

**CSPC-45**  
**Assignment-1**  
**Packet Sniffing**  
**By**  
**Shivam Dhammi**  
**CS 5**  
**11720013**

## **Packet sniffing**

Packet sniffing is the practice of gathering, collecting, and logging some or all packets that pass through a computer network, regardless of how the packet is addressed. In this way, every packet, or a defined subset of packets, may be gathered for further analysis. You as a network administrators can use the collected data for a wide variety of purposes like monitoring bandwidth and traffic. A packet sniffer, sometimes called a packet analyzer, is composed of two main parts. First, a network adapter that connects the sniffer to the existing network. Second, software that provides a way to log, see, or analyze the data collected by the device.

## **How does packet sniffing work?**

A network is a collection of nodes, such as personal computers, servers, and networking hardware that are connected. The network connection allows data to be transferred between these devices. The connections can be physical with cables, or wireless with radio signals. Networks can also be a combination of both types. As nodes send data across the network, each transmission is broken down into smaller pieces called packets. The defined length and shape allows the data packets to be checked for completeness and usability. Because a network's infrastructure is common to many nodes, packets destined for different nodes will pass through numerous other nodes on the way to their destination. To ensure data is not mixed up, each packet is assigned an address that represents the intended destination of that packet. A packet's address is examined by each network adapter and connected device to determine what node the packet is destined for. Under normal operating conditions, if a node sees a packet that is not addressed to it, the node ignores that packet and its data. Packet sniffing ignores this standard practice and collects all, or some of the packets, regardless of how they are addressed.

## **Packet sniffing tools:**

## 1. Paessler PRTG Network Monitor

The PRTG Network Monitor from Paessler includes an impressive array of packet capture capabilities. The software relies on four core sensors in your network to sniff IP packets. Each sensor has its own unique capabilities. The packet sniffing sensor is designed to help sysadmins monitor an array of traffic, including web, mail, file transfer, infrastructure, and remote control traffic. It only analyzes packet headers, not packet payloads, so it places less strain on your system and helps safeguard sensitive information. The sFlow sensors are designed to place even less strain on your system: they analyze every nth packet, making them suited for large networks.

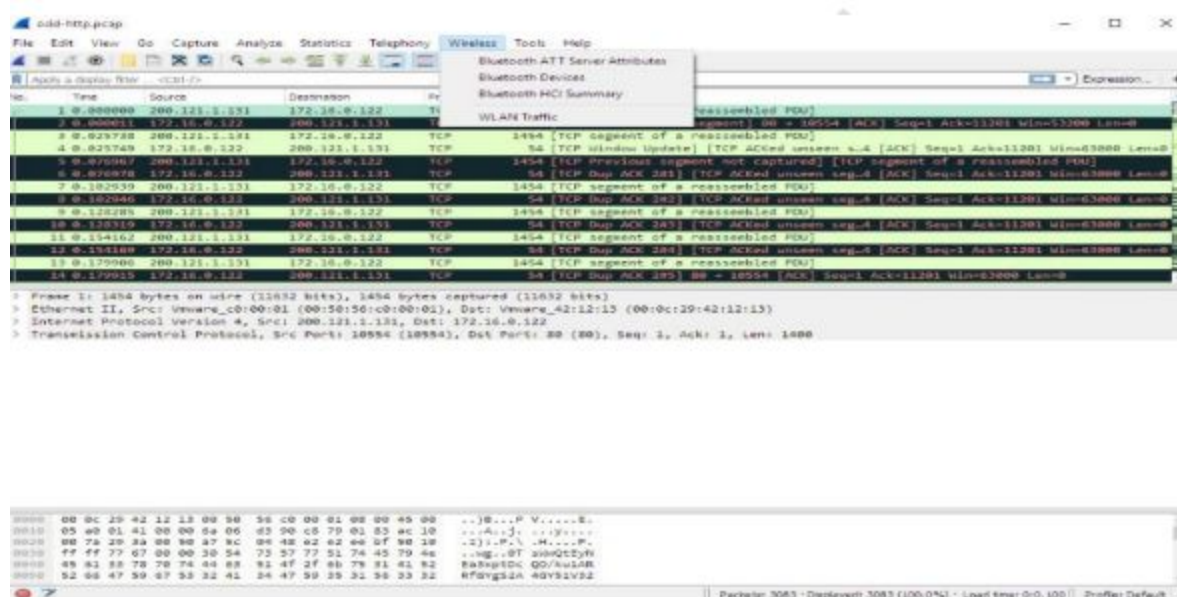
## 2. tcpdump

Many sysadmins know tcpdump as the original packet sniffer. While it has evolved slightly since its launch in 1987, it remains largely unchanged. An open-source tool, tcpdump comes installed on nearly all Unix-like operating systems and is a go-to for packet capture on the fly. Because it's a command-line tool, it doesn't require a heavy-duty desktop to run, making it a favorite among sysadmins.

## 3. WinDump

Due to the success of tcpdump on Unix-like operating systems, it was "ported over" to the Windows platform. This simply means it was cloned to allow for Windows packet capture. Like tcpdump, WinDump is a command-line tool, and its output can be saved to a file for deeper analysis by a third-party tool. WinDump is used in much the same way as tcpdump in nearly every aspect. In fact, the command-line options are the same, and the results tend to be pretty much identical.

## 4. Wireshark



Like tcpdump and WinDump, Wireshark has been around for a few decades and helped set the standard for network protocol analysis. Wireshark is a completely free, open-source tool that has been ported over to nearly all network operating systems, including Windows, Linux, macOS, Solaris, FreeBSD, and NetBSD. To this day, Wireshark remains a volunteer-run organization backed by several significant sponsorships. The Wireshark packet sniffing tool is known for both its data capture and its analysis capabilities. You can apply filters to limit the scope of data Wireshark collects, or simply let it collect all traffic passing through your selected network. Importantly, it can only collect data on a server with a desktop installed. Since desktops aren't common on servers, many sysadmins choose to use tcpdump or WinDump to capture traffic to a file, which they then load into Wireshark for in-depth analysis.

## Program:

```
# coding: utf-8

# In[6]:

from socket import *
import struct
import sys
import re

# receive a datagram
def receiveData(s):
    data = ''
    try:
        data = s.recvfrom(65565)
    except timeout:
        data = ''
    except:
        print ("An error happened: ")
        sys.exc_info()
    return data[0]

# get Type of Service: 8 bits
def getTOS(data):
    precedence = {0: "Routine", 1: "Priority", 2: "Immediate", 3: "Flash",
4: "Flash override", 5: "CRITIC/ECP",
6: "Internetwork control", 7: "Network control"}
    delay = {0: "Normal delay", 1: "Low delay"}
```

```

throughput = {0: "Normal throughput", 1: "High throughput"}
reliability = {0: "Normal reliability", 1: "High reliability"}
cost = {0: "Normal monetary cost", 1: "Minimize monetary cost"}

# get the 3rd bit and shift right
D = data & 0x10
D >>= 4

# get the 4th bit and shift right
T = data & 0x8
T >>= 3

# get the 5th bit and shift right
R = data & 0x4
R >>= 2

# get the 6th bit and shift right
M = data & 0x2
M >>= 1

# the 7th bit is empty and shouldn't be analyzed

tabs = '\n\t\t\t\t'
TOS = precedence[data >> 5] + tabs + delay[D] + tabs + throughput[T] +
tabs + reliability[R] + tabs + cost[M]
return TOS

# get Flags: 3 bits
def getFlags(data):
    flagR = {0: "0 - Reserved bit"}
    flagDF = {0: "0 - Fragment if necessary", 1: "1 - Do not fragment"}
    flagMF = {0: "0 - Last fragment", 1: "1 - More fragments"}

# get the 1st bit and shift right
R = data & 0x8000
R >>= 15

# get the 2nd bit and shift right
DF = data & 0x4000
DF >>= 14

# get the 3rd bit and shift right
MF = data & 0x2000
MF >>= 13

tabs = '\n\t\t\t\t'

```

```

    flags = flagR[R] + tabs + flagDF[DF] + tabs + flagMF[MF]
    return flags

# the public network interface
HOST = gethostbyname(gethostname())

# create a raw socket and bind it to the public interface
s = socket(AF_INET, SOCK_RAW, IPPROTO_IP)
s.bind((HOST, 0))

# Include IP headers
s.setsockopt(IPPROTO_IP, IP_HDRINCL, 1)
s.ioctl(SIO_RCVALL, RCVALL_ON)
data = receiveData(s)

# get the IP header (the first 20 bytes) and unpack them
# B - unsigned char (1)
# H - unsigned short (2)
# s - string
unpackedData = struct.unpack('!BBHHHBBH4s4s' , data[:20])

version_IHL = unpackedData[0]
version = version_IHL >> 4 # version of the IP
IHL = version_IHL & 0xF # internet header length
TOS = unpackedData[1] # type of service
totalLength = unpackedData[2]
ID = unpackedData[3] # identification
flags = unpackedData[4]
fragmentOffset = unpackedData[4] & 0x1FFF
TTL = unpackedData[5] # time to live
protocolNr = unpackedData[6]
checksum = unpackedData[7]
sourceAddress = inet_ntoa(unpackedData[8])
destinationAddress = inet_ntoa(unpackedData[9])

print ("An IP packet with the size %i was captured." % (unpackedData[2]))
print ("Raw data: " + str(data))
print ("\nParsed data")
print ("Version:\t\t" + str(version))

```

```

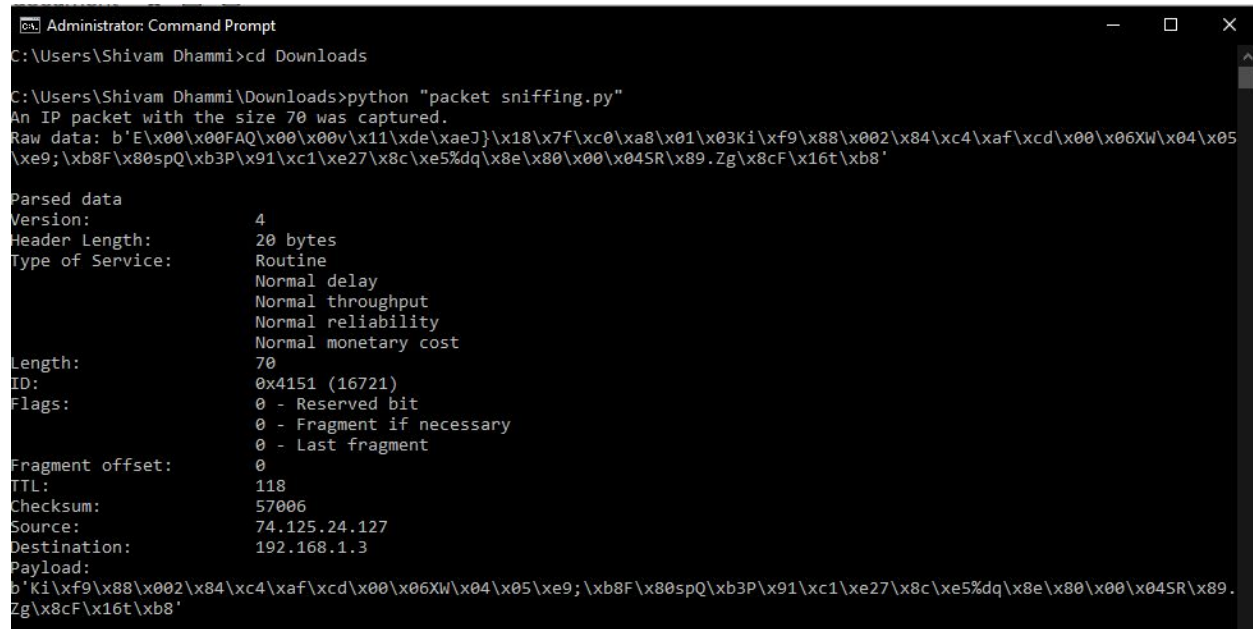
print ("Header Length:\t\t" + str(IHL*4) + " bytes")
print ("Type of Service:\t" + getTOS(TOS))
print ("Length:\t\t\t" + str(totalLength))
print ("ID:\t\t\t" + str(hex(ID)) + " (" + str(ID) + ")")
print ("Flags:\t\t\t" + getFlags(flags))
print ("Fragment offset:\t" + str(fragmentOffset))
print ("TTL:\t\t\t" + str(TTL))

print ("Checksum:\t\t" + str(checksum))
print ("Source:\t\t\t" + sourceAddress)
print ("Destination:\t\t" + destinationAddress)
print ("Payload:\n" + str(data[20:]))

# disabled promiscuous mode
s.ioctl(SIO_RCVALL, RCVALL_OFF)

```

## Output:



```

Administrator: Command Prompt
C:\Users\Shivam Dhammi>cd Downloads

C:\Users\Shivam Dhammi\Downloads>python "packet sniffing.py"
An IP packet with the size 70 was captured.
Raw data: b'E\x00\x00FAQ\x00\x00v\x11\xde\xaeJ}\x18\x7f\xc0\xa8\x01\x03Ki\xf9\x88\x002\x84\xc4\xaf\xcd\x00\x06XW\x04\x05
\xe9;\xb8F\x80spQ\xb3P\x91\xc1\xe27\x8c\xe5%dq\x8e\x80\x00\x045R\x89.Zg\x8cF\x16t\xb8'

Parsed data
Version:          4
Header Length:    20 bytes
Type of Service:  Routine
                  Normal delay
                  Normal throughput
                  Normal reliability
                  Normal monetary cost
Length:           70
ID:               0x4151 (16721)
Flags:            0 - Reserved bit
                  0 - Fragment if necessary
                  0 - Last fragment
Fragment offset:  0
TTL:              118
Checksum:         57006
Source:           74.125.24.127
Destination:      192.168.1.3
Payload:
b'Ki\xf9\x88\x002\x84\xc4\xaf\xcd\x00\x06XW\x04\x05\xe9;\xb8F\x80spQ\xb3P\x91\xc1\xe27\x8c\xe5%dq\x8e\x80\x00\x045R\x89.
Zg\x8cF\x16t\xb8'

```

