INTRODUCTION

In today's life networks is playing a very important role in telecommunication. without the network, almost all types of communication and service are useless. hence, this makes network concept more important for all programmers and network administrators.  
  
To maintain and manage the security of network communication, many times network administrators or network maintainers need to find and control the traffic flowing into the network wire and also find exactly what and which types of data packets are actually flowing into the networks.

For this situation, there are many types of Networks analysing tools are available on the internet.  basically, these types of tools come on the ground to help network administrator like Wireshark and other. These tools are fast, easy and reliable to handle many types of network problems but as we know, networking concept is not that easy. so, many time these types of tools do not support our exact situation requirement and we have to find any other solution for our problem and at that time python and its socket module comes on the ground like a big boy to help network administrators.  
  
Well, as we know python is really the very awesome language and also very powerful language. With Python, a programmer can do almost any types of programming in fastest and easiest way. hence, with python and socket module, our today project is very easy to codes if compared to other programming languages.

ABOUT PROJECT

Our project is divided in two separate files as follow in order to keep its simplicity ->

1. **pypackets.py**

The first file will be used to capture the packets on the network and pass them to the next module for extraction.

Of course, for this job we are going to use socket module. basically, socket module is the main player in our games because in python programming language socket module provides us the facility to play with network concept. so here for capturing packets, we are going to use socket.socket module.

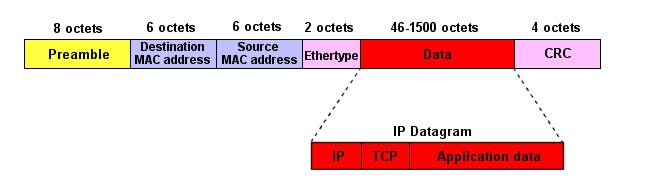
For sniffing with socket module in python we have to create a socket.socket class object with special configuration. In simple words, we have to configure socket.socket class object to capture low-level packets from the network so that it can capture packet from low-level networks and provides us output without doing any type of changes in capture packets.

1. **pye.py**

This the second file to which the first module pypackets.py passes data to be pared int the required format.

There are various types of data formats are available in networking. But for practice purpose here, we are only going to describe few important and most usable data formats. In order to understand these data formats, let's take a look at data structure diagrams.

**Ethernet Frame Format**



As you can see in ethernet frame format diagram there are more than 3 fields to extract but here, for this project we are only going to extract only 3 fields, source mac address, destination mac address and ethernal protocol type.

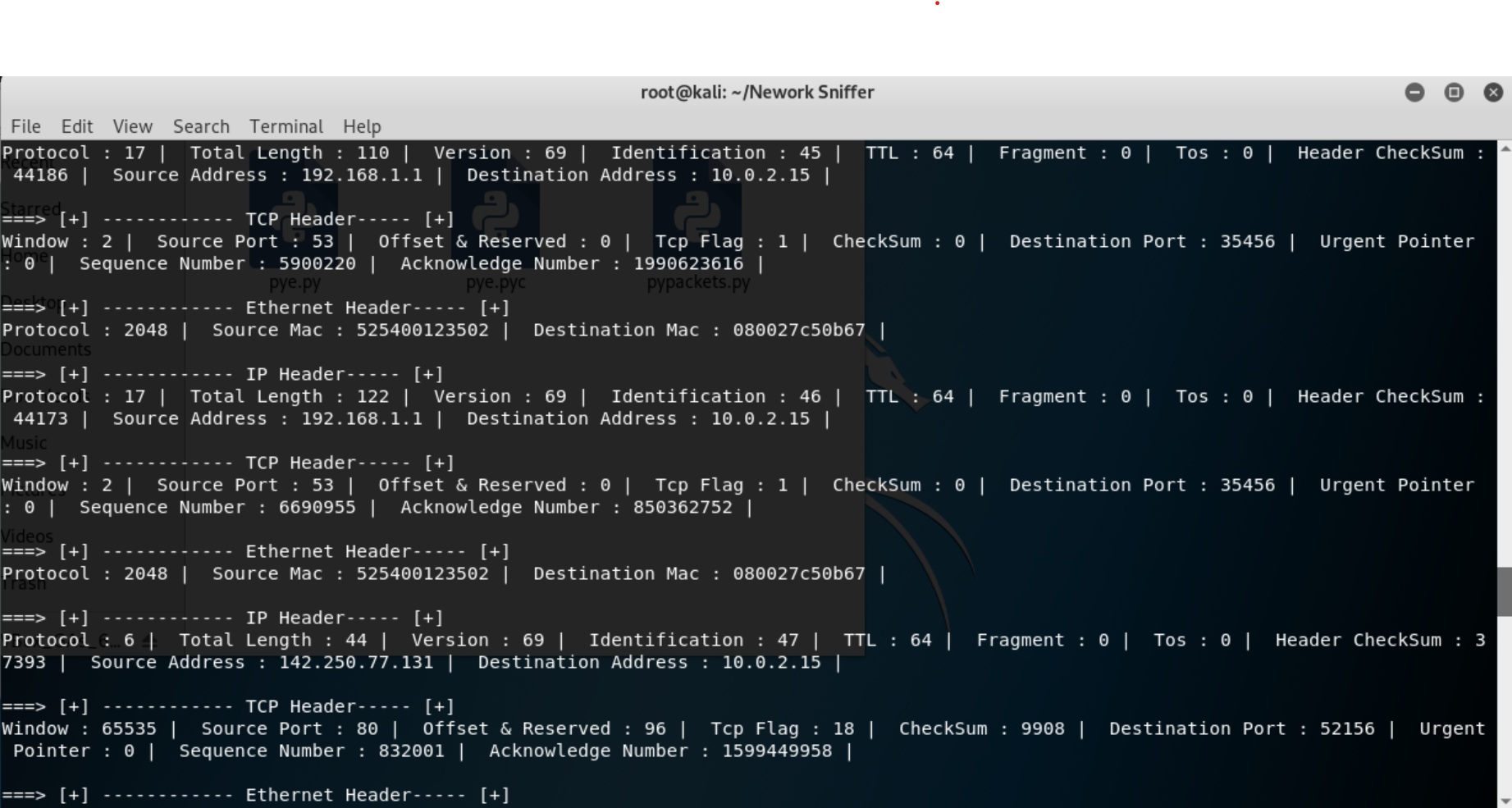
To extract source address, destination address, and ethernet type address, we are using struct module which can unpack network packets.

### We have done the same for all the header files like ICMP header format, IP header format, TCP header format and UDP header file

FEATURES OF THE PROJECT

* No External Dependencies
* Using Custom python Script For Extracting Header
* Supported Header: TCP/IP, IPv4 , UDP, ICMP
* Fast Header Extraction

WORKING OF THE PROJECT(KALI LINUX)



CODE FILES

* 1. **pypackets.py**

# import modules

import socket

import struct

import binascii

import os

import pye

# print author details on terminal

print pye.\_\_author\_\_

# if operating system is windows

if os.name == "nt":

    s = socket.socket(socket.AF\_INET,socket.SOCK\_RAW,socket.IPPROTO\_IP)

    s.bind(("YOUR\_INTERFACE\_IP",0))

    s.setsockopt(socket.IPPROTO\_IP,socket.IP\_HDRINCL,1)

    s.ioctl(socket.SIO\_RCVALL,socket.RCVALL\_ON)

# if operating system is linux

else:

    s=socket.socket(socket.PF\_PACKET, socket.SOCK\_RAW, socket.ntohs(0x0800))

# create loop

while True:

    # Capture packets from network

    pkt=s.recvfrom(65565)

    # extract packets with the help of pye.unpack class

    unpack=pye.unpack()

    print ("\n\n===> [+] ------------ Ethernet Header----- [+]")

    # print data on terminal

    for i in unpack.eth\_header(pkt[0][0:14]).iteritems():

        a,b=i

        print "{} : {} | ".format(a,b),

    print ("\n\n===> [+] ------------ IP Header----- [+]")

    for i in unpack.ip\_header(pkt[0][14:34]).iteritems():

        a,b=i

        print "{} : {} | ".format(a,b),

    print ("\n\n===> [+] ------------ TCP Header----- [+]")

    for  i in unpack.tcp\_header(pkt[0][34:54]).iteritems():

        a,b=i

        print "{} : {} | ".format(a,b),

* 1. **pye.py**

#!usr/bin/python

# Importing Modules

import socket, struct, binascii

\_\_author\_\_ = '''

# =========================================================================|

# ------------------------Network Packet Snifer----------------------------

# =========================================================================|

######################################################

  Made  By

######################################################

==> Shubham sharma (1/18/FET/BCS/188)

==> Dawood Adim    (1/18/FET/BCS/191)

==> Rachit Chugh   (1/18/FET/BCS/194)

==> Bharat Mago    (1/18/FET/BCS/198)

######################################################

 Sniffing Data Packet Extractor

'''

\_\_headers\_support\_\_ = """

Ethernet header Extraction

IPv4 header Extraction

Tcp header Extraction

ICMP header Extraction

UDP header Extraction

"""

class unpack:

    def \_\_cinit\_\_(self):

        self.data = None

    # Ethernet Header

    def eth\_header(self, data):

        storeobj = data

        storeobj = struct.unpack("!6s6sH", storeobj)

        destination\_mac = binascii.hexlify(storeobj[0])

        source\_mac = binascii.hexlify(storeobj[1])

        eth\_protocol = storeobj[2]

        data = {"Destination Mac": destination\_mac,

                "Source Mac": source\_mac,

                "Protocol": eth\_protocol}

        return data

    # ICMP HEADER Extraction

    def icmp\_header(self, data):

        icmph = struct.unpack('!BBH', data)

        icmp\_type = icmph[0]

        code = icmph[1]

        checksum = icmph[2]

        data = {'ICMP Type': icmp\_type,

                "Code": code,

                "CheckSum": checksum}

        return data

    # UDP Header Extraction

    def udp\_header(self, data):

        storeobj = struct.unpack('!HHHH', data)

        source\_port = storeobj[0]

        dest\_port = storeobj[1]

        length = storeobj[2]

        checksum = storeobj[3]

        data = {"Source Port": source\_port,

                "Destination Port": dest\_port,

                "Length": length,

                "CheckSum": checksum}

        return data

    # IP Header Extraction

    def ip\_header(self, data):

        storeobj = struct.unpack("!BBHHHBBH4s4s", data)

        \_version = storeobj[0]

        \_tos = storeobj[1]

        \_total\_length = storeobj[2]

        \_identification = storeobj[3]

        \_fragment\_Offset = storeobj[4]

        \_ttl = storeobj[5]

        \_protocol = storeobj[6]

        \_header\_checksum = storeobj[7]

        \_source\_address = socket.inet\_ntoa(storeobj[8])

        \_destination\_address = socket.inet\_ntoa(storeobj[9])

        data = {'Version': \_version,

                "Tos": \_tos,

                "Total Length": \_total\_length,

                "Identification": \_identification,

                "Fragment": \_fragment\_Offset,

                "TTL": \_ttl,

                "Protocol": \_protocol,

                "Header CheckSum": \_header\_checksum,

                "Source Address": \_source\_address,

                "Destination Address": \_destination\_address}

        return data

    # Tcp Header Extraction

    def tcp\_header(self, data):

        storeobj = struct.unpack('!HHLLBBHHH', data)

        \_source\_port = storeobj[0]

        \_destination\_port = storeobj[1]

        \_sequence\_number = storeobj[2]

        \_acknowledge\_number = storeobj[3]

        \_offset\_reserved = storeobj[4]

        \_tcp\_flag = storeobj[5]

        \_window = storeobj[6]

        \_checksum = storeobj[7]

        \_urgent\_pointer = storeobj[8]

        data = {"Source Port": \_source\_port,

                "Destination Port": \_destination\_port,

                "Sequence Number": \_sequence\_number,

                "Acknowledge Number": \_acknowledge\_number,

                "Offset & Reserved": \_offset\_reserved,

                "Tcp Flag": \_tcp\_flag,

                "Window": \_window,

                "CheckSum": \_checksum,

                "Urgent Pointer": \_urgent\_pointer

                }

        return data

    # Mac Address Formating

def mac\_formater(a):

    b = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (ord(a[0]), ord(a[1]), ord(a[2]), ord(a[3]), ord(a[4]), ord(a[5]))

    return b

def get\_host(q):

    try:

        k = socket.gethostbyaddr(q)

    except:

        k = 'Unknown'

    return k