GEOLOCALIZACIÓN

Realizado por: Pedro Egea, Adrián Gea, Alejandro Coves y Roberto Mateo de 1º X

Indice

Introducción

En el presente trabajo, se abordará el análisis y geolocalización de tramas de red capturadas con Wireshark utilizando el lenguaje de programación Python. El objetivo es demostrar cómo podemos extraer información valiosa de los paquetes de red y representarla visualmente en un mapa, facilitando la comprensión de la distribución geográfica de las comunicaciones de red.

El proceso comenzará con la captura de tráfico de red usando Wireshark, que nos permitirá obtener un archivo PCAP con las tramas de red de interés. Luego, mediante la librería dpkt, leeremos y procesaremos este archivo para extraer las direcciones IP de origen y destino de los paquetes. Utilizando pygeoip, obtendremos la información geográfica asociada a estas direcciones IP. Finalmente, generaremos un archivo KML (Keyhole Markup Language), compatible con Google Maps, que nos permitirá visualizar en un mapa las ubicaciones geográficas de las direcciones IP involucradas en las tramas capturadas.

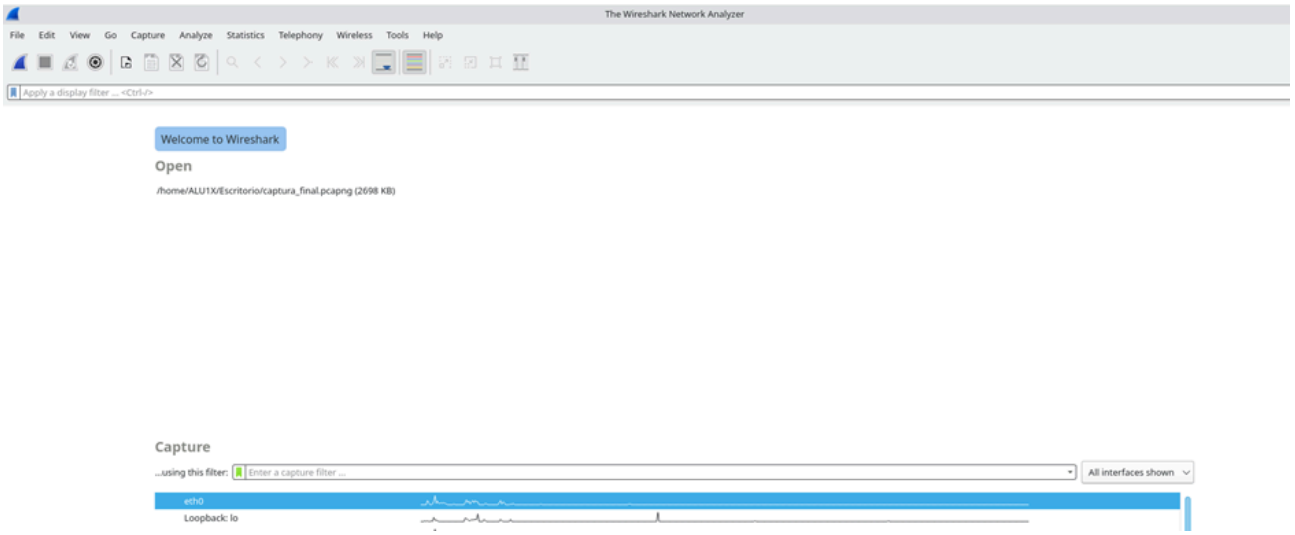
WireShark

Para capturar tráfico, una grandísima herramienta es Wireshark.

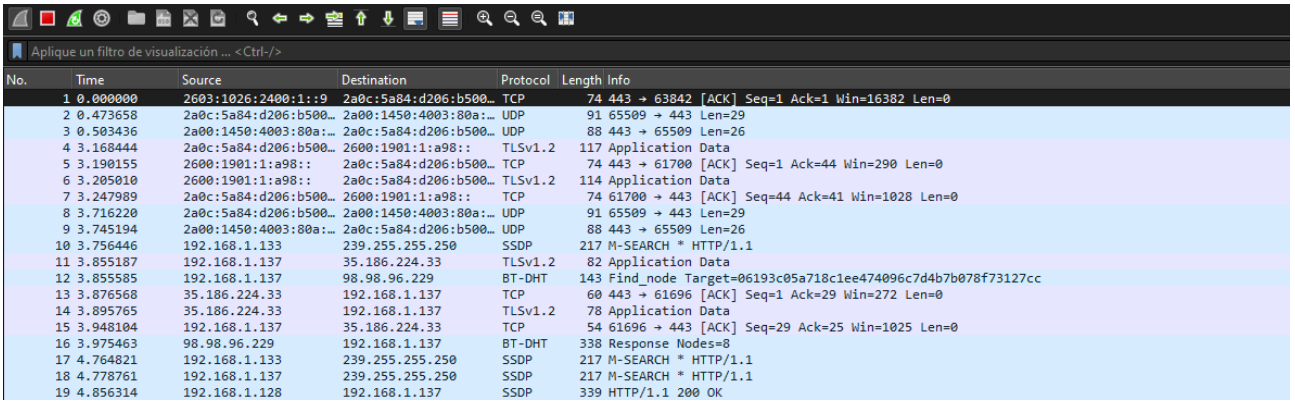
Para empezar, debemos instalar wireshark en nuestro equipo si no es que la tenemos ya preinstalada como en los ordenadores de clase.

Al entrar a Wireshark veremos todas las interfaces de red que tenemos. En caso de tener diferentes interfaces, podremos capturar también el tráfico de todas a la vez.

Para comenzar a capturar le daremos al icono azul que tenemos a nuestra izquierda en la esquina superior.



Una vez clicado en este icono, veremos que empieza a capturar tráfico. Para detener la captura le damos al botón rojo.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	2603:1026:2400:1::9	2a0c:5a84:d206:b500…	TCP	74	443 → 63842 [ACK] Seq=1 Ack=1 Win=16382 Len=0
2	0.473658	2a0c:5a84:d206:b500…	2a00:1450:4003:80a:…	UDP	91	65509 → 443 Len=29
3	0.503436	2a00:1450:4003:80a:…	2a0c:5a84:d206:b500…	UDP	88	443 → 65509 Len=26
4	3.168444	2a0c:5a84:d206:b500…	2600:1901:1:a98::	TLSv1.2	117	Application Data
5	3.190155	2600:1901:1:a98::	2a0c:5a84:d206:b500…	TCP	74	443 → 61700 [ACK] Seq=1 Ack=44 Win=290 Len=0
6	3.205010	2600:1901:1:a98::	2a0c:5a84:d206:b500…	TLSv1.2	114	Application Data
7	3.247989	2a0c:5a84:d206:b500…	2600:1901:1:a98::	TCP	74	61700 → 443 [ACK] Seq=44 Ack=41 Win=1028 Len=0
8	3.716220	2a0c:5a84:d206:b500…	2a00:1450:4003:80a:…	UDP	91	65509 → 443 Len=29
9	3.745194	2a00:1450:4003:80a:…	2a0c:5a84:d206:b500…	UDP	88	443 → 65509 Len=26
10	3.756446	192.168.1.133	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
11	3.855187	192.168.1.137	35.186.224.33	TLSv1.2	82	Application Data
12	3.855585	192.168.1.137	98.98.96.229	BT-DHT	143	Find_node Target=06193c05a718c1ee474096c7d4b7b078f73127cc
13	3.876568	35.186.224.33	192.168.1.137	TCP	60	443 → 61696 [ACK] Seq=1 Ack=29 Win=272 Len=0
14	3.895765	35.186.224.33	192.168.1.137	TLSv1.2	78	Application Data
15	3.948104	192.168.1.137	35.186.224.33	TCP	54	61696 → 443 [ACK] Seq=29 Ack=25 Win=1025 Len=0
16	3.975463	98.98.96.229	192.168.1.137	BT-DHT	338	Response Nodes=8
17	4.764821	192.168.1.133	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
18	4.778761	192.168.1.137	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
19	4.856314	192.168.1.128	192.168.1.137	SSDP	339	HTTP/1.1 200 OK

Importante filtrar por TCP, ya que HTTP funciona con este protocolo. Al filtrar solo nos saldrá el tráfico que ha capturado de las páginas web que estamos visitando mientras capturamos.

Ya filtrado por TCP, exportamos el tráfico capturado a un archivo “.pcap” que posteriormente utilizaremos para crear el mapa.

Librerías Python

Librería DPKT

La librería DPKT es una herramienta que se utiliza para la manipulación o análisis de de paquetes de red, en el código lo utilizamos para leer el archivo que sacamos del wireshark.

Librería SOCKET

La librería SOCKET tiene muchas formas de uso pero en el código la utilizamos para poder definir de cual es la ip de destino y cuál es la ip de origen para poder poder saber a la hora de dibujar el mapa saber de donde sale el paquete y a donde llega.

Librería PYGEOIP

La librería PYGEOIP es una herramienta con la cual sacamos la información geográfica de las IPs en el código cuando con la librería socket ya sacamos la ip de destino y de origen sacamos sus coordenadas y ya sabríamos de donde sale el paquete y hasta donde llega

Para importar estas librerias a nuestro código, ejecutaremos desde la terminal los comandos:

-pip install dpkt
-pip install socket
-pip install pygeoip

Explicación Código

ETH_TYPE_IP: Define el tipo de protocolo Ethernet para IP.

gi: Carga la base de datos de GeoLiteCity para la obtención de información geográfica.

```
ETH_TYPE_IP = 0x0800  # Si es IP protocol
gi = pygeoip.GeoIP('GeoLiteCity.dat')
```

Lectura del archivo PCAP:
Abre el archivo ejercicio.pcap en modo binario y lo lee con dpkt.pcap.Reader.

```
with open('ejercicio.pcap', 'rb')  as f:
    pcap = dpkt.pcap.Reader(f)
```

*KML es un lenguaje marcado basado en XML que sirve para representar datos geográficos en Google Maps.
Generación de archivo KML:
Crea las cabeceras (kmlheader) y pies (kmlfooter) del archivo KML.
Llama a plotIPs(pcap) para generar el contenido del KML y lo combina con las cabeceras y pies.
Escribe el documento KML resultante en resultado.kml.

```
#Generaremos un archivo KML
#Un archivo KML no es más que un archivo XML compatibles con Maps de Google
kmlheader = '<?xml version="1.0" encoding="UTF-8"?> \n<kml xmlns="http://www.opengis.net/kml/2.2">\n<Document>\n'\
'<Style id="transBluePoly">' \
            '<LineStyle>' \
            '<width>1.5</width>' \
            '<color>501400E6</color>' \
            '</LineStyle>' \
            '</Style>'
kmlfooter = '</Document>\n</kml>\n'
kmldoc=kmlheader+plotIPs(pcap)+kmlfooter
```

```
        #print(kmldoc)
        with open ('resultado.kml','w') as f:
            f.write(kmldoc)

    except :
        print ("Se han producido errores al ejecutar el código")
```

La función plotIPs sirve para procesar los paquetes dentro del archivo pcap y convertir la información a un formato legible.

```
def plotIPs(pcap):
    '''Leemos el fichero pcap y lo recorremos de manera que obtenemos informacion
    de sus IP y a la vez con dichas IP obtenemos sus datos de posicionamiento GPS
    '''
    kmlPts = ''
    listaDatos=[]
    for (ts, buf) in pcap:
        try:
            Direccion={}
            eth = dpkt.ethernet.Ethernet(buf)
            if eth.type==ETH_TYPE_IP:#peticion de tipo IP
                ip = eth.data
                #convierto la direccion a un formato legible
                src = socket.inet_ntoa(ip.src)
                dst = socket.inet_ntoa(ip.dst)

                KML, datoPosicion = retKML(dst, src)
                kmlPts = kmlPts + KML
                listaDatos.append(datoPosicion)


        except:
            pass
    with open ('resultado.json','w') as f:
        json.dump(listaDatos,f)
    return kmlPts
```

Dentro de la función plotIPs se exporta la información a un archivo .json:

```
with open ('resultado.json','w') as f:
    json.dump(listaDatos,f)
return kmlPts
```

La función retKML sirve para generar KML y obtener datos de Ips.

```
def retKML(dstip, srcip):
    '''
    Obtenemos para cada petición ip los datos de posicionamientos GPS
    y lo convertimos a registro XML
    con el formato KML
    '''
    Direccion={}
    #Saco información de destino
    dst = gi.record_by_name(dstip)
    #información de mi maquina publica
    src = gi.record_by_name('83.36.10.85')
    Direccion["ip_origen"]=src
    Direccion["ip_destion"]=dst

    #esta la cojo a "mano" usad esta pues es con la que hemos creado los datos
    #podríamos cogerla, pero sería complicar el ejemplo más
    try:
        dstlongitude = dst['longitude']
        dstlatitude = dst['latitude']
        srclongitude = src['longitude']
        srclatitude = src['latitude']
        kml = (
            '<Placemark>\n'
            '<name>%s</name>\n'
            '<extrude>1</extrude>\n'
            '<tessellate>1</tessellate>\n'
            '<styleUrl>#transBluePoly</styleUrl>\n'
            '<LineString>\n'
            '<coordinates>%6f,%6f\n%6f,%6f</coordinates>\n'
            '</LineString>\n'
            '</Placemark>\n'
        )%(dstip, dstlongitude, dstlatitude, srclongitude, srclatitude)
        return kml, Direccion
    except:
        return '',{}
```

En resultado.json, vemos la ip origen y destino de la trama formateada para que sea legible.