Writing an Quick Packet Sniffer with **Python** & **Scapy**

Author: <u>Roshan Poudel</u>

Abstract

Everyday Information Security Professionals and Network Engineers come across several problems where available tools became quite ineffective to resolve the problem. Available tools can also provide desired results but sometimes they provide huge chunks, deluge of data which itself consumes time and effort to filter out. Hence, Having a knowledge of some scripting always helps in problem solving, research and automation which end up saving time, cost and effort. This paper elaborates the development of a quick packet sniffer using Python Programming language and Scapy Framework. Packet sniffer is a software tool to intercept, log, and analyze network traffic and data. Writing a packet sniffer clarifies us with understanding packet layers, components of network packets, crafting, sniffing, dissection and also assists getting the hands dirty for further exploration.

1.0 Introduction

The internet Penetration rate of the world is accelerating at a significant rate (internet world stat , 2020) . At such a pace, analyzing and evaluating the network traffic is extremely challenging as well as important. Parallel with the growth of internet penetration rate; cyber incidents are increasing and emerged as a big challenge for organization of diverse backgrounds. Hence, information security engineers and professionals have a sensitive responsibility to prevent these incidents.

Scripting is the highest sought skill in the professionals of information security or even for the network side. Scripting knowledge of some programming languages helps to automate security tasks, develop tools and possibility goes on. Packet Sniffers are a nifty and extremely handy tool for network and security professionals. Sniffers allow security professionals to analyze, filter and monitor network packets. As the topic of this paper, we are going straight forward to develop a packet sniffer to sniff network packets and classify packets based on protocols TCP, UDP,ICMP. If we refer ourselves to the internet statistics of the internet world stat we can analyze that the growth rate of internet penetration rate is at a dizzying pace. Below is a picture of the internet penetration rate of the world on first quarter of 2020.
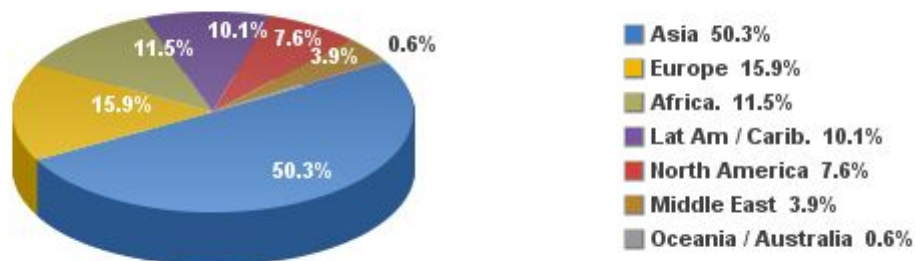
Fig: Growth of Internet Penetration rate (src:Internet world stat, 2020)

1.2 Aims and Objectives of Paper

The main aim of this paper is to develop a packet sniffer that can help some infosec professionals. Actually, I learned scapy on my college days and wrote several useful tools using python and scapy. These tools are extremely useful for me today as a security professional. Hence, today I want to share some of the tools I developed. Some of the key subjects that will be addressed in this paper are:

- Brief Explanation of Packet Sniffers and Use Cases of them
- Development of packet Sniffer with sequential steps on a layman term
- Availability of all codes on the github repository

1.3 Expected Knowledge on Readers

I am not going deep into describing all the protocols and detailed elaboration of packet sniffer. I have elaborated all the details of network protocols and packet sniffer on my previous paper which is available here. Hence, some sort of scripting python and network knowledge is expected on the readers of this paper. Also, scapy readers are expected to have basic knowledge of scapy framework. The detailed documentation of scapy is available here. Having knowledge of scapy always assists to develop security tools in python.

1.4 Scope of the Paper

This paper is written to address the development of a quick packet sniffer using python and scapy. In this paper we are going to classify all packets using layer composition. The packet sniffer will sniff all the incoming packets and outgoing packets from the host machine from all interfaces. Packets will be classified based on TCP, UDP and ICMP protocols. On each protocol classification they are divided into incoming packets and outgoing packets. Some of the information eject on the console are source ip, destination ip, source port, destination port, geo location etc. The packet sniffer does not have GUI interface and is executed from command console.

2.0 Packet Sniffer and its Development

Packet sniffing is used within a network to capture and register data flows. The process allows you to discern each individual packet and analyze its content based on predefined parameters. Packet Sniffing allows for very detailed network monitoring and bandwidth usage analysis. It, however, requires a broader knowledge of networks and their inner functions, to be able to recognize the relevance of the data being monitored(source: Paessler AG, 2013).

A packet sniffer can be usually be setup in two very different paradigms:
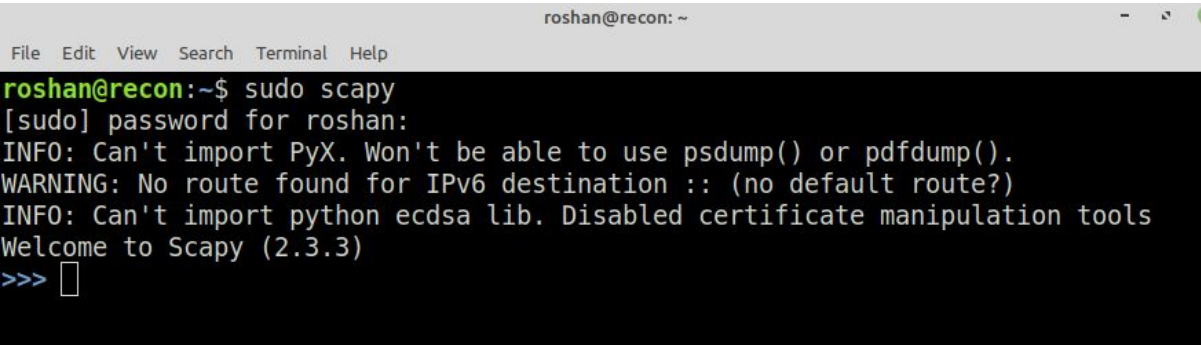
> ✓ Unfiltered Packets Paradigm – This setup captures all the packets generated in the network

> ✓ Filtered Packets Paradigm – This setup captures only those packets that seem to contain specific versions of data elements. (source: HowStuffWorks, Inc, 2013).

2.1 Scripting Packet Sniffer with Python and Scapy

Pre-requisites: LInux Operating System (I am writing this on debian)

Installing Scapy: Installing scapy in your operating system. For more information refer to official documentation here. For debian it can be installed as
sudo apt install scapy

Step 1: Once the scapy is installed it can be confirmed using **sudo scapy.** A successful installation will provide the following output.

Fig: Successful installation of scapy

Step 2: Create a python file and import all the required modules.
i.e *os, socket, scapy, datetime.*
Note: Geoip is not pre-installed in python so it can be installed using **pip install python-geoip** and official documentation here

```python
#!/usr/bin/python

from scapy.all import *
import socket
import datetime
import os
from geoip import geolite2
import time
```

Fig: importing all required modules

Step 3: After importing all the required modules creating a function and using python builtin main function. Also, using the prn parameter helps to sniff packets continuously. In place of prn if we use count=1 then only one packet will be sniffed,

```python
def network_monitoring_for_visualization_version(pkt):


if __name__ == '__main__':
    sniff(prn=network_monitoring_for_visualization_version)
```

Fig: Creating function and start sniffing

Step 4: Classifying packets into TCP, UDP and ICMP. Scapy has a builtin function to check if a packet has layers of protocols. i.e packet.haslayer(TCP), or packet.haslayer(UDP) or any protocols supported by scapy.

```python
def network_monitoring_for_visualization_version(pkt):
    # Importing time to get exact time when packet sniffed
    time=datetime.datetime.now()
    #classifying packets into TCP
    if pkt.haslayer(TCP):
    #classifying packets into UDP
    if pkt.haslayer(UDP):
    #classifying packets into ICMP
    if pkt.haslayer(ICMP):

if __name__ == '__main__':
    sniff(prn=network_monitoring_for_visualization_version)
```

Fig 4: Classifying packets into TCP, UDP and ICMP

Step 5: Classifying packets into incoming and outgoing packets. At the time i am writing this paper there are ipv4 network addresses used in my ISP. Now, we can classify the packets into incoming and outgoing packets based on the location.

Classification Criteria:

If source address in packet is IP address of my machine=outgoing packet

If destination address in packet is IP address of my machine=incoming packet

```python
10   def network_monitoring_for_visualization_version(pkt):
11       time=datetime.datetime.now()
12
13       # Classifying packets into TCP
14       if pkt.haslayer(TCP):
15           # classyfying packets into TCP Incoming packets
16           if socket.gethostbyname(socket.gethostname())== pkt[IP].dst:
17           # classyfying packets into TCP Outgoing packets
18           if socket.gethostbyname(socket.gethostname())==pkt[IP].src:
19
20       # Classifying packets into UDP
21       if pkt.haslayer(UDP):
22           if socket.gethostbyname(socket.gethostname())==pkt[IP].src:
23           # classyfying packets into UDP Outgoing packets
24           if socket.gethostbyname(socket.gethostname())==pkt[IP].dst:
25           # classyfying packets into UDP Incoming packets
26
27       # Classifying packets into ICMP
28       if pkt.haslayer(ICMP):
29           # classyfying packets into ICMP Outgoing packets
30           if socket.gethostbyname(socket.gethostname())==pkt[IP].src:
31           # classyfying packets into ICMP incmoming packets
32           if socket.gethostbyname(socket.gethostname())==pkt[IP].dst:
33   if __name__ == '__main__':
34       sniff(prn=network_monitoring_for_visualization_version)
```

Fig 5: Classifying packets into incoming and outgoing packets

Step 6: We are at the end of packet sniffer development. Now the final step is to print all required information in the console. Information printed in the console are: *Time stamp, Source Port, Destination Port, Source Ip Address, Destination IP Address, Geo location*

Sample of printed data from TCP-Incoming packet:

print(str("[")+str(time)+str("]")+" "+"TCP-IN:{}".format(len(pkt[TCP]))+" Bytes"+" "+"SRC-MAC:" +str(pkt.src)+"   "+ "DST-MAC:"+str(pkt.dst)+"   "+ "SRC-PORT:"+str(pkt.sport)+"   "+"DST-PORT:"+str(pkt.dport)+" "+"SRC-IP:"+str(pkt[IP].src)+"   "+"DST-IP:"+str(pkt[IP].dst)+" " +"Location:" +geolite2.lookup(pkt[IP].src).timezone)

```python
def network_monitoring_for_visualization_version(pkt):
    time=datetime.datetime.now()
    #classifying packets into TCP
    if pkt.haslayer(TCP):
        # classyfying packets into TCP Incoming packets
        if socket.gethostbyname(socket.gethostname())== pkt[IP].dst:
            print(str("[")+str(time)+str("]")+"  "+"TCP-IN:{}".format(len(pkt[TCP]))+" Bytes"+"     "+"SRC-MAC:" +str(pkt.src)+"

        if socket.gethostbyname(socket.gethostname())==pkt[IP].src:
            print(str("[")+str(time)+str("]")+"  "+"TCP-OUT:{}".format(len(pkt[TCP]))+" Bytes"+"     "+"SRC-MAC:" +str(pkt.src)+"
    #classifying packets into UDP
    if pkt.haslayer(UDP):
        if socket.gethostbyname(socket.gethostname())==pkt[IP].src:
            # classyfying packets into UDP Outgoing packets
            print(str("[")+str(time)+str("]")+"  "+"UDP-OUT:{}".format(len(pkt[UDP]))+" Bytes "+"     "+"SRC-MAC:" +str(pkt.src)+"

        if socket.gethostbyname(socket.gethostname())==pkt[IP].dst:
            # classifying packets into UDP Incoming packets
            print(str("[")+str(time)+str("]")+"  "+"UDP-IN:{}".format(len(pkt[UDP]))+" Bytes "+"     "+"SRC-MAC:" +str(pkt.src)+"
    #classifying packets into ICMP
    if pkt.haslayer(ICMP):
        # classyfying packets into ICMP Outgoing packets
        if socket.gethostbyname(socket.gethostname())==pkt[IP].src:
            print(str("[")+str(time)+str("]")+"  "+"ICMP-OUT:{}".format(len(pkt[ICMP]))+" Bytes"+"     "+"IP-Version:"+str(pkt[IP].
        # classyfying packets into ICMP incmoming packets
        if socket.gethostbyname(socket.gethostname())==pkt[IP].dst:
            print(str("[")+str(time)+str("]")+"  "+"ICMP-IN:{}".format(len(pkt[ICMP]))+" Bytes"+"     "+"IP-Version:"+str(pkt[IP].v
if __name__ == '__main__':
    sniff(prn=network_monitoring_for_visualization_version)
```

Fig 6: printing all required values from packet.

Step 7: Sniffing Packets. The program is now completed and now we can sniff packets using the script. To sniff packets just run sudo python <script name.py>. The code is also available in github here.

Output: The console output of the code is:-

Fig 7: TCP-UDP incoming and outgoing packets are captured by script

Step 8: Capturing ICMP Packets by pinging a machine.

Fig: Captured ICMP packets while i ping an ip address

3.0 Conclusion and Codes

Packet Sniffer developed was able to capture packets like TCP, UDP, ICMP. For each component the information like source ip, destination ip, source port, destination port, geo location , timestamp is printed out. This script is handy to capture packets from the terminal. This tool is extremely handy for infosec and network professionals to capture packets by running a script with small size. The code are available at github here.

4.0 References:

1. Internet Users Distribution in the world
Available on https://www.internetworldstats.com/stats.htm
[Accessed on 2020/05/28]
2. Miller, R. (2019). The OSI Model: An Overview. SANS Institute., Page(s):5-12
3 Nimisha P, R. G. (2014). Packet Sniffing: Network Wiretapping. IEEE International Advance Computing Conference.

4 Pallavi Asrodia, H. (2012). Network Traffic Analysis Using Packet Sniffer . International

Journal of Engineering Research and Applications .

5. Magers Daniel.(2002). Packet Sniffing: An Integral Part of Network Defense ,SANS Institute

6. Scapy official documentation

Available on https://scapy.readthedocs.io/en/latest/

[Accessed on 2020/01/20]