



## Contenido

Wireshark .....	2
1. Descripción .....	2
2. Instalación .....	3
3. Configuración de Wireshark y menús .....	4
4. Demos.....	11
4.1. Ping .....	11
4.2. Detección ataque DDoS .....	12
4.3. Comprobar balanceo .....	16
4.4. Envío SCP .....	19
4.5. Obtener contraseña por HTTP .....	20
5. Bibliografía.....	22

# Wireshark

## 1. Descripción

Wireshark es el analizador de protocolos de red más importante y usado en el mundo. Es el estándar en muchas empresas comerciales y sin fines de lucro, agencias gubernamentales e instituciones educativas, siendo utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica.

Las mayores ventajas frente a otros analizadores de protocolos es tener una interfaz gráfica y muchas opciones de organización y filtrado de información. Así, permite ver todo el tráfico que pasa a través de una red estableciendo la configuración en modo promiscuo (captura del tráfico).

Permite examinar datos de una red en uso o de un archivo de captura salvado en disco. Se puede analizar la información capturada, a través de los detalles y sumarios por cada paquete.

Wireshark incluye un completo lenguaje para filtrar lo que queremos ver y la habilidad de mostrar el flujo reconstruido de una sesión de TCP.

Wireshark es software libre, y se ejecuta sobre la mayoría de sistemas operativos Unix y compatibles, incluyendo Linux, Solaris, FreeBSD, NetBSD, OpenBSD, Android y Mac OS X, así como en Microsoft Windows.

El desarrollo de Wireshark prospera gracias a las contribuciones voluntarias de expertos en redes en todo el mundo y es la continuación del proyecto Ethernet, iniciado por Gerald Combs en 1998.

Para asegurarnos que la captura que realicemos sea completa, deberemos ejecutar Wireshark como súper usuario, también se puede configurar para que la captura la realicen usuarios normales del sistema, pero para asegurarnos al 100% es recomendable ejecutar como administrador.

Si quisiéramos guardar directamente el tráfico analizado, podemos usar una herramienta que viene incorporada en Wireshark llamada tcpdump, la cual tiene el mismo funcionamiento que Wireshark, pero almacenando los datos en un fichero para luego ser analizados.

También existe otra herramienta llamada Tshark, la cual es una versión de Wireshark, pero en línea de comandos

## 2. Instalación

Wireshark está disponible para múltiples sistemas operativos, en su página oficial podemos encontrar los instaladores para Windows de 64 y 32 bits, un portable de 32 bits para Windows, un instalador para macOS y el código fuente, el cual puede ser utilizado para distribuciones Linux, adicionalmente también encontramos la documentación:

<https://www.wireshark.org/download.html>

En Linux, además podemos instalar directamente desde el terminal desde los repositorios oficiales con el siguiente comando:

```
sudo apt install wireshark
```

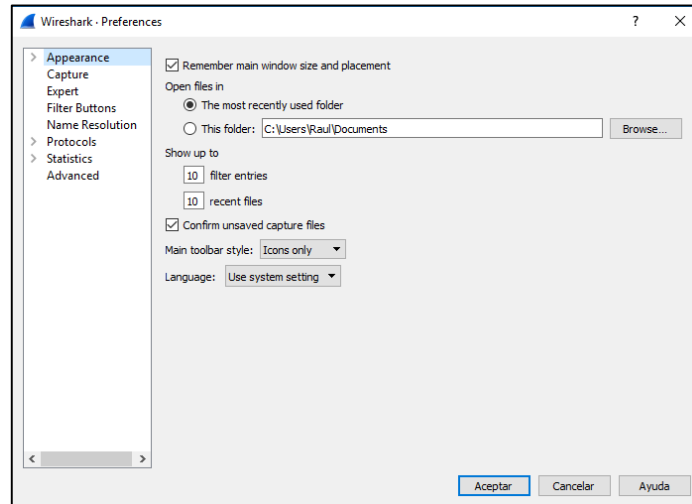
Si no queremos instalarlo de los repositorios oficiales, podemos instalarlo con los comandos:

```
sudo add-apt-repository ppa:wireshark-dev/stable  
sudo apt update  
sudo apt install wireshark
```

Otra forma de instalarlo en Ubuntu, es mediante el centro de software del sistema.

### 3. Configuración de Wireshark y menús

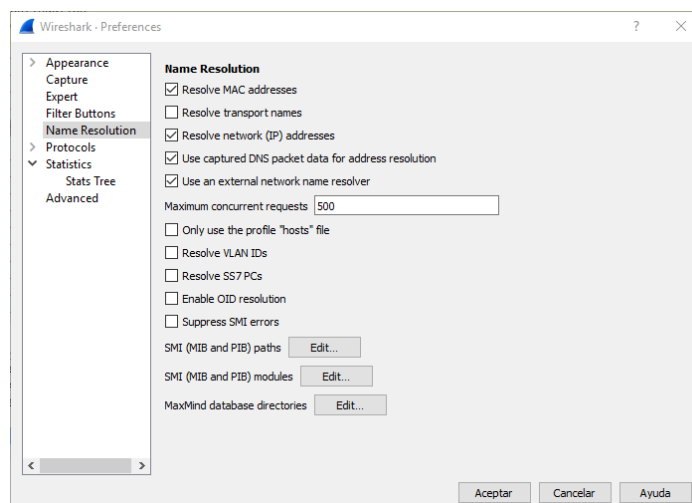
Antes de empezar a usar Wireshark, hay algunas cosas que debemos configurar para una correcta captura del tráfico, esto lo hacemos desde la pestaña “Edit” en la opción “Preferences”, donde veremos la siguiente imagen:



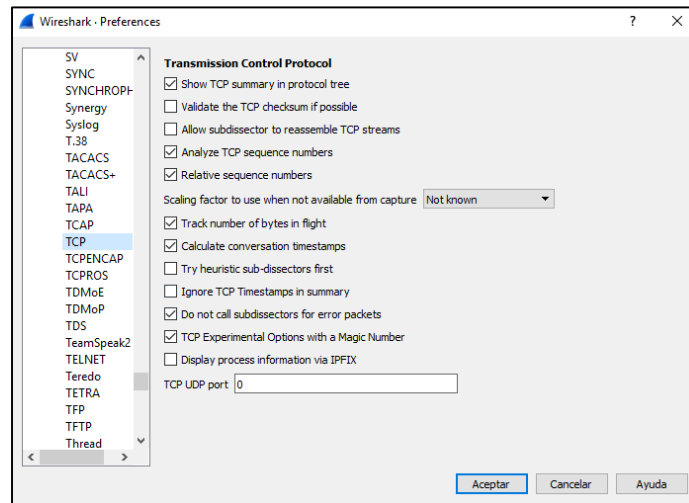
En el menú lateral izquierdo, podemos ver las diferentes ventanas que podremos configurar, las más importantes son:

- Name Resolution
- Protocols
- Statistics

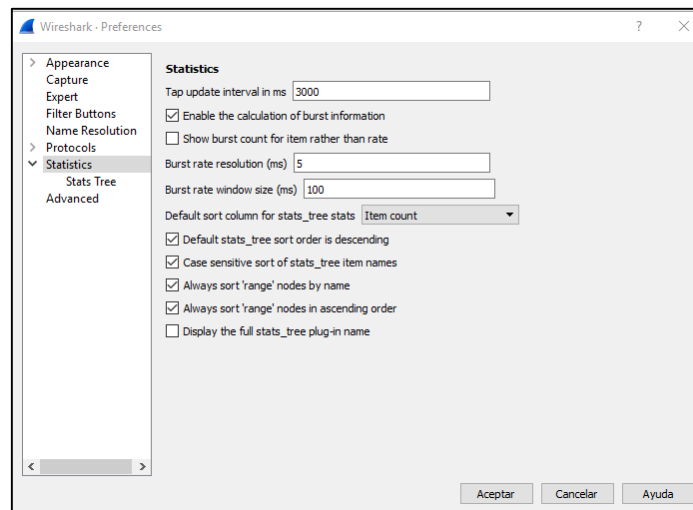
En la ventana de “Name Resolution”, podremos configurar si queremos que Wireshark resuelva el nombre del dominio y nos lo muestre en vez de la dirección IP si está disponible, también podemos marcar opciones para que resuelva los identificadores de las Virtual LAN o las direcciones MAC.



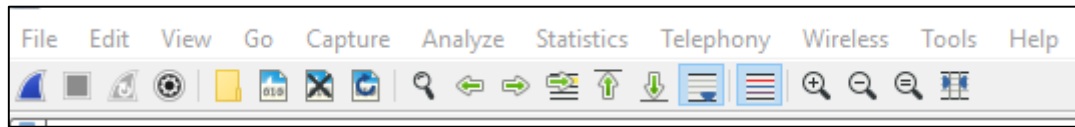
En la ventana “Protocols”, podemos ver una lista de todos los protocolos de los que dispone Wireshark, que no son pocos, por ejemplo, en el protocolo TCP, podemos hacer que no permita re ensamblar las transmisiones TCP.



Y la ventana “Statistics”, donde podremos configurar como se nos muestra la información:



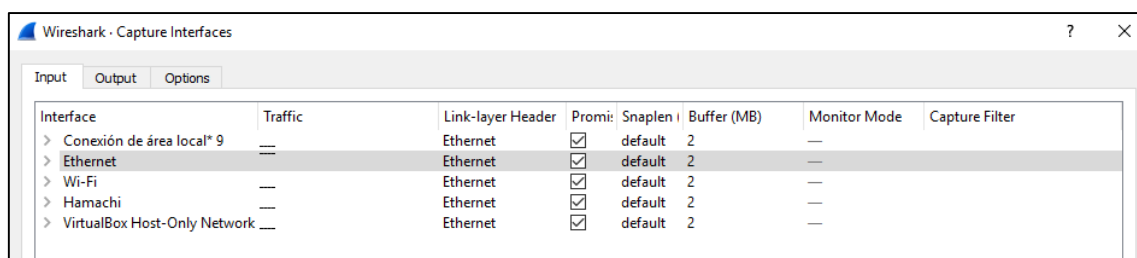
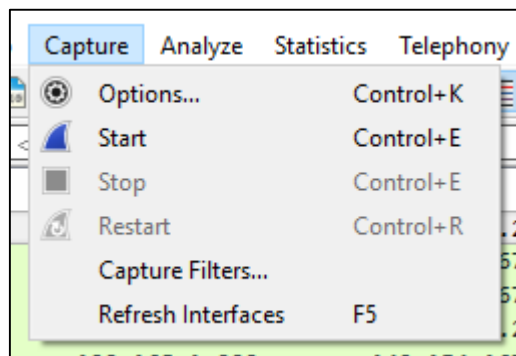
Además, Wireshark dispone de una serie de menús en la parte superior, donde podremos manipular y cambiar la forma en la que tratamos el tráfico capturado.



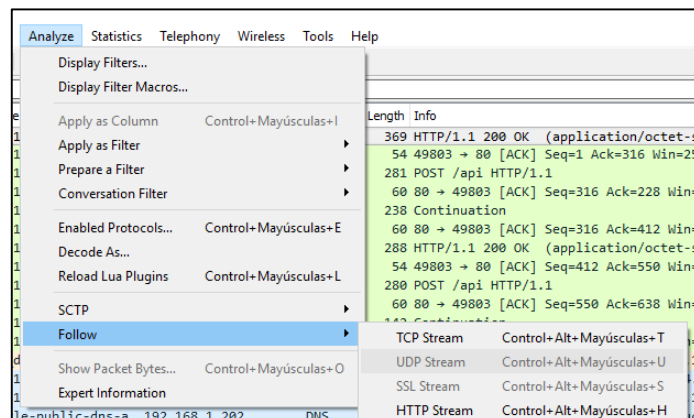
Algunos de los menús que consideramos que son los más importantes son los siguientes:

- Capture
- Analyze
- Statistics
- Tools
- Telephony
- Wireless

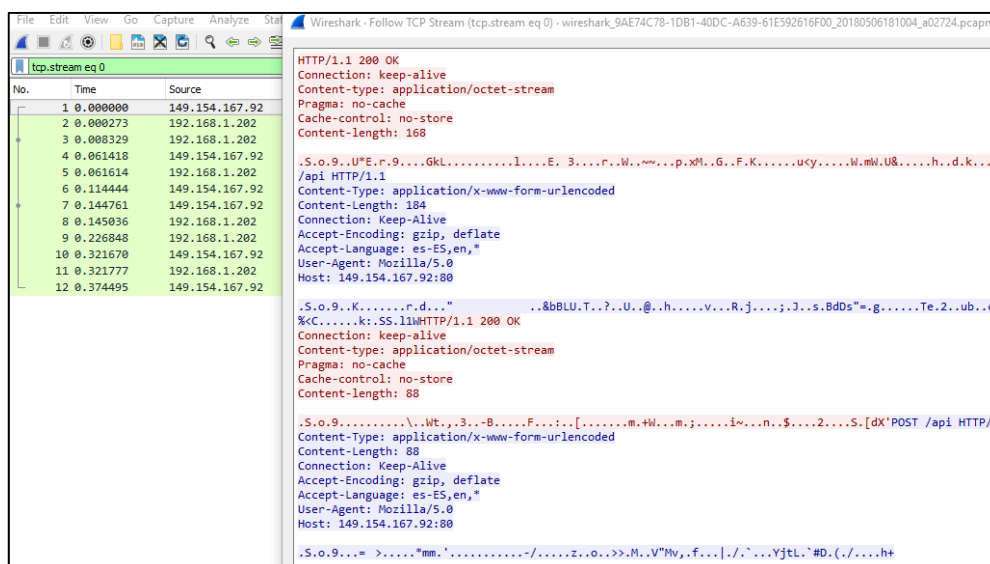
El menú “Capture” nos permite configurar la forma en la que capturamos el tráfico, permitiendo elegir la red en la que capturar, iniciar la captura, pararla o reiniciarla, y capturar según unos filtros dados.



El menú “Analyze”, nos permite analizar el tráfico capturado de forma más individual, así podemos centrarnos en lo que realmente queremos analizar sin desviarnos por capturas intermedias, como alguna captura DNS esporádica.



La opción más interesante que hemos encontrado en este menú, ha sido la de “TCP Stream” y la de “HTTP Stream”, las cuales nos crean un filtrado de la opción elegida, y abren una ventana con información de la traza como podemos ver en la imagen inferior:





El siguiente menú es “Statistics”, este menú nos permite ver el tráfico de distintas formas, ya sea como diagrama de flujo, gráficos de I/O, puntos finales o incluso mirar las estadísticas de IPv4 o IPv6.

Wireshark · Endpoints · wireshark\_vboxnet0\_20180506165138\_BFV7Wm

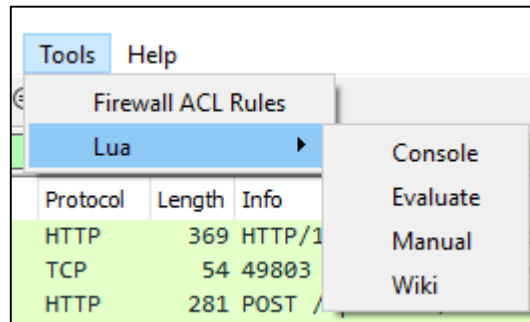
Ethernet · 5		IPv4 · 7		IPv6 · 2		TCP · 65539		UDP · 10					
Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	AS Number	Country	City	Latitude	Longitude		
91.189.88.149	6	444	0	0	6	444	AS41231 Canonical Ltd	United Kingdom	—	51.500000	-0.130000		
91.189.91.26	6	444	0	0	6	444	AS41231 Canonical Ltd	United States	Boston, MA	42.358398	-71.059799		
91.189.91.157	1	90	0	0	1	90	AS41231 Canonical Ltd	United States	Boston, MA	42.358398	-71.059799		
192.168.56.1	227.934	35 M	12	1643	227.922	35 M	—	—	—	—	—		
192.168.56.125	227.935	35 M	227.935	35 M	0	0	—	—	—	—	—		
192.168.56.255	3	276	0	0	3	276	—	—	—	—	—		
224.0.0.251	9	1367	0	0	9	1367	—	—	—	—	—		



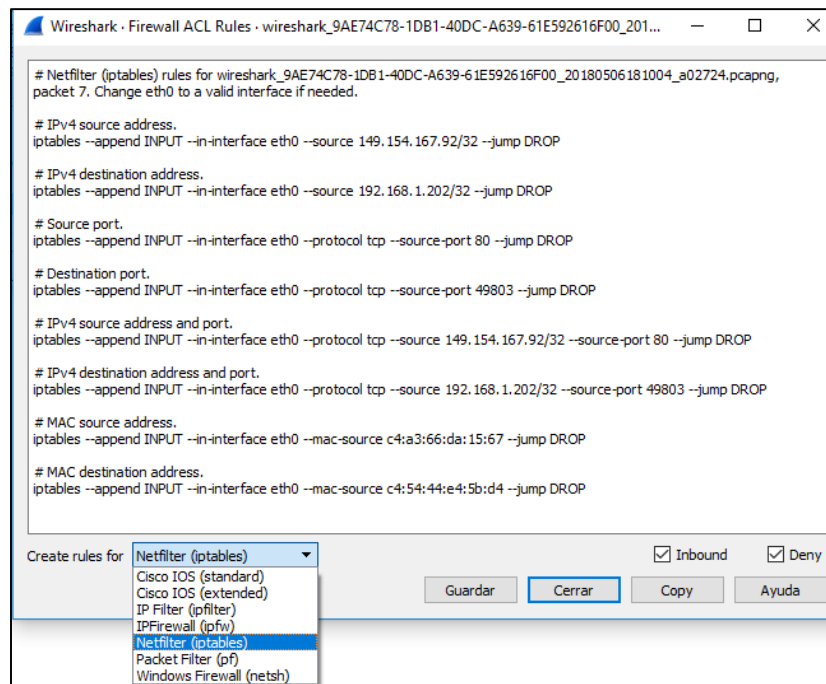
Statistics Telephony Wireless Tools Help

- Capture File Properties Control+Alt+Mayúsculas+C
- Resolved Addresses
- Protocol Hierarchy
- Conversations
- Endpoints
- Packet Lengths
- I/O Graph
- Service Response Time
- DHCP (BOOTP) Statistics
- ONC-RPC Programs
- 29West
- ANCP
- BACnet
- Collectd
- DNS
- Flow Graph
- HART-IP
- HPFEEDS
- HTTP
- HTTP2
- Sametime
- TCP Stream Graphs
- UDP Multicast Streams
- F5
- IPv4 Statistics
- IPv6 Statistics

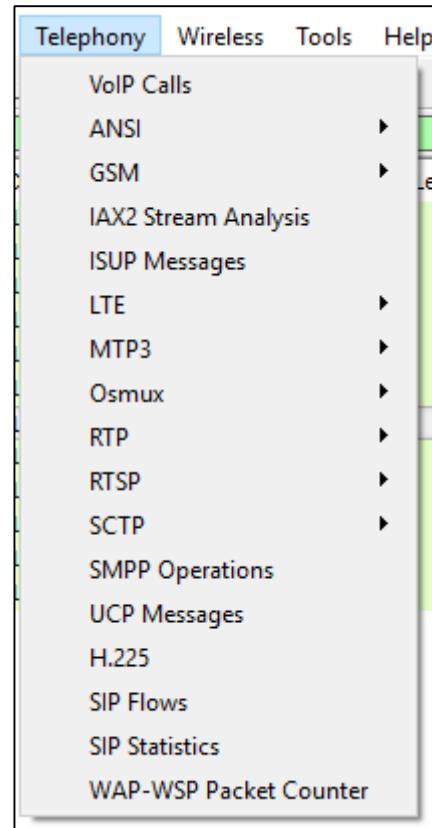
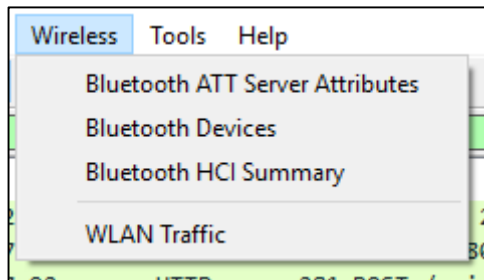
El ultimo menú importante a nuestro parecer, es “Tools”, en el cual, podemos establecer las reglas del firewall que usa Wireshark para analizar el tráfico.



En la siguiente imagen podemos ver las reglas iptables que genera para un perfil propio:



Finalmente, creemos que debemos mencionar estos dos menús: “Telephony” y “Wireless”, los cuales nos permiten analizar el tráfico generado en Llamadas o las conexiones mediante Bluetooth en el menú Wireless.



## 4. Demos

En esta parte, vamos a realizar una serie de pruebas en las que veremos el funcionamiento de Wireshark y cómo interpretar el tráfico asociado a cada evento.

### 4.1. Ping

El programa ping usa el protocolo ICMP. Por tanto, para comprobar el uso de ping deberemos filtrar dicho protocolo.

Lanzamos ping a Google y capturamos la red. Filtramos el protocolo ICMP.

Ahora podemos comprobar en la interfaz de Wireshark que nuestra máquina (192.168.1.204) manda una solicitud a Google (8.8.8.8), y a su vez, Google responde.

icmp						
No.	Time	Source	Destination	Protocol	Length	Info
5	4.917428134	192.168.1.204	188.78.168.217	ICMP	120	Destination unreachable (Port unreachable)
15	28.145204441	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=1/256, ttl=64 (reply in 16)
16	28.163913253	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=1/256, ttl=56 (request in 15)
17	29.147040198	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=2/512, ttl=64 (reply in 18)
18	29.166175422	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=2/512, ttl=56 (request in 17)
20	30.148489016	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=3/768, ttl=64 (reply in 21)
21	30.167371590	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=3/768, ttl=56 (request in 20)
23	31.150542412	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=4/1024, ttl=64 (reply in 24)
24	31.169122465	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=4/1024, ttl=56 (request in 23)
26	32.152289087	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=5/1280, ttl=64 (reply in 27)
27	32.171593070	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=5/1280, ttl=56 (request in 26)
30	33.153738868	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=6/1536, ttl=64 (reply in 31)
31	33.172689097	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=6/1536, ttl=56 (request in 30)
35	34.154794300	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=7/1792, ttl=64 (reply in 36)
36	34.173511156	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=7/1792, ttl=56 (request in 35)
37	35.156613401	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=8/2048, ttl=64 (reply in 38)
38	35.174988302	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=8/2048, ttl=56 (request in 37)
39	36.158141491	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=9/2304, ttl=64 (reply in 40)
40	36.176754136	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=9/2304, ttl=56 (request in 39)
41	37.160006419	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=10/2560, ttl=64 (reply in 42)
42	37.178453049	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=10/2560, ttl=56 (request in 41)
44	38.161642900	192.168.1.204	8.8.8.8	ICMP	98	Echo (ping) request id=0x0bbd, seq=11/2816, ttl=64 (reply in 45)
45	38.180199927	8.8.8.8	192.168.1.204	ICMP	98	Echo (ping) reply id=0x0bbd, seq=11/2816, ttl=56 (request in 44)

Ahora hacemos un ping desde otra máquina de la red (192.168.1.133) a la máquina que tiene Wireshark (192.168.1.204).

Al capturar la red y filtrar el protocolo ICMP podemos comprobar que efectivamente, la máquina con IP 192.168.1.133 manda una solicitud a la máquina con IP 192.168.1.204, y a su vez esta le responde.

88	46.702087069	192.168.1.204	172.217.17.14	TCP	66	32982 → 443 [ACK] Seq=47 Ack=47 Win=256 Len=0 TSval=1292771 TSecr=4160
89	51.331482242	192.168.1.133	192.168.1.204	ICMP	74	Echo (ping) request id=0x0001, seq=303/12033, ttl=128 (reply in 92)
90	51.331510285	IntelCor_5c:70:10	Broadcast	ARP	42	Who has 192.168.1.133? Tell 192.168.1.204
91	51.334073220	HewlettP_e7:2d:48	IntelCor_5c:70:10	ARP	60	192.168.1.133 is at fc:15:b4:e7:2d:48
92	51.334080096	192.168.1.204	192.168.1.133	ICMP	74	Echo (ping) reply id=0x0001, seq=303/12033, ttl=128 (reply in 94)
93	52.252949348	192.168.1.133	192.168.1.204	ICMP	74	Echo (ping) request id=0x0001, seq=304/12289, ttl=128 (request in 93)
94	52.252973721	192.168.1.204	192.168.1.133	ICMP	74	Echo (ping) reply id=0x0001, seq=304/12289, ttl=64 (request in 93)
95	53.276972198	192.168.1.133	192.168.1.204	ICMP	74	Echo (ping) request id=0x0001, seq=305/12545, ttl=128 (request in 96)
96	53.276991142	192.168.1.204	192.168.1.133	ICMP	74	Echo (ping) reply id=0x0001, seq=305/12545, ttl=64 (request in 95)
97	54.307941987	192.168.1.133	192.168.1.204	ICMP	74	Echo (ping) request id=0x0001, seq=306/12801, ttl=128 (reply in 98)
98	54.307964807	192.168.1.204	192.168.1.133	ICMP	74	Echo (ping) reply id=0x0001, seq=306/12801, ttl=64 (request in 97)

Como podemos ver en la imagen superior, al seleccionar una línea, Wireshark nos indica donde están las filas relacionadas. En este ejemplo, al seleccionar el ping request, Wireshark nos marca a la izquierda donde está el ping reply asociado a ese ping request. Esto muy útil cuando tenemos que analizar grandes cantidades de tráfico, o donde una operación consta de varias partes, como puede ser una petición http.

## 4.2. Detección ataque DDoS

En este apartado vamos a ver cómo detectar un ataque DDoS con Wireshark. Un ataque DDoS (Denial of Service) es un ataque de denegación de servicio, este consiste en la saturación de los puertos con múltiples flujos de información, haciendo que un recurso o servicio sea inaccesible.

Para simular el ataque DDoS hemos utilizado la herramienta hping3 desde una máquina virtual con IP (192.168.56.101) a otra la cual está haciendo peticiones ping a google constantemente (192.168.56.102).

Para instalar la herramienta hping3 ejecutamos el siguiente comando en la terminal de Ubuntu:

```
sudo apt install hping3
```

Una vez instalado, podemos iniciar el ataque con el siguiente comando:

```
sudo hping3 -c 100 -d 100 -S -w 64 -p 80 --flood 192.168.56.102
```

Donde los parámetros dados corresponden con:

- -c --> Numero de paquetes a enviar.
- -d --> Longitud del mensaje a enviar.
- -S --> Bandera SYN.
- -w --> Tamaño paquete TCP.
- -p --> Puerto al que se envían los paquetes.
- --flood --> Los paquetes se envían en tiempo real de forma masiva.

Y acto seguido, en la maquina atacada, la cual está esnifando la red, empieza a aparecer una cantidad de trafico grandísima, ralentizando toda la máquina virtual, en la imagen siguiente podemos ver que, en casi 4 segundos, ha generado unos 75000 envíos.

No.	Time	Source	Destination	Protocol	Length	Info
156020	3.883702000	192.168.56.101	192.168.56.102	TCP	54	http > 65531 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156021	3.883708000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156022	3.883710000	192.168.56.101	192.168.56.102	TCP	54	http > 65532 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156023	3.883766000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156024	3.883770000	192.168.56.101	192.168.56.102	TCP	54	http > 65533 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156025	3.883837000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156026	3.883842000	192.168.56.101	192.168.56.102	TCP	54	http > 65534 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156027	3.883852000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156028	3.883854000	192.168.56.101	192.168.56.102	TCP	54	http > 65535 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156029	3.883865000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156030	3.883867000	192.168.56.101	192.168.56.102	TCP	54	http > 0 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156031	3.883876000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156032	3.883878000	192.168.56.101	192.168.56.102	TCP	54	http > tcpmux [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156033	3.883886000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156034	3.883888000	192.168.56.101	192.168.56.102	TCP	54	http > compressnet [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156035	3.883943000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156036	3.883947000	192.168.56.101	192.168.56.102	TCP	54	http > compressnet [RST, ACK] Seq=1 Ack=121 Win=0 Len=0
156037	3.884017000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]
156038	3.884020000	192.168.56.101	192.168.56.102	TCP	54	http > 0 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0

Como podemos ver en la imagen anterior, las filas rojas corresponden al envío de la conexión desde el atacante, y en oscuro la respuesta. Las líneas en rojo indican que la conexión ha sido cerrada [RST, ACK].

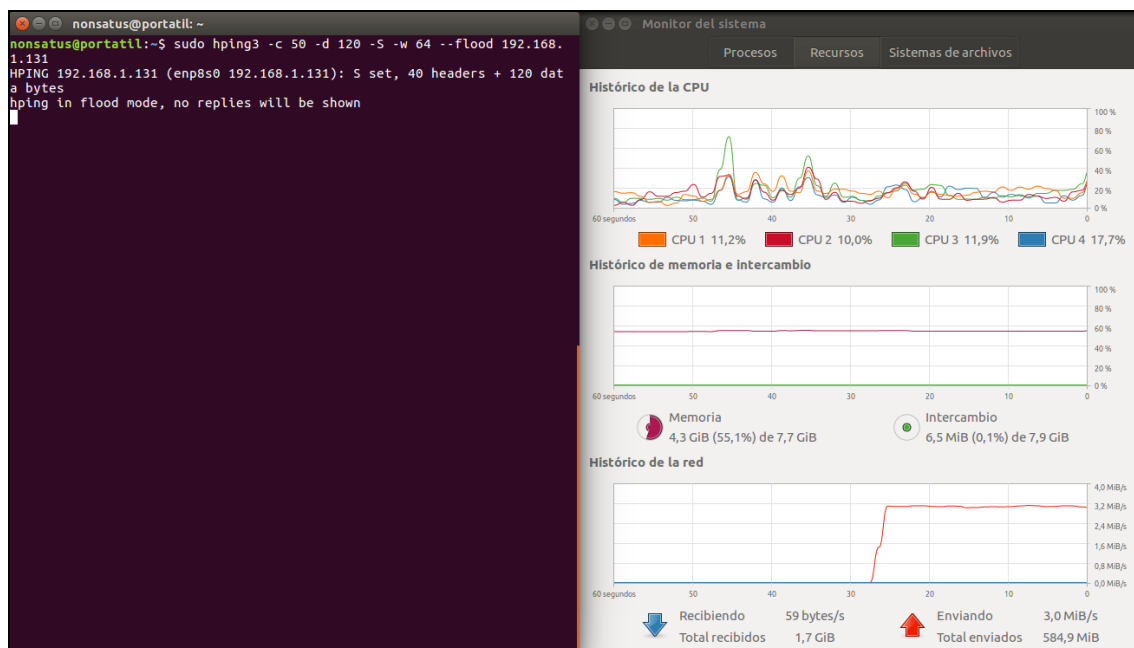
A continuación, vamos a ver un ataque DDoS mientras se ejecutan un ping a google en la maquina atacada. Como podemos ver en la siguiente imagen, cuando se empieza a realizar el ataque, los pings dejan de aparecer, para ser sustituidos por las trazas DDoS.

35	9.014163000	10.0.2.15	google-public-dns-a.g	ICMP	98	Echo (ping) request	id=0x1122, seq=29/7424, ttl=64 (reply in
36	9.029061000	google-public-dns-a.google.com	10.0.2.15	ICMP	98	Echo (ping) reply	id=0x1122, seq=29/7424, ttl=63 (request i
37	10.016893000	10.0.2.15	google-public-dns-a.g	ICMP	98	Echo (ping) request	id=0x1122, seq=30/7680, ttl=64 (reply in
38	10.031689000	google-public-dns-a.google.com	10.0.2.15	ICMP	98	Echo (ping) reply	id=0x1122, seq=30/7680, ttl=63 (request i
39	11.018958000	10.0.2.15	google-public-dns-a.g	ICMP	98	Echo (ping) request	id=0x1122, seq=31/7936, ttl=64 (reply in
40	11.033134000	google-public-dns-a.google.com	10.0.2.15	ICMP	98	Echo (ping) reply	id=0x1122, seq=31/7936, ttl=63 (request i
41	11.981345000	192.168.56.102	192.168.56.101	TCP	174	[TCP segment of a reassembled PDU]	
42	11.981428000	192.168.56.101	192.168.56.102	TCP	54	http > netdb-export [RST, ACK] Seq=1 Ack=121 Win=0 Len=0	
43	11.982672000	192.168.56.102	192.168.56.101	TCP	174	[TCP segment of a reassembled PDU]	
44	11.982782000	192.168.56.101	192.168.56.102	TCP	54	http > streetperfect [RST, ACK] Seq=1 Ack=121 Win=0 Len=0	
45	11.982721000	192.168.56.102	192.168.56.101	TCP	174	[TCP segment of a reassembled PDU]	
46	11.982724000	192.168.56.101	192.168.56.102	TCP	54	http > intersan [RST, ACK] Seq=1 Ack=121 Win=0 Len=0	
47	11.982731000	192.168.56.102	192.168.56.101	TCP	174	[TCP segment of a reassembled PDU]	

En el momento en el que se acaba el ataque, podemos ver como el ping continúa haciéndose.

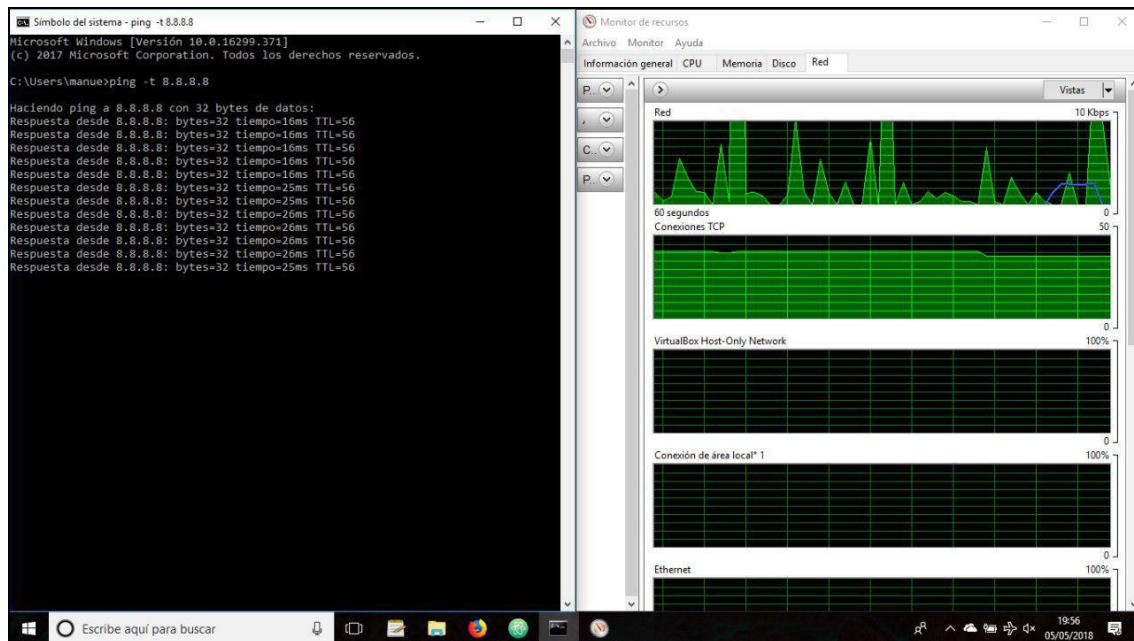
173012	26.765624000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]	
173013	26.765627000	192.168.56.101	192.168.56.102	TCP	54	http > 59536 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0	
173014	26.765634000	192.168.56.102	192.168.56.101	TCP	174	[TCP Port numbers reused] [TCP segment of a reassembled PDU]	
173015	26.765636000	192.168.56.101	192.168.56.102	TCP	54	http > 59537 [RST, ACK] Seq=1 Ack=121 Win=0 Len=0	
173016	27.058408000	10.0.2.15	google-public-dns-a.g	ICMP	98	Echo (ping) request	id=0x1122, seq=47/12032, ttl=64 (reply in
173017	27.073331000	google-public-dns-a.google.com	10.0.2.15	ICMP	98	Echo (ping) reply	id=0x1122, seq=47/12032, ttl=63 (request
173018	28.060134000	10.0.2.15	google-public-dns-a.g	ICMP	98	Echo (ping) request	id=0x1122, seq=48/12288, ttl=64 (reply in
173019	28.075853000	google-public-dns-a.google.com	10.0.2.15	ICMP	98	Echo (ping) reply	id=0x1122, seq=48/12288, ttl=63 (request
173020	29.062170000	10.0.2.15	google-public-dns-a.g	ICMP	98	Echo (ping) request	id=0x1122, seq=49/12544, ttl=64 (reply in

Para ver mejor el ataque, hemos simulado un ataque desde Ubuntu a una maquina Windows, a continuación, vemos la ejecución del comando hping3 y la cantidad de datos enviados en ese momento, que como podemos observar, es de 3MB/s.

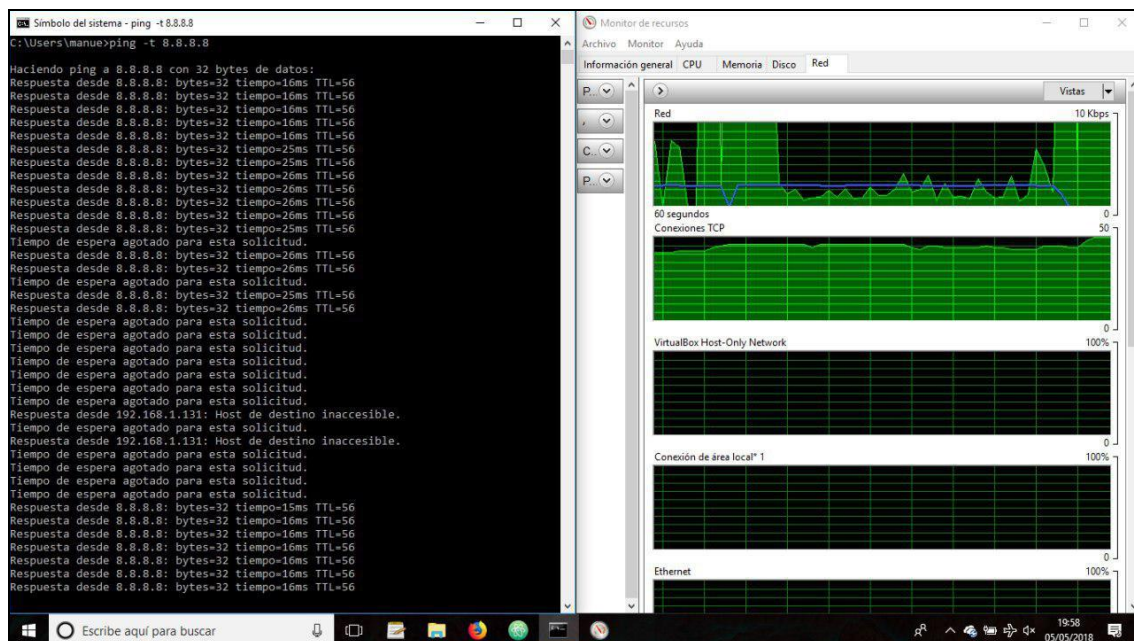




En la maquina atacada, podemos ver como el ping que se está realizando, deja de ser funcional hasta que el ataque para.

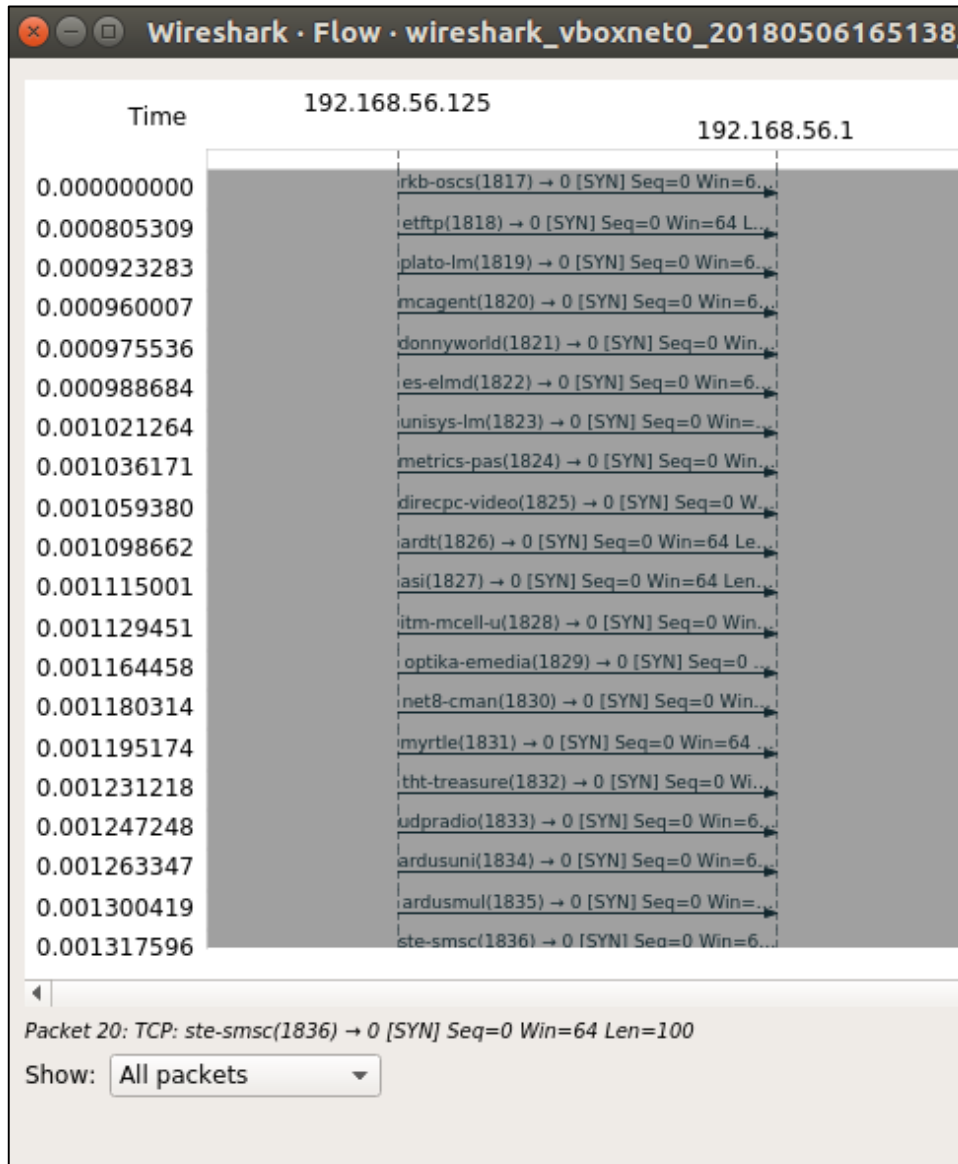


En el monitor del sistema podemos ver como al terminar el ataque, el uso de red cae.



En la siguiente imagen, podemos ver, el diagrama de flujo de un ataque DDoS, en la imagen, podemos ver dos direcciones IP:

- 192.168.56.125 (atacante)
- 192.168.56.1 (atacado) (corresponde con la maquina anfitrión)



En la imagen podemos ver como para un periodo MUY corto de tiempo (de 0 segundos a 0.0013 milisegundos) se han generado muchos intentos de conexión (20 conexiones) y podemos observar que se abren muchas conexiones [SYN] sin respuesta, lo que hace que la maquina se sature.



### 4.3. Comprobar balanceo

Ahora vamos a ver el funcionamiento del balanceador con Wireshark. Para esto, usamos una máquina virtual con Wireshark en la misma red en la que el balanceador y las maquinas servidoras están funcionando.

Primero, desde la maquina cliente (192.168.56.101) hacemos dos peticiones curl a la maquina balanceadora (192.168.56.125) y en Wireshark usamos un filtrado por http, ya que nos interesa ver los archivos que está balanceando la maquina balanceadora.

Observando las trazas, podemos ver que desde la maquina cliente, se hacen dos peticiones al balanceador mediante un método GET sobre el archivo hola.html y a continuación de dichas peticiones, obtenemos una respuesta de la maquina balanceadora con el archivo que hemos pedido en texto plano.

Ahora, si miramos la información interna de las respuestas del balanceador, veremos que se nos proporcionan los archivos de cada máquina servidora (maquina1 y maquina2) a través del balanceador.

1	0.000	192.168.56.1	192.168.56.125	TCP	74	51052 → http(80) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6330620 TSecr=0..
2	0.000	192.168.56.125	192.168.56.1	TCP	74	http(80) → 51052 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1237..
3	0.000	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=6330620 TSecr=123750
4	0.000	192.168.56.1	192.168.56.125	HTTP	153	GET /hola.html HTTP/1.1
5	0.001	192.168.56.125	192.168.56.1	HTTP	340	HTTP/1.1 200 OK (text/html)
6	0.001	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330620 TSecr=123751
7	0.002	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [FIN, ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330620 TSecr=123751
8	0.002	192.168.56.125	192.168.56.1	TCP	66	http(80) → 51052 [FIN, ACK] Seq=275 Ack=89 Win=29956 Len=0 TSval=123751 TSecr=6330620
9	0.002	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [ACK] Seq=89 Ack=276 Win=30336 Len=0 TSval=6330621 TSecr=123751
10	0.006	192.168.56.1	192.168.56.125	TCP	74	51054 → http(80) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6330622 TSecr=0..
11	0.006	192.168.56.125	192.168.56.1	TCP	74	http(80) → 51054 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1237..
12	0.006	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=6330622 TSecr=123752
13	0.006	192.168.56.125	192.168.56.1	HTTP	153	GET /hola.html HTTP/1.1
14	0.006	192.168.56.125	192.168.56.1	HTTP	340	HTTP/1.1 200 OK (text/html)
15	0.008	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330622 TSecr=123752
16	0.008	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [FIN, ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330622 TSecr=123752
17	0.008	192.168.56.125	192.168.56.1	TCP	66	http(80) → 51054 [FIN, ACK] Seq=275 Ack=89 Win=29956 Len=0 TSval=123752 TSecr=6330622
18	0.008	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [ACK] Seq=89 Ack=276 Win=30336 Len=0 TSval=6330622 TSecr=123752
19	0.018	192.168.56.1	192.168.56.125	TCP	74	51056 → http(80) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6330625 TSecr=0..
Last-Modified: Thu, 22 Mar 2018 17:17:52 GMT\r\n						
ETag: "2f-568037cd0160"\r\n						
Accept-Ranges: bytes\r\n						
Content-Length: 47\r\n						
Content-Type: text/html\r\n						
\r\n						
[HTTP response 1/1]						
[Time since request: 0.001642247 seconds]						
[Request in frame: 4]						
File Data: 47 bytes						
Line-based text data: text/html						
<HTML>\n						
<body>\n						
Maquina1\n						
</body>\n						
</HTML>\n						

1	0.000	192.168.56.1	192.168.56.125	TCP	74	51052 → http(80) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6330620 TSecr=0..
2	0.000	192.168.56.125	192.168.56.1	TCP	74	http(80) → 51052 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1237..
3	0.000	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=6330620 TSecr=123750
4	0.000	192.168.56.1	192.168.56.125	HTTP	153	GET /hola.html HTTP/1.1
5	0.001	192.168.56.125	192.168.56.1	HTTP	340	HTTP/1.1 200 OK (text/html)
6	0.001	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330620 TSecr=123751
7	0.002	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [FIN, ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330620 TSecr=123751
8	0.002	192.168.56.125	192.168.56.1	TCP	66	http(80) → 51052 [FIN, ACK] Seq=275 Ack=89 Win=29956 Len=0 TSval=123751 TSecr=6330620
9	0.002	192.168.56.1	192.168.56.125	TCP	66	51052 → http(80) [ACK] Seq=89 Ack=276 Win=30336 Len=0 TSval=6330621 TSecr=123751
10	0.006	192.168.56.1	192.168.56.125	TCP	74	51054 → http(80) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6330622 TSecr=0..
11	0.006	192.168.56.125	192.168.56.1	TCP	74	http(80) → 51054 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1237..
12	0.006	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=6330622 TSecr=123752
13	0.006	192.168.56.1	192.168.56.125	HTTP	153	GET /hola.html HTTP/1.1
14	0.008	192.168.56.125	192.168.56.1	HTTP	340	HTTP/1.1 200 OK (text/html)
15	0.008	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330622 TSecr=123752
16	0.008	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [FIN, ACK] Seq=88 Ack=275 Win=30336 Len=0 TSval=6330622 TSecr=123752
17	0.008	192.168.56.125	192.168.56.1	TCP	66	http(80) → 51054 [FIN, ACK] Seq=275 Ack=89 Win=29956 Len=0 TSval=123752 TSecr=6330622
18	0.008	192.168.56.1	192.168.56.125	TCP	66	51054 → http(80) [ACK] Seq=89 Ack=276 Win=30336 Len=0 TSval=6330622 TSecr=123752
19	0.018	192.168.56.1	192.168.56.125	TCP	74	51056 → http(80) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6330625 TSecr=0..
Last-Modified: Thu, 22 Mar 2018 17:50:11 GMT\r\n						
ETag: "2f-5680370713980"\r\n						
Accept-Ranges: bytes\r\n						
Content-Length: 47\r\n						
Content-Type: text/html\r\n						
\r\n						
[HTTP response 1/1]						
[Time since request: 0.001402343 seconds]						
[Request in frame: 13]						
File Data: 47 bytes						
Line-based text data: text/html						
<HTML>\n						
<body>\n						
Maquina2\n						
</body>\n						
</HTML>\n						

Además, si no filtramos por protocolo HTML, podemos ver todas trazas que se realizan para el curl.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.101	192.168.56.105	TCP	74	57034 > http [SYN] Seq=0 Win=
2	0.000251000	192.168.56.105	192.168.56.101	TCP	74	http > 57034 [SYN, ACK] Seq=
3	0.000266000	192.168.56.101	192.168.56.105	TCP	66	57034 > http [ACK] Seq=1 Ack=
4	0.000542000	192.168.56.101	192.168.56.105	HTTP	153	GET /hola.html HTTP/1.1
5	0.000692000	192.168.56.105	192.168.56.101	TCP	66	http > 57034 [ACK] Seq=1 Ack=
6	0.000943000	192.168.56.105	192.168.56.101	HTTP	340	HTTP/1.1 200 OK (text/html)
7	0.000949000	192.168.56.101	192.168.56.105	TCP	66	57034 > http [ACK] Seq=88 Ac
8	0.001231000	192.168.56.101	192.168.56.105	TCP	66	57034 > http [FIN, ACK] Seq=
9	0.001397000	192.168.56.105	192.168.56.101	TCP	66	http > 57034 [FIN, ACK] Seq=
10	0.001403000	192.168.56.101	192.168.56.105	TCP	66	57034 > http [ACK] Seq=89 Ac
11	5.010198000	CadmusCo_48:d4:71	CadmusCo_c1:8a:bc	ARP	60	Who has 192.168.56.101? Tel
12	5.010216000	CadmusCo_c1:8a:bc	CadmusCo_48:d4:71	ARP	42	192.168.56.101 is at 08:00:2

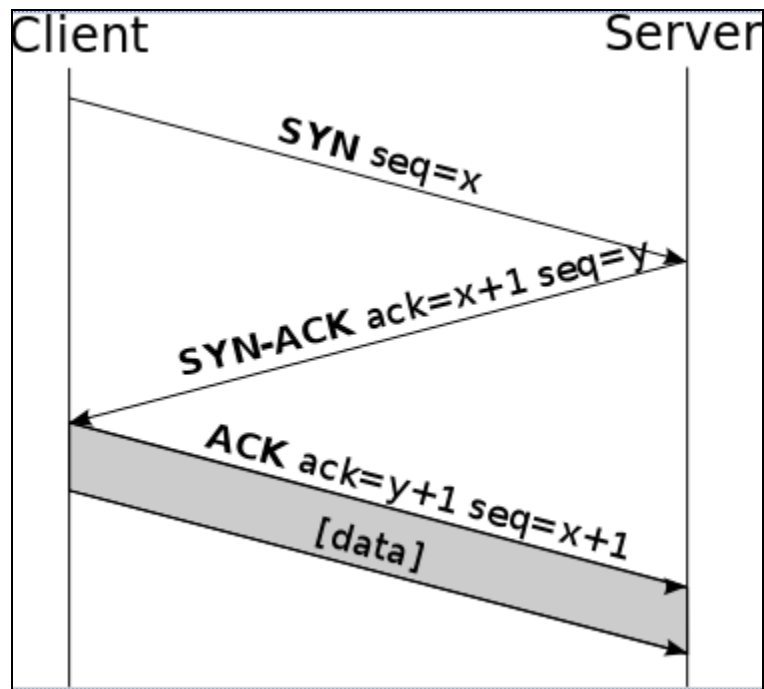
▶ Frame 6: 340 bytes on wire (2720 bits), 340 bytes captured (2720 bits) on interface 0  
 ▶ Ethernet II, Src: CadmusCo 48:d4:71 (08:00:27:48:d4:71), Dst: CadmusCo\_c1:8a:bc (08:00:27:c1:8a:bc)  
 ▶ Internet Protocol Version 4, Src: 192.168.56.105 (192.168.56.105), Dst: 192.168.56.101 (192.168.56.101)  
 ▶ Transmission Control Protocol, Src Port: http (80), Dst Port: 57034 (57034), Seq: 1, Ack: 88, Len: 274  
 ▶ Hypertext Transfer Protocol  
 ▼ Line-based text data: text/html  
 <HTML>\n  
 <body>\n  
 Maquina1\n  
 </body>\n  
 </HTML>\n

0000 08 00 27 c1 8a bc 08 00 27 48 d4 71 08 00 45 00 ' ' 'H n F

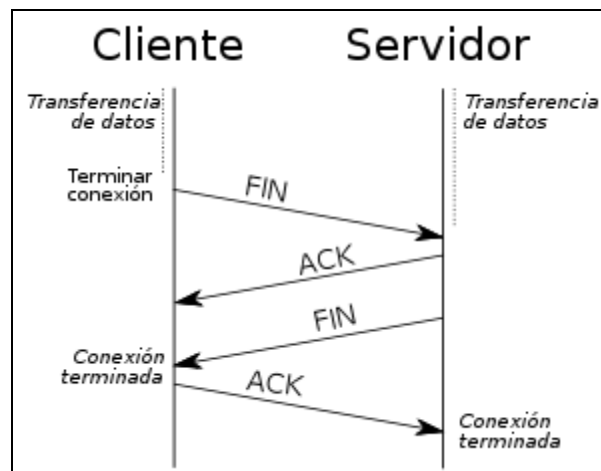
En la imagen superior, podemos ver mucha información:

- Líneas 1, 2 y 3: Inicio de la negociación en tres pasos:
  - 1: [SYN] (Cliente 192.168.56.101 -> Server 192.168.56.105)
  - 2: [SYN, ACK] (Server 192.168.56.105 -> Cliente 192.168.56.101)
  - 3: [ACK] (Cliente 192.168.56.101 -> Server 192.168.56.105)
- Líneas 4 y 6: Petición y envío del curl:
  - 4: Get /hola.html HTTP/1.1 (Petición del cliente 192.168.56.101 al servidor 192.168.56.105)
  - 6: HTTP/1.1 200 OK (text/html) (Recibo del fichero HTML solicitado desde la maquina servidora 192.168.56.105 al cliente 192.168.56.101)
- Líneas 8 y 9: Finalización de la negociación en tres pasos:
  - 8: Fin de la conexión del cliente al servidor.
  - 9: Fin de la conexión del servidor al cliente.

### Inicio 3-way-handshake



### Fin 3-way-handshake



#### 4.4. Envío SCP

A continuación, vamos a ver el funcionamiento de la herramienta scp al transferir archivos de una maquina a otra. En esta ocasión, se va a transferir un archivo desde la maquina anfitrión a una maquina servidora de Virtual Box.

Wireshark está instalado ahora en la maquina anfitrión, por lo que, en las capturas, esta aparece con la ip correspondiente a la puerta de enlace 192.168.56.1, mientras que la máquina que recibe tiene la ip 192.168.56.105.

Scp o Secure Copy es una herramienta que se utiliza para la transferencia de archivos de forma segura, usando el protocolo SSH o Secure Shell, por tanto, no vamos a poder ver la información que se transfiere, pero sí de donde a donde y como.

Para transferir un archivo, debemos escribir el siguiente comando:

```
sudo scp usuario@host:directorio/ArchivoOrigen ArchivoDestino
ó
scp ArchivoOrigen usuario@host:directorio/ArchivoDestino
```

En este caso, como el archivo se envía desde la maquina anfitrión, usaremos el siguiente comando, el cual transferirá el archivo al directorio de usuario de la maquina receptora:

```
sudo scp file rauldpm@192.168.56.105:/home/rauldpm/
```

Al ejecutar el comando, vemos lo siguiente con Wireshark:

1 0.000	192.168.56.1	192.168.56.105	TCP	74	55452 → ssh(22) [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=6839498 TSecr=0
2 0.000	192.168.56.105	192.168.56.1	TCP	74	ssh(22) → 55452 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=633105
3 0.000	192.168.56.1	192.168.56.105	TCP	66	55452 → ssh(22) [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=6839498 TSecr=633105
4 0.004	192.168.56.105	192.168.56.1	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.2)
5 0.004	192.168.56.1	192.168.56.105	TCP	66	55452 → ssh(22) [ACK] Seq=1 Ack=42 Win=29312 Len=0 TSval=6839500 TSecr=633106
6 0.012	192.168.56.1	192.168.56.105	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4)
7 0.013	192.168.56.105	192.168.56.1	TCP	66	ssh(22) → 55452 [ACK] Seq=42 Ack=42 Win=29856 Len=0 TSval=633108 TSecr=6839502
8 0.013	192.168.56.1	192.168.56.105	SSHv2	1402	Client: Key Exchange Init
9 0.013	192.168.56.105	192.168.56.1	TCP	66	ssh(22) → 55452 [ACK] Seq=42 Ack=1378 Win=31872 Len=0 TSval=633108 TSecr=6839502
10 0.014	192.168.56.105	192.168.56.1	SSHv2	1042	Server: Key Exchange Init
11 0.016	192.168.56.1	192.168.56.105	SSHv2	114	Client: Diffie-Hellman Key Exchange Init
12 0.021	192.168.56.105	192.168.56.1	SSHv2	430	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=84)
13 0.060	192.168.56.1	192.168.56.105	TCP	66	55452 → ssh(22) [ACK] Seq=1426 Ack=1382 Win=34048 Len=0 TSval=6839514 TSecr=633110
14 1.706	192.168.56.1	192.168.56.105	SSHv2	82	Client: New Keys
15 1.743	192.168.56.105	192.168.56.1	TCP	66	ssh(22) → 55452 [ACK] Seq=1382 Ack=1442 Win=31872 Len=0 TSval=633541 TSecr=6839925
16 1.996	192.168.56.1	192.168.56.105	SSHv2	110	Client: Encrypted packet (len=44)
17 1.996	192.168.56.105	192.168.56.1	TCP	66	ssh(22) → 55452 [ACK] Seq=1382 Ack=1486 Win=31872 Len=0 TSval=633604 TSecr=6839997
18 1.996	192.168.56.105	192.168.56.1	SSHv2	110	Server: Encrypted packet (len=44)
19 1.996	192.168.56.1	192.168.56.105	TCP	66	55452 → ssh(22) [ACK] Seq=1486 Ack=1426 Win=34048 Len=0 TSval=6839998 TSecr=633604
20 1.996	192.168.56.1	192.168.56.105	SSHv2	134	Client: Encrypted packet (len=68)
21 1.998	192.168.56.105	192.168.56.1	SSHv2	118	Server: Encrypted packet (len=52)
22 1.998	192.168.56.1	192.168.56.105	SSHv2	438	Client: Encrypted packet (len=372)
23 1.998	192.168.56.105	192.168.56.1	SSHv2	118	Server: Encrypted packet (len=52)

En la imagen anterior, podemos ver 3 recuadros de colores.

- El recuadro verde, corresponde con el 3-way-handshake
- El recuadro rojo, corresponde con el intercambio de claves entre la maquina anfitrión y la maquina receptora
- El recuadro morado, corresponde con el envío a través de ssh por el puerto 22 de los paquetes encriptados.

Finalmente, una vez que se ha realizado la transferencia, vemos como se cierra el 3-way-handshake:

49	3.672...	192.168.56.1	192.168.56.105	TCP	66	55452 → ssh(22)	[FIN, ACK]
50	3.673...	192.168.56.105	192.168.56.1	TCP	66	ssh(22) → 55452	[ACK] Seq=
51	3.676...	192.168.56.105	192.168.56.1	TCP	66	ssh(22) → 55452	[FIN, ACK]
52	3.676...	192.168.56.1	192.168.56.105	TCP	66	55452 → ssh(22)	[ACK] Seq=

#### 4.5. Obtener contraseña por HTTP

Finalmente, vamos a demostrar lo poco seguro que es el protocolo http cuando hay información personal de por medio.

El protocolo http (protocolo de transferencia de hipertexto) es un protocolo de comunicación que usa texto plano para la transferencia de información, pero que no está cifrada de ninguna forma, lo que lo hace muy vulnerable a diversos ataques como un MITM o un eavesdropping (ataques de escuchas telefónicas, por ejemplo).

Hoy en día hay muy pocas webs que aun usen el protocolo http, debido a que su predecesor, el protocolo https, ofrece esa seguridad que http no tiene al cifrar la información.

Para esta demostración, hemos usado como ejemplo, la página: <http://elcotodecaza.com>, una página orientada a la caza, pero que la cual dispone de un servicio de compra de productos de caza, cuya página principal, usa protocolo http, cosa más que suficiente para poder obtener información personal.



Ahora simplemente tenemos que observar Wireshark a la espera de que nuestra víctima inicie sesión con sus credenciales, para esta prueba, vamos a ingresar la siguiente información:

- Nombre de usuario: Victima
- Contraseña: pass

Nombre de usuario\*

Victima

Contraseña\*

....

Iniciar sesión

[Deseo registrarme](#) | [Recuperar mi contraseña](#)

Y al hacer click la víctima, en Wireshark nos aparecería, al igual que en las otras demos, una conexión por http con la información introducida mediante el método POST:

No.	Time	Source	Destination	Protocol	Length	Info
58	1.488	192.168.1.202	104.18.46.40	HTTP	798	POST /user?destination=portada HTTP/1.1 (application/x-www-form-urlencoded)
95	4.891	104.18.46.40	192.168.1.202	HTTP	74	HTTP/1.1 200 OK (text/html)

Referer: http://www.elcotodecaza.com/\r\n

Content-Type: application/x-www-form-urlencoded\r\n

Content-Length: 64\r\n

Cookie: has\_js=1; \_\_cfduid=d0808cebc8685f5347c122bb8e4c80d461525192006; SESS2a00d8913ec85ac5889c2efa683f6fcd=75349ea78c382de91d079984eeffa3857; \_ga=GA1.2.8389\r\n

Connection: keep-alive\r\n

Upgrade-Insecure-Requests: 1\r\n

\r\n

[Full request URI: http://www.elcotodecaza.com/user?destination=portada]

[HTTP request 1/1]

[Response in frame: 95]

File Data: 64 bytes

HTML Form URL Encoded: application/x-www-form-urlencoded

Form item: "form\_id" = "user\_login"

Form item: "name" = "Victima"

Key: name

Value: Victima

Form item: "pass" = "pass"

Key: pass

Value: pass

Form item: "op" = "Iniciar sesión"

## 5. Bibliografía

- Instalación y uso de Wireshark
  - <https://www.askmetutorials.com/2017/06/install-wireshark-227-in-ubuntu-1604.html>
  - <http://tuxylinux.com/instalar-y-configurar-wireshark-en-linux/>
  - <https://www.enlinux.org/instalar-wireshark-en-gnulinix-debian-ubuntu-server-ubuntu-desktop/>
- Instalación y uso de hping3
  - <https://www.redeszone.net/gnu-linux/hping3-manual-de-utilizacion-de-esta-herramienta-para-manipular-paquetes-tcp-ip/>
  - <https://kali-linux.net/article/hping3/>
- Uso de Wireshark
  - <https://comofriki.com/como-usar-wireshark-capturar-filtrar-analizar-paquetes/>
  - <https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/certifseguridadanalisistraficowireshark.pdf>
  - <https://www.lifewire.com/wireshark-tutorial-4143298>
- Ataques DDoS
  - <https://es.wikipedia.org/wiki/Ataquededenegaci%C3%B3ndeservicio>
  - <https://www.lifewire.com/wireshark-tutorial-4143298>
  - <https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/certifseguridadanalisistraficowireshark.pdf>