128	8192
9	Windows XP	128	65535

Generally speaking, Operating systems of different vendors and architecture might have different TTL values and different window sizes the combination of which could be used as a distinguishing OS signature. Table 1 lists the various types of operating systems along with their respective TTL values and TCP window sizes, showing that the combination of the two is different for most of the operating systems with different founding architectures. Putting it another way, apart from FreeBSD and MAC OS, which are said to have the same founding architecture, all other operating systems have unique combination of TTL and Window size.

B. Network Mapper (NMAP)

NMAP stands for Network Mapper. It is free, open source security scanner for auditing network that runs on most platforms and was originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich). When it is aimed at a particular host, NMAP can determine which ports are open, which operating systems and versions are running, what services are offered and what firewalls are being used [1,2].

Though NMAP is a powerful Network Discovery Tool, the output might not make sense to us human unless some apt analysis and processing is done on the discovered packets. Usually the output is presented in the form of lengthy or list (a data structure type) comprising a number of tuples as elements. Hence, the knowledge of Python data structures like Dictionary, Lists, Tuples, and the corresponding methods and functions coupled with Scapy features is extremely vital to successfully unpack packets and extract useful/insightful information. Regular Expressions are also of paramount importance in analyzing packets. Table 2 depicts some of the commonest NMAP commands [1, 2].

Table 2: NMAP commands

S/N	***NMAP Command***	***Brief Description***
1	NMAP -sP	Ping scans the network, listing machines that respond to ping
2	NMAP –p –sV -sS	Full TCP port scan using service version detection
3	NMAP -sS	TCP Sync Scan
4	NMAP -F	Fragment packets

		(optionally w/ given MTU)
5	NMAP -sV	Probe open ports to determine services/version information
6	NMAP –O	Enable OS Detection
7	NMAP -sn	Ping scan - Disable port scan
8	NMAP –T 0-5	Set timing template-higher is faster (less accurate)

Furthermore, NMAP can be broken into a number of classes based on functionalities. Some of the NMAP commands enable us to detect the type of operating system running in target hosts, as depicted in table 2, and others help us determine the type and version of services running in the target host. Table 3 portrays the various classes or functionalities of NMAP. Target Specification NMAP commands help us specify the targets to be scanned in a file named ***name.file*** or to make a random scan. Host Discovery is another functionality of NMAP that determines whether the target host is alive or dead/off using simple ICMP ping scan by disabling ports for it doesn't care about what services are running in the target host. All functionalities of NMAP are listed in table 3 with examples [2].

Table 3: Various functions of NMAP

S/N	***NMAP Functions***	***Examples***
1	Target Specification	-iR: choose random target
2	Host Discovery	-sn: ping scan
3	Scan Techniques	-sS: TCP Syn connect
4	Port Specification	-p: scan specified ports
5	Service/Version Detection	-sV: probe versions
6	Script Scan	-sC: equivalent to script
7	OS Detection	-O: enable OS detection
8	Timing and Performance	-T: set timing
9	Firewall/IDS Evasion and Spoofing	-S: spoof source address
10	Output	-V: increase verbosity
11	MISC	-6: enable IPv6

C. SCAPY

It is an interactive packet manipulation tool with which you can capture the code, analyze, create, and send network packet. Scapy is flexible that lets you create any packet you can think of and send it on the wire even if it is invalid packet. You can also use it to create or build network Scanners, Sniffers, or even attack tools for ARP poisoning or DHCP starvation [1]. Of course, we should build such kind of utilities for learning and testing purposes only as an ethical or white hacker. It has plenty of useful methods like lsc() that lists the commands supported by Scapy, ls() that outputs long list of network protocols supported by Scapy, sniff() for listening or sniffing packets, and packets.show() for packet analysis [11,12, 13].

In other words, the special capability of scapy allows the construction of tools that can probe, scan or attack networks. As depicted in figure 2, it can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. Besides, it can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark) [11, 12, 13].

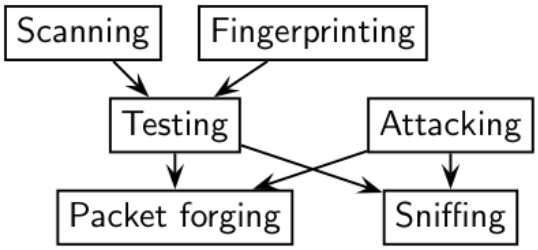

Figure 2: Functions of Scapy Module [11]

Some of the most powerful and useful functions of Scapy are depicted in Table 4.

Table 4: Most common Scapy functions

Functions	***Purpose***
ls()	Lists all protocols supported by scapy
lsc()	Returns all the functions that scapy provide
show()	Gives more details about the results
summary()	Same as show() but w/o packet identification
nsummary()	Show() defaults to nsummary, with ID
RandShort()	For random value field, example src_port
RandNum()	Random within a range
fuzz()	Scapy identifies the required fields
send()	Sends packet at layer 3
sendp()	Sends packet at layer 2
sniff()	Sniffs packet
sr()	Sends and receives packet at layer 3
sr1()	Same as sr(), but returns only the 1st packet
srp()	Sends and receives packet at layer 2
srp1()	The same as srp(), but returns the first one

```
ubuntu@ubuntu-VirtualBox:~$ sudo pip install scapy
The directory '/home/ubuntu/.cache/pip/http' or its parent directory is
The directory '/home/ubuntu/.cache/pip' or its parent directory is not
Collecting scapy
  Downloading https://files.pythonhosted.org/packages/68/01/b9943984447
    100% |                                | 3.1MB 5.7MB/s
Installing collected packages: scapy
  Running setup.py install for scapy ... done
Successfully installed scapy-2.4.0
ubuntu@ubuntu-VirtualBox:~$ python2
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import scapy
>>> from scapy.all import *
>>>
```

Figure 3: Scapy installation and verification

Figure 3 illustrates how the Scapy python module can be installed on Linux or Debian operating systems and how its successful installation can be verified.

III. Objectives

The main objective of this project is to implement a part of the network scanning technique and intranet management system. That is to say, the types and versions of operating systems installed in hosts and the services they run will be detected. Specifically, the major tasks are to

- Define peculiar attributes of different operating systems.
- Build and implement a Scapy based network scanning application
- Analyze packet tuples to extract such features as the window size and TTL from the packet received from the target host.
- Detect the OS and service types running in the target host.

IV. Implementation Results & Discussion

In this section, the implementation model I used is discussed. That is to say, the connection topology of the Raspberry PI on which the python script is run and hosts whose running OS and services will be detected, the python script, some requirements, and the results are discussed in this section

A. *Implementation Model*

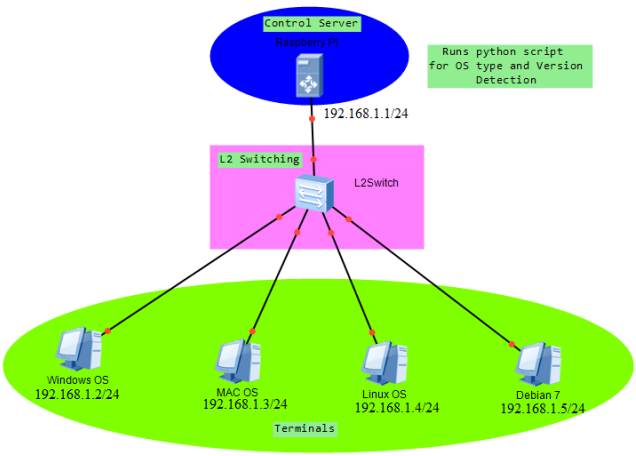

Figure 4: Simple OS Detection Model

The python script was developed with the intent of equipping the Raspberry PI with capabilities to detect the operating systems and services (along with their versions) running in hosts connected to any network to which the Raspberry PI or server running the OS detection python script is connected. For security, safe practice, and ethical reasons, I have decided to create my own LAN where the Raspberry PI will be the heart of it as portrayed in figure 4. Then, as many hosts running different operating systems as possible can be connected to it using a layer 2 switch as depicted in figure 4. Putting it another way, the python script developed exploiting the important security features of NMAP and SCAPY can be deployed in any network, but for security and simple demonstration reasons, I will run it in an isolated LAN similar to the one depicted in figure 4.

Summing it up, the architecture portrayed in figure 4 is what I have proposed for the simple but meaningful demonstration of my project. I already have the Tinker Board of Raspberry PI with a Micro SD card adapter and a micro SD card memory of size 64GB.

B. *Python Script Development*

I first fully developed my python script based on Scapy features despite the limited capacity of the Raspberry PI. The python script was developed exploiting the powers of NMAP and Scapy modules of Python Programming languages to easily detect OS's of hosts in a network. I tested my script on an actual LAN, on my home LAN, by running it on my laptop, and it successfully detected the operating systems of all devices connected to the WLAN in my apartment area. Scapy is a very powerful tool not only for packet analysis but also for packet creation and fabrication. It is a very powerful and useful module, as a result, I fully gravitated towards the use of many of its features and functions in my python script. To make it compatible and light-weight as to run in the Raspberry PI, I have tried to keep it simple and less resource hungry. Putting it another way, I focused on a light-weight program which can be run on the Raspberry PI.

The python script developed comprises such major parts as exception handling, setting the exit interface of the host running the script, defining target attributes like IP Address and Port Numbers, defining a function that does a TCP scan, and defining a function that discovers hosts by the use of Internet Control Message Protocol (ICMP) packets. The whole python script developed is affixed on appendix A, you can refer to it for further details.

C. *Requirements to run the python script*

```
ubuntu@ubuntu-VirtualBox:~$ sudo pip install scapy
The directory '/home/ubuntu/.cache/pip/http' or its paren
The directory '/home/ubuntu/.cache/pip' or its parent dir
Collecting scapy
  Downloading https://files.pythonhosted.org/packages/68/
    100% |                                | 3.1MB 5.7MB/s
Installing collected packages: scapy
  Running setup.py install for scapy ... done
Successfully installed scapy-2.4.0
```

Figure 5: Installation of the Scapy Module

For the python script developed to detect OS and services running in network hosts to properly work, there are some requirements that must be met. They include installation of Python 2 or python 3 in the host where the script will be running to perform the OS and services detection. Following the installation of either version of

Pythons, the Scapy module needs to be installed in super-admin privilege as depicted in figure 5.

Once the installation is done, you can verify using the command depicted in figure 6 whether it was successfully installed or not. That is, you can enter into the python terminal by typing "Python2 or Python 3" depending on what version of python you have installed on your OS terminal. Then, once you are in the python terminal, you can type "import scapy" and check if it can run without any error as depicted in figure 6.

```
ubuntu@ubuntu-VirtualBox:~$ python2
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for
>>> import scapy
>>> from scapy.all import *
>>>
```

Figure 6: Verification of successful installation

Afterwards, the python script should be run in the root privilege as depicted in figure 7. It cannot successfully run in users with privileges other than the root or super-admin privilege. Whenever we want to run it, it must be preceded by the "sudo" as portrayed in figure 7.

```
alem@alem-Satellite-S40-A:~$ sudo scapy
[sudo] password for alem:
```

Figure 7: Running Scapy

Another requirement is that the python script must be run on the root privilege preceded with the "sudo" command as portrayed in figure 8.

```
ubuntu@ubuntu-VirtualBox:~$ chmod 777 OSDetect.pyy
ubuntu@ubuntu-VirtualBox:~$ sudo python2 OSDetect.py
```

Figure 8: running the python script

What is more, the exit interface of the Raspberry PI or any host on which the scanning script will be running must be checked, identified and changed accordingly in the script to for correct traffic channeling.

D. Results and Discussion

```
alem@alem-Satellite-S40-A:~$ sudo python2 OSDetect.py
[sudo] password for alem:

---> Host with IP address 149.125.24.35 is down or unreachable.



---> Host with IP address 149.125.24.36 and MAC address d8:c4:6a:9d:06:ff is up.

TCP Ports:

50743 is closed!
111 is closed!
135 is closed!
22 is closed!
59178 is closed!
Cannot detect host OS --> no open ports found.


---> Host with IP address 149.125.24.37 and MAC address 00:fc:8b:62:dc:0d is up.

TCP Ports:

50743 is closed!
111 is closed!
135 is closed!
22 is closed!
59178 is closed!
Cannot detect host OS --> no open ports found.


---> Host with IP address 149.125.24.38 is down or unreachable.
```

Figure 9: Detection results

After the completion of the development phase of the python script for the detection of OS and services running in network hosts, I have conducted tests on both virtual and real networks and it correctly worked as it is designed and developed for. Figure 9 depicts the result of a scanning on a physical LAN where the listed ports are not open in the network hosts.

Figure 10 shows the result of scanning by running the python script in a virtual environment. As clearly depicted in the figure, the script is able to correctly detect the type of operating systems running in the virtual machines connected to the virtual machine on which the script was running the python script.

```
---> Host with IP address 172.16.1.2
     and MAC address 08:00:27:42:df:7a is up.
TCP Ports:

50743 is OPEN!
111 is OPEN!
135 is closed!
22 is filtered!

OS type is Linux Kernel 3.x.


---> Host with IP address 172.16.1.3
     and MAC address 08:00:27:5a:a2:44 is up.
TCP Ports:

50743 is closed!
111 is closed!
135 is OPEN!
22 is closed!

OS type is Windows 7.
```

Figure 10: Successful OS Detection

Hence, the script can be run on the Raspberry PI to successfully detect the operating systems of hosts connected to it as depicted in figure 4.

V. FUTURE WORKS

I strongly feel like I have accomplished the objective of my project. However, this work can be extended to a research work. A powerful Intranet management system can be developed using the Scapy module for the detection of OS types, service types and versions running in the network hosts, and the vulnerabilities possibly residing in them. This capability will definitely enable Intranets to timely patch weakness, and quarantine hosts with severe vulnerabilities to avoid any possible exploitation by attackers.

VI. CONCLUSION

In conclusion, the various techniques for the detection of operating systems running in network hosts and available technologies were surveyed and studied. Particularly, data

about the distinguishing features of operating systems were analyzed and identified. What is more, the powers and features of NMAP and the Scapy python module were thoroughly studied followed by the development and testing of a python script exploiting their important attributes. The python script for the detection of Services and OS running in network hosts was developed based on Scapy features. Hence, this script can be used for stealthy scanning of hosts unlike NMAP where the traffic is flagged and can be detected by monitoring and analyzing the traffic. The written script doesn't flag the traffic send to discover hosts and check what type of service and OS they are running. It is a normal traffic, and it is hard to detect it as malicious one or as a scanning traffic.

ACKNOWLEDGMENT

First and foremost, I would like to articulate my heartfelt gratitude to Professor Yu Chen for continuously endeavoring to motivate our class to engage in a number of paper reviews, and project works in an effort to help us improve our research making, findings presentation, and scientific writing skills. The Hardware Based Security course had been a very informative class in that I have learned a lot of things pertinent to hardware security and OS detection using Scapy. What is more, capitalizing on the opportunity, I would like to say thank you to my classmates for their attentive attendance of my paper-review and project presentations and constructive questions throughout the whole semester.

REFERENCES

[1] Alan Freedman, "Computer Desktop Encyclopedi", unpublished Electronic version.

[2] https://highon.coffee/blog/nmap-cheat-sheet/, accessed on 18 April 2018

[3] Abbate, Janet Ellen. "From ARPANET to Internet: A history of ARPA-sponsored computer networks, 1966--1988." (1994): 8-13

[4] Bellovin, Steven M. "A look back at" security problems in the TCP/IP protocol suite." *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004: 1-2

[5] Richardson, Robert. "2011 CSI computer crime and security survey," 2011." (2010): 19-21

[6] Singh, Utkarsh, Sumit Jaiswal, and R. S. Singh. "Cloud banking." *CSI Transactions on ICT* (2016): 1-6.

[7] CeRT, MAJOR AUSTRALIAN BUSINESS. "2015 CYBER SECURITY SURVEY." (2015): 19-23

[8] White, G. K. *Simple Institutional and User Best Practices for Cybersecurity in Research Reactors*. No. LLNL-CONF-679241. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2015: 3-7

[9] Ivanovs, Ivo, and Sintija Deruma. "Revising Cybersecurity Skills for Enterprises."

[10] Larry L.Peterson and Bruce S. Davie, "Computer Networks, A System Approach", 3rd Edition, Morgan Kaufman Publishers

[11] http://www.secdev.org/projects/scapy

[12] https://github.com/secdev/scapy

[13] http://scapy.readthedocs.io/en/latest/introduction.html

APPENDIX A: PYTHON SCRIPT

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""=============================================================================="""
"""IDE used: Spyder
  Python 2.7.14 on Anaconda2
  In Scapy v2 use from scapy.all import * instead of from scapy import *.
  """
#==============================================================================
"""
Created on Friday April 6 20:18:57 2018
Due Date: Tuesday May 15 23:59 2018
@author: Alem Haddush Fitwi
Email:afitwi1@binghamton.edu
Hardware-Based Security - EECE658
Department of Electrical & Computer Engineering
Watson Graduate School of Engineering & Applied Science
The State University of New York @ Binghamton
"""
#==============================================================================
"""
Project : OS Type Detection using TTL and Window Size
"""
#==============================================================================
#------------------------------------------------------------------------------
```

```
"""
Step_1: Importing Required Packages or modules:
   Importing the logging module to suppress all messages that have lower level of severity as compared to
   error messages. This is mainly to suppress the messages that appear every time we open Scapy
"""
#-----------------------------------------------------------------------
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
logging.getLogger("scapy.interactive").setLevel(logging.ERROR)
logging.getLogger("scapy.loading").setLevel(logging.ERROR)
try:
   from scapy.all import *
except ImportError:
   print ("Scapy package for Python is not installed on your system.")
   print ("Get it from https://pypi.python.org/pypi/scapy and try again.")
   sys.exit()
#=======================================================================
#-----------------------------------------------------------------------
"""
Step_2: Define the targets IP & Port addresses, and the exit interface of
      the host on which this python script will be running.
"""
#-----------------------------------------------------------------------
# You can define as many IP Addresses of Target hosts as you want
targets = ['149.125.24.35', '149.125.24.36', '149.125.24.37']
#You can define the target port numbers or services below
ports = [80,50743, 111, 135, 22, 59178] #specified ports
#Define the exit interface of the host that runs this script
interface = "wlan0" #please check your working interface first
#=======================================================================
#-----------------------------------------------------------------------
"""
Step_3: Define a method called "scanICMP":
"""
#-----------------------------------------------------------------------
def scanICMP():
   """It checks network hosts by sending an ICMP echo, analyze the
     reply of the target hosts, and extract the TTL and Window Size
     in order to determine the type of OS they are running.
   """
   for target in targets:
      ping_reply = srp1(Ether() / IP(dst = target) / ICMP(),
                  timeout = 5, iface = interface, verbose = 0)
      if (str(type(ping_reply)) == "<type 'NoneType'>" or
            ping_reply.getlayer(ICMP).type == "3"):
         print ("\n---> Host with IP %s is down or unreachable."  % target)
      else:
         print ("\n\n---> Host with IP %s and MAC address %s is up."
                % (target, ping_reply[Ether].src))
```

```
        print ("\nTCP Ports:\n")
        open_ports = scanTCP(target, ports)
        if len(open_ports) > 0:
           pkt = sr1(IP(dst = target) / TCP(dport = open_ports[0],  flags = "S"), timeout = 2, iface = interface,
                     verbose = 0)
           ttl = str(pkt[IP].ttl)
           window = str(pkt[TCP].window)
           if ttl == "64" and window == "5720":
              print("Detected OS type is Chrome OS/Android.\n")
           elif ttl=="255" and window == "4128":
              print("Detected OS type is Cisco IOS 12.4.\n")
           elif ttl=="64" and window == "65535":
              print("Detected OS type is FreeBSD.\n")
           elif ttl == "64" and window == "5840":
              print("Detected OS type is Linux Kernel 2.x.\n")
           elif ttl == "64" and window == "14600":
              print("Detected OS type is Linux Kernel 3.x.\n")
           elif ttl == "64" and window == "65535":
              print("Detected OS type is MAC OS.\n")
           elif ttl == "128" and window == "16384":
              print("Detected OS type is Windows 2000.\n")
           elif ttl == "128" and window == "8192":
              print("Detected OS type is Windows 7.\n")
           elif ttl == "128" and window == "65535":
              print("Detected OS type is Windows XP.\n")
        else:
           print("Cannot detect host OS --> no open ports found.")
#==============================================================================
#------------------------------------------------------------------------------
"""
Step_4: Define a method called "scanTCP":
"""
#------------------------------------------------------------------------------
def scanTCP(target, port):
   """It scans for opened ports and services by sending a SYN pkt
     Input@rgument: target, and port
     Output:open_ports
   """
   open_ports = [] #list of ports
   ans, unans = sr(IP(dst = target) / TCP(sport = RandShort(),
                dport = port, flags = "S"), timeout = 5,
                iface = interface, verbose = 0)
   for sent, received in ans: #Handles answered packets
      if received.haslayer(TCP) and str(received[TCP].flags) == "18":
         print str(sent[TCP].dport) + " is OPEN!" #syn pkt
         open_ports.append(int(sent[TCP].dport))
      elif received.haslayer(TCP) and str(received[TCP].flags) == "20":
         print str(sent[TCP].dport) + " is closed!" #reset
      elif received.haslayer(ICMP) and str(received[ICMP].type) == "3":
```

```
            print str(sent[TCP].dport) + " is filtered!" #unreachable
    for sent in unans:#handles unanswered packets
        print str(sent[TCP].dport) + " is filtered!"
    return open_ports
#================================================================================
#--------------------------------------------------------------------------
"""
Step_5: Call the  "scanICMP" method to perform scanning:
"""
#--------------------------------------------------------------------------
scanICMP()
#================================================================================
#--------------------------------------------------------------------------
""" ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~End of Program ~~~~~~~~~~~~~~~~~~~~~~~~~~~~"""
#================================================================================
```