

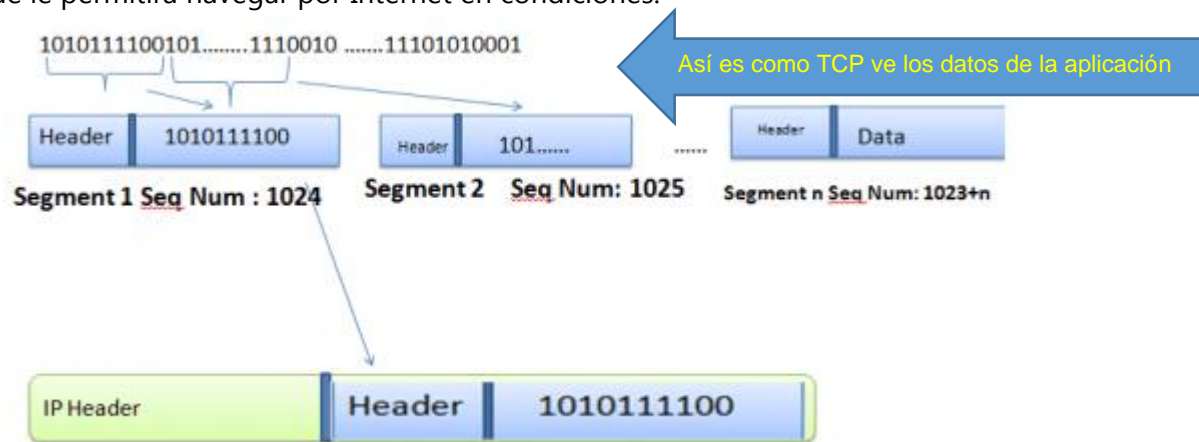
### Construyendo un sniffer de red con Python

<b>Nombre:</b> Univ. Mamani Chavez Carla Vanesa	<b>CI:</b> 9124602 LP <b>Paralelo:</b> Martes
<b>Docente:</b> Lic. Gallardo Portanda Franz Ramiro	<b>Fecha :</b> 27/06/2020

1. Escriba un programa **Python** que interprete todos los campos de la **cabecera IP** y los despliegue en la pantalla. Pruebe su programa, ya sea **generando trafico local en su máquina o conectada a la red**. Analice la información que arroja su programa en el contexto del trafico capturado.

#### 1.1. Cabecera Ip

Es la parte que el protocolo IP añade al Segmento de datos proveniente de la capa de transporte para formar el datagrama IP. Contiene distintos campos con la información que le permitirá navegar por Internet en condiciones.



Los campos de la cabecera Ip son los siguientes y trabajaremos con los siguientes:



Y primero recibimos los datos del socket, el método **recvfrom** en el módulo de socket nos ayuda a recibir todos los datos del socket. El parámetro pasado es el tamaño del búffer 65565 es el tamaño máximo del búffer.

```
def RecepcionData(s):
    data = ''
    try:
        data = s.recvfrom(65565)
    except timeout:
        data = ''
    except:
        print ("Ocurrio un error: ")
        sys.exc_info()
    return data[0]
```

Despues obtenemos el tipo de Servicio (Tos) del tamaño del campo: 8 bits. El cual indica la forma en que se trata un datagrama. Eso se traduce en si el datagrama tiene preferencia de encaminamiento o no. Realmente eso no se usa en Internet y todos los datagramas tienen el mismo valor ya que si se pudiese modificar piratas informáticos podrían aprovecharse de esa vulnerabilidad haciendo que sus paquetes se trataran con mayor urgencia.

```
19 def obtenerTOS(data):
20     precedence = {0: "Routine", 1: "Priority", 2: "Immediate", 3: "Flash", 4: "Flash override", 5: "CRITIC/ECP",
21                  6: "Internetwork control", 7: "Network control"}
22     delay = {0: "Normal delay", 1: "Low delay"}
23     throughput = {0: "Normal throughput", 1: "High throughput"}
24     reliability = {0: "Normal reliability", 1: "High reliability"}
25     cost = {0: "Normal monetary cost", 1: "Minimize monetary cost"}
26     # adquirir el 3er bit y se desplaza a la derecha
27     D = data & 0x10
28     D >>= 4
29     # adquirir el 4to bit y se desplaza a la derecha
30     T = data & 0x8
31     T >>= 3
32     # adquirir el 5to bit y se desplaza a la derecha
33     R = data & 0x4
34     R >>= 2
35     # adquirir el 6to bit y se desplaza a la derecha
36     M = data & 0x2
37     M >>= 1
38     # el 7mo bit está vacío y no debe analizarse
39
40     tabs = '\n\t\t\t\t\t'
41     TOS = precedence[data >> 5] + tabs + delay[D] + tabs + throughput[T] + tabs + \
42           reliability[R] + tabs + cost[M]
43     return TOS
44
```

### Donde:

precedence permite indicar una procedencia.

d delay, permite optimizar el retardo.

t throughput, permite optimizar la velocidad eficaz.

r reliability, permite optimizar la fiabilidad.

c cost, permite optimizar el costo (económico).

Despues pasamos a la captura de los Flags

```

46 def obtenerFlags(data):
47     flagR = {0: "0 - Reserved bit"}
48     flagDF = {0: "0 - Fragment if necessary", 1: "1 - Do not fragment"}
49     flagMF = {0: "0 - Last fragment", 1: "1 - More fragments"}
50
51     # toma el primer bit y va a la derecha
52     R = data & 0x8000
53     R >>= 15
54     # toma el segundo bit y va a la derecha
55     DF = data & 0x4000
56     DF >>= 14
57     # toma el tercer bit y va a la derecha
58     MF = data & 0x2000
59     MF >>= 13
60
61     tabs = '\n\t\t\t\t'
62     flags = flagR[R] + tabs + flagDF[DF] + tabs + flagMF[MF]
63     return flags
64

```

**Donde:**

Tenemos los bit de reserva como flagR

Fragmento si es necesario o No fragmentar como flagDF

Último fragmento o Más fragmentos como flagMF

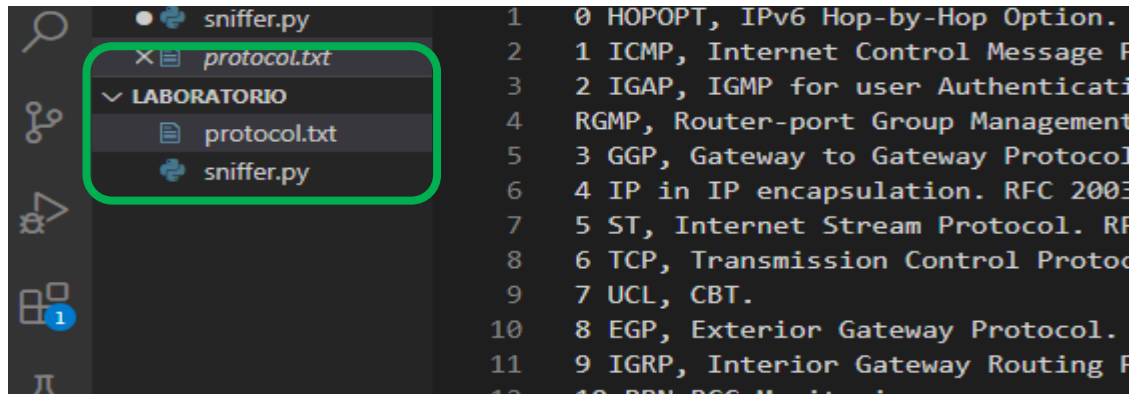
Posteriormente capturamos los protocolos donde leemos el archivo protocolos.txt con la siguiente instrucción

```

protocolFile = open('protocol.txt', 'r')
protocolData = protocolFile.read()
protocol = re.findall(r'\n' + str(protocolNr) + ' (?..)+\n', protocolData)

```

Algo muy importante que el archivo protocolos.txt tiene que estar en el mismo directorio que el archivo sniffer.py



```

1 0 HOPOPT, IPv6 Hop-by-Hop Option.
2 1 ICMP, Internet Control Message P
3 2 IGAP, IGMP for user Authentica
4 RGMP, Router-port Group Manage
5 3 GGP, Gateway to Gateway Protoc
6 4 IP in IP encapsulation. RFC 200
7 5 ST, Internet Stream Protocol. RF
8 6 TCP, Transmission Control Protoc
9 7 UCL, CBT.
10 8 EGP, Exterior Gateway Protocol.
11 9 IGRP, Interior Gateway Routing F
12 10 BBN RCC Monitoring

```

```

66 def obtenerProtocol(protocolNr):
67     #Leemos el protocol.txt se hacara desde ahi que protocolo se esta usando
68     protocolFile = open('protocol.txt', 'r')
69     protocolData = protocolFile.read()
70     protocol = re.findall(r'\n' + str(protocolNr) + ' (?..)+\n', protocolData)
71     if protocol:
72         protocol = protocol[0]
73         protocol = protocol.replace("\n", "")
74         protocol = protocol.replace(str(protocolNr), "")
75         protocol = protocol.lstrip()
76         return protocol
77
78     else:
79         return 'no existe tal protocolo'
80

```

También el tamaño del campo: 8 bits. Multiplexación de protocolos a nivel superior. Identifica mediante un número natural qué protocolo se ha encapsulado en este datagrama: TCP = 6, UDP = 17, ICMP = 1, IP = 0

**Continuamos con la función `gethostbyname` recupera la información del host correspondiente a un nombre de host de una base de datos de host.**

**Y también creamos un socket sin procesar y vincularlo a la interfaz pública.**

```
81  # la interfaz de red pública
82  HOST = gethostbyname(gethostname())
83
84  #creamos un socket sin procesar y vincularlo a la interfaz pública
85  s = socket(AF_INET, SOCK_RAW, IPPROTO_IP)
86  s.bind((HOST, 0))
87
```

**Incluimos encabezados IP y el modo promiscuo habilitado**

```
87
88  s.setsockopt(IPPROTO_IP, IP_HDRINCL, 1)
89  s.ioctl(SIO_RCVALL, RCVALL_ON)
90
```

**Recibimos los datos con `data = RecepcionData(s)` y adquirimos el encabezado IP (los primeros 20 bytes) y descomprimirlos.**

```
91
92  data = RecepcionData(s)
93  unpackedData = struct.unpack('!BBHHHBBH4s4s', data[:20])
94
95  version_IHL = unpackedData[0]
96  version = version_IHL >> 4           # Version IP
97  IHL = version_IHL & 0xF             # longitud del encabezado
98  TOS = unpackedData[1]               # type of service
99  totalLength = unpackedData[2]
100 ID = unpackedData[3]                # ID
101 flags = unpackedData[4]
102 fragmentOffset = unpackedData[4] & 0x1FFF
103 TTL = unpackedData[5]               # TTL
104 protocolNr = unpackedData[6]
105 checksum = unpackedData[7]
106 sourceAddress = inet_ntoa(unpackedData[8])
107 destinationAddress = inet_ntoa(unpackedData[9])
108
```

**Donde:**

**B - carácter sin signo (1)**

**H - corto sin signo (2)**

**s - cadena**

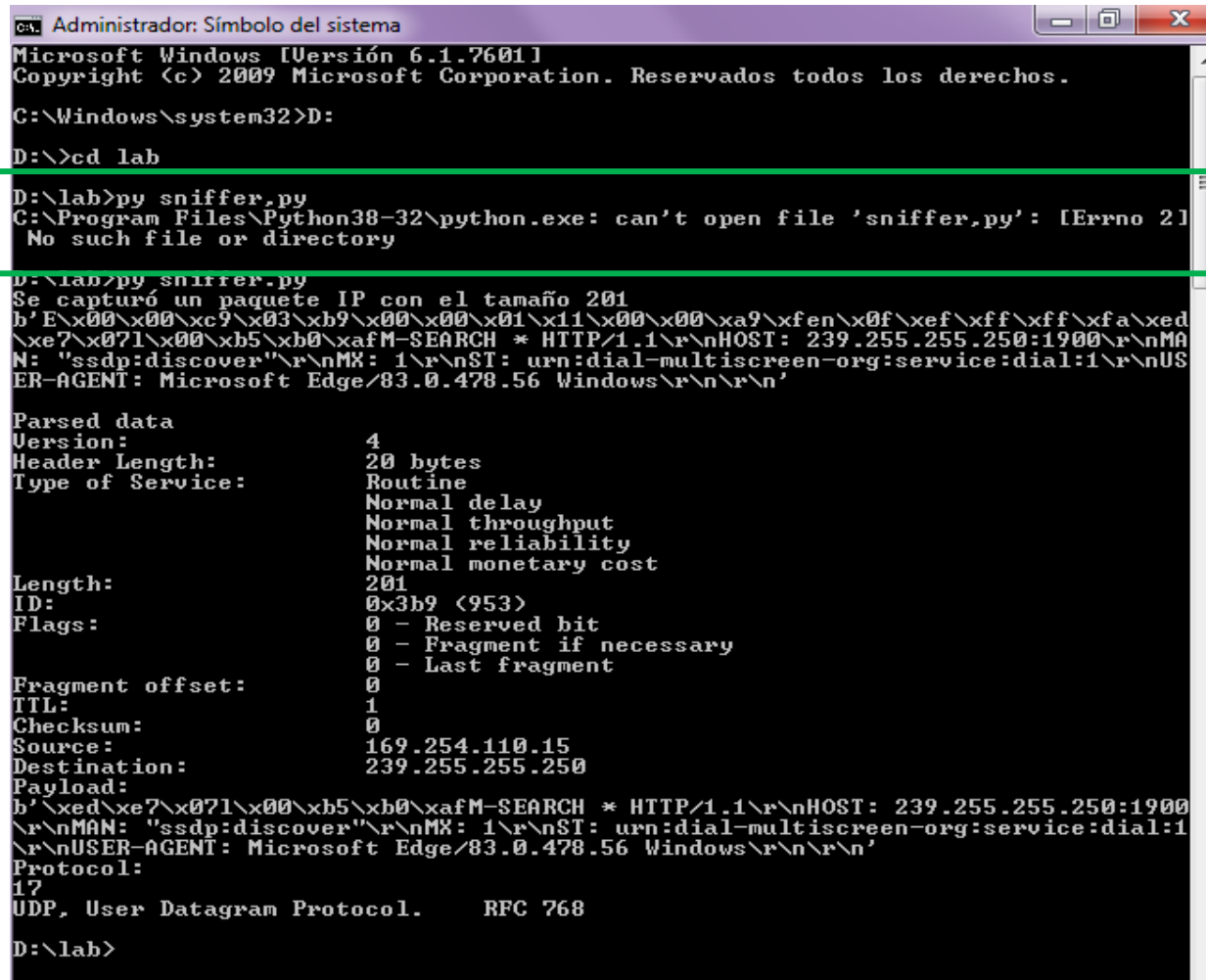
### Para finalizar mostramos todos los campos de la cabecera de la IP

```

109
110 print ("Se capturó un paquete IP con el tamaño %i" % (unpackedData[2]))
111 print (data)
112 print ("\nParsed data")
113 print ("Version:\t\t" + str(version))
114 print ("Header Length:\t\t" + str(IHL*4) + " bytes")
115 print ("Type of Service:\t" + obtenerTOS(TOS))
116 print ("Length:\t\t\t" + str(totalLength))
117 print ("ID:\t\t\t" + str(hex(ID)) + " (" + str(ID) + ")")
118 print("Flags:\t\t\t" + obtenerFlags(flags))
119 print("Fragment offset:\t" + str(fragmentOffset))
120 print("TTL:\t\t\t" + str(TTL))
121 print("Checksum:\t\t\t" + str(checksum))
122 print("Source:\t\t\t" + sourceAddress)
123 print("Destination:\t\t" + destinationAddress)
124 print("Payload:")
125 print(data[20:])
126 print("Protocol:\t\t")
127 print(protocolNr)
128 print(obtenerProtocol(protocolNr))
129 # modo promiscuo deshabilitado
130 s.ioctl(SIO_RCVALL, RCVALL_OFF)

```

Lo ejecutamos el modo administrador en Símbolos de sistema (Porque estamos trabajando en el sistema operativo Windows)



```

C:\Windows\system32>D:
D:\>cd lab
D:\lab>py sniffer.py
C:\Program Files\Python38-32\python.exe: can't open file 'sniffer.py': [Errno 2]
No such file or directory
D:\lab>py sniffer.py
Se capturó un paquete IP con el tamaño 201
b'\xe0\xe9\x03\xb9\x00\x00\x01\x11\x00\xa9\xfe\xff\xff\xff\xfa\xed
\xe7\x71\x00\xb5\b0\xafM-SEARCH * HTTP/1.1\r\nHOST: 239.255.255.250:1900\r\nMAN: "ssdp:discover"\r\nMX: 1\r\nST: urn:dial-multiscreen-org:service:dial:1\r\nUSER-AGENT: Microsoft Edge/83.0.478.56 Windows\r\n\r\n'

Parsed data
Version:          4
Header Length:    20 bytes
Type of Service:  Routine
                  Normal delay
                  Normal throughput
                  Normal reliability
                  Normal monetary cost
Length:           201
ID:               0x3b9 (953)
Flags:            0 - Reserved bit
                  0 - Fragment if necessary
                  0 - Last fragment
Fragment offset:  0
TTL:              1
Checksum:         0
Source:           169.254.110.15
Destination:      239.255.255.250
Payload:
b'\xe7\xe9\x71\x00\xb5\b0\xafM-SEARCH * HTTP/1.1\r\nHOST: 239.255.255.250:1900\r\nMAN: "ssdp:discover"\r\nMX: 1\r\nST: urn:dial-multiscreen-org:service:dial:1\r\nUSER-AGENT: Microsoft Edge/83.0.478.56 Windows\r\n\r\n'
Protocol:
17
UDP, User Datagram Protocol.    RFC 768
D:\lab>

```

En el recuerdo verde me genera error, porque cometi un error al poner el comando py sniffer.py en vez de eso puse py sniffer,py. Y genero el erro la coma en vez del punto.