

HDC



Wireshark

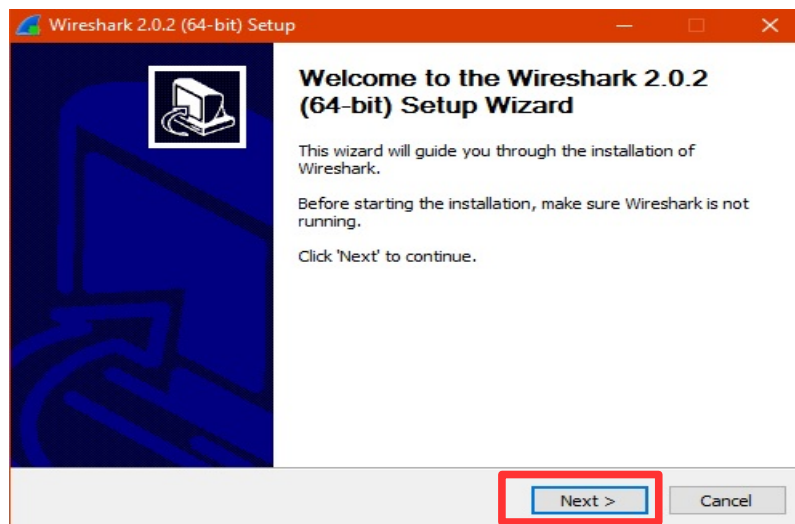
“¿Wire... qué? ¿Se come?”

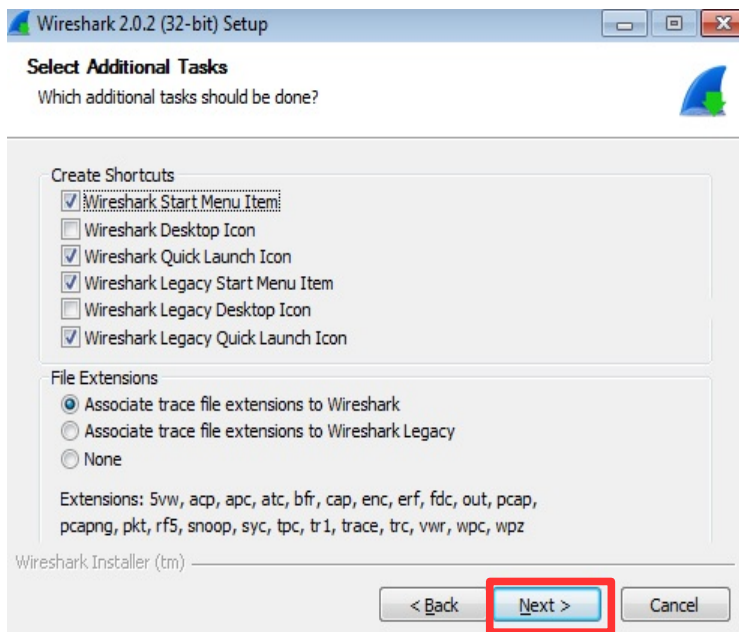
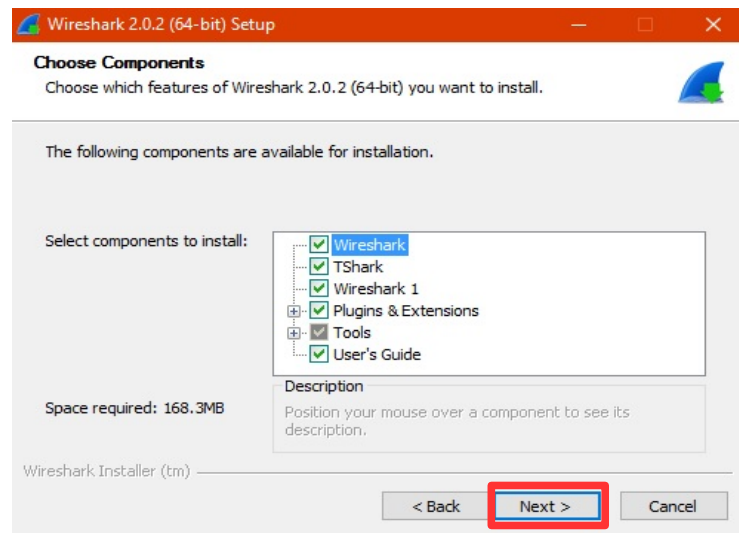
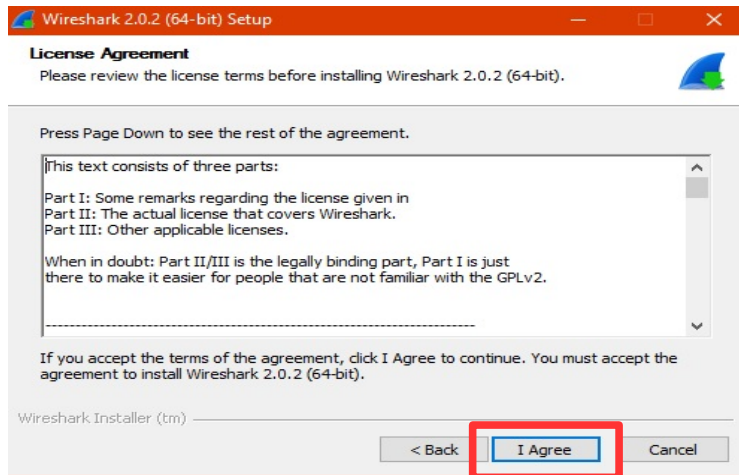
Buenos días, Manolo. No, no se come. El **Wireshark** es un **analizador de protocolos de red**. Esto quiere decir que toma lo que va viajando por la red y lo muestra al usuario con los datos claros (en vez de mostrar bytes, muestra texto legible para el humano).

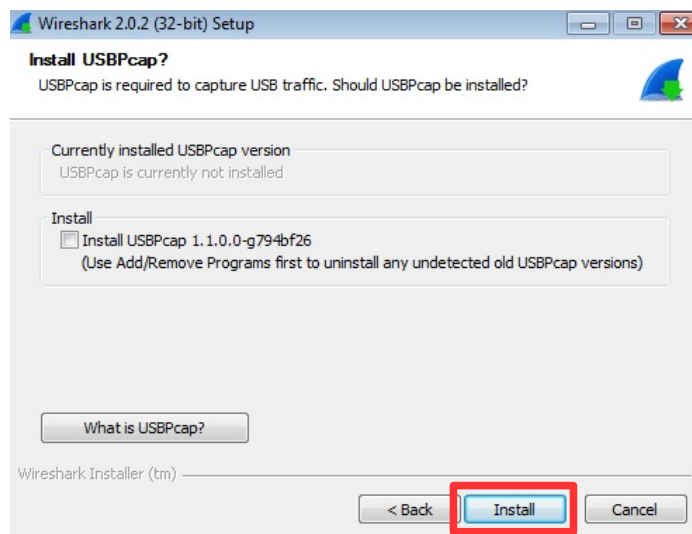
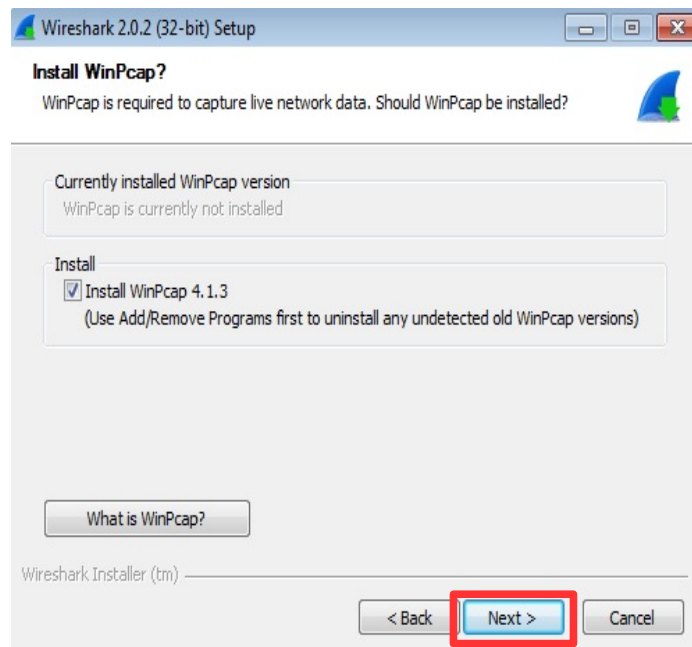
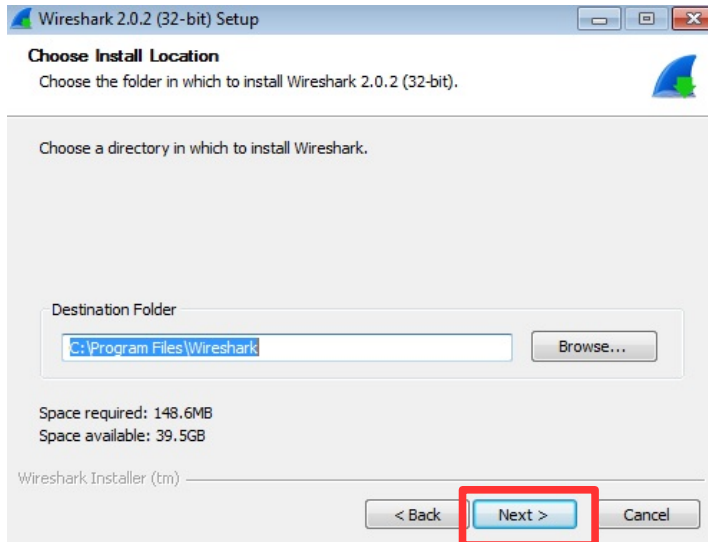
“Suena interesante. ¿Pero suena a una herramienta que sólo podría usar en GNU/Linux.”

Por suerte, existe apoyo para las 3 plataformas más populares. Pero nosotros aún nos estamos manejando con **Windows** (claro, el que quiera hacerlo con su S.O. habitual y es otro, hágalo), así que lo haremos allí. Primero a **descargarlo** de la página oficial: www.wireshark.org

Para **instalarlo**, los pasos son los siguientes:





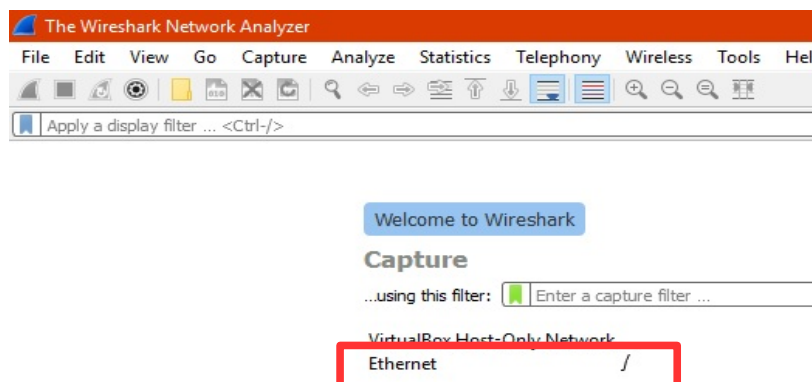


Fácil, ¿Verdad?

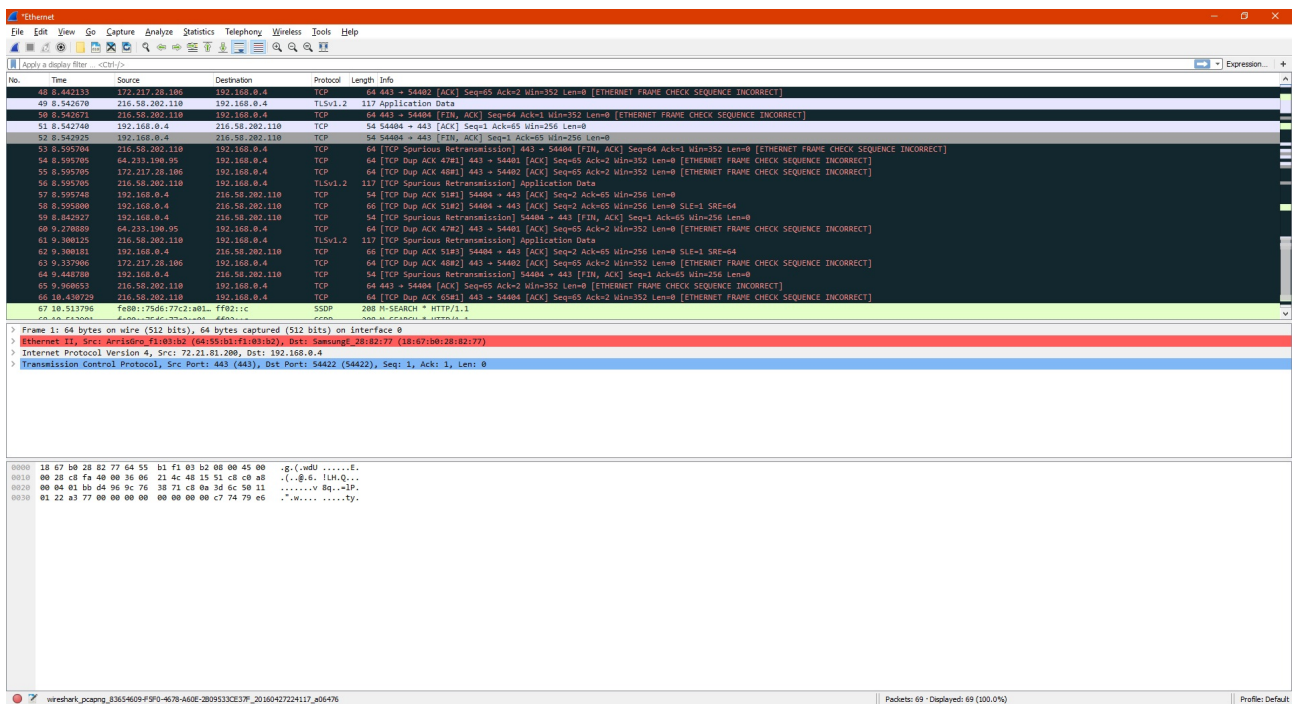
“¡Espera! ¿Qué es WinPcap y por qué debo instalarlo?”

Bien, Manolo. Me gusta esa curiosidad. **WinPcap** es la **herramienta** de Windows para **capturar y enviar los paquetes de red**. Esto es lo que necesita el Wireshark para informarnos qué está pasando.

Una vez lo abrimos, al software, nos encontramos con una pantalla como esta. Elegimos nuestra **placa de red** para que analice. En mi caso, voy a clicar sobre la **ethernet** (o sea, red cableada).



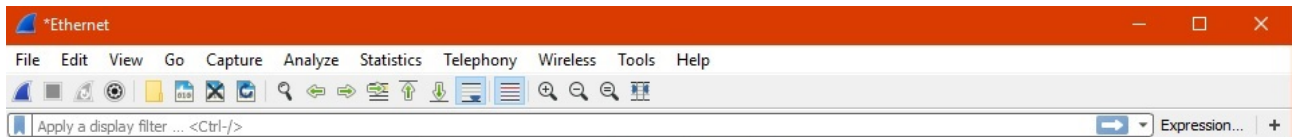
Abrió la interfaz de análisis, y...



“Wow. Con esto parece que estás hackeando la NASA.”

Bueno, estamos un poco lejos de eso. Pero te dejo soñar. **Vayamos por partes**, dijo Jack.

Lo **primero** que tenemos, en la cabecera es el **menú**, la **barra de herramientas** y el textbox de los **filtros** (que es en lo que hoy nos vamos a concentrar).



Lo más destacado aquí es que los primeros tres botones de la izquierda de la barra de herramientas corresponden a **Iniciar captura**, **Detener captura** y **Reiniciar captura**; respectivamente.

Luego, tenemos en esta ventana los **paquetes de datos** que llegan, organizados y vistosos.

A screenshot of the Wireshark packet list pane. It displays a table with columns for No., Time, Source, Destination, Protocol, Length, and Info. The table contains 17 rows of captured packets, including TCP and TLSv1.2 traffic between 31.13.73.1 and 192.168.0.4.

No.	Time	Source	Destination	Protocol	Length	Info
17	6.986107	31.13.73.1	192.168.0.4	TCP	64	443 → 50070 [ACK] Seq=1 Ack=212 Win=2043 Len=0 [ETHERNET II]
18	6.986346	31.13.73.1	192.168.0.4	TLSv1.2	96	Application Data
19	6.986555	31.13.73.1	192.168.0.4	TLSv1.2	100	Application Data
20	6.986586	192.168.0.4	31.13.73.1	TCP	54	50070 → 443 [ACK] Seq=212 Ack=89 Win=253 Len=0
21	7.029057	31.13.73.1	192.168.0.4	TLSv1.2	177	Application Data
22	7.093578	192.168.0.4	31.13.73.1	TCP	54	50070 → 443 [ACK] Seq=212 Ack=212 Win=253 Len=0
23	7.131464	192.168.0.4	31.13.73.1	TLSv1.2	281	Application Data
24	7.263846	31.13.73.1	192.168.0.4	TLSv1.2	96	Application Data
25	7.318874	192.168.0.4	31.13.73.1	TCP	54	50070 → 443 [ACK] Seq=439 Ack=254 Win=258 Len=0
26	7.348206	31.13.73.1	192.168.0.4	TLSv1.2	1285	Application Data
27	7.348475	31.13.73.1	192.168.0.4	TCP	1464	[TCP segment of a reassembled PDU]
28	7.348516	192.168.0.4	31.13.73.1	TCP	54	50070 → 443 [ACK] Seq=439 Ack=2895 Win=258 Len=0
29	7.348924	31.13.73.1	192.168.0.4	TLSv1.2	1283	Application Data
30	7.349027	31.13.73.1	192.168.0.4	TLSv1.2	600	Application Data

Por columnas tenemos: un **Nº de paquete** (que es un id de referencia para el Wireshark); un **tiempo relativo** en el cual se ha encontrado el paquete; **de donde viene, hacia donde va, el protocolo** que utiliza el paquete, **el largo** (en bytes) e **información complementaria**. Luego analizaremos algunos paquetes.

Después de que dimos para que analice, el Wireshark está **captando paquetes a tiempo real**, así que verán que incrementa la cantidad.

Más abajo, el Wireshark, **analiza el paquete** que tenemos seleccionado con la **info organizada** para fácil visualización.


```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
> Ethernet II, Src: Giga-Byt_ec:f5:58 (90:2b:34:ec:f5:58), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Address Resolution Protocol (request)
```

Y por último, el **paquete recibido sin analizar**, con los bytes pelados.

```
0000  ff ff ff ff ff ff 90 2b 34 ec f5 58 08 06 00 01  .....+ 4..X....
0010  08 00 06 04 00 01 90 2b 34 ec f5 58 c0 a8 00 03  .....+ 4..X....
0020  00 00 00 00 00 00 c0 a8 00 08 00 00 00 00 00 00  .....
0030  00 00 00 00 00 00 00 00 00 00 00 00  .....

```

wireshark_pcapng_83654609-F5F0-4678-A60E-2B09533CE37F_20160502153653_a06124 | Packets: 37 · Displayed: 37 (100.0%) | Profile: Default

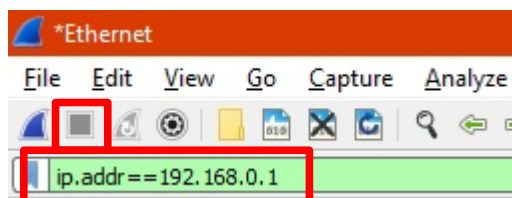
Ahora sí. Pasemos directamente a usar los **filtros** y las oportunidades que nos ofrece este gran soft.

En la **ventana de filtro**, vamos a escribir:

ip.addr == X.X.X.X

Donde **X.X.X.X** es el **número IP por el que queremos filtrar**. Es decir, sólo aparecerán los paquetes en el que el destino o la fuente, tengan esa IP.

Para el ejemplo, use la IP que tiene el router al cual estoy conectado y anteriormente dejé el Wireshark corriendo una hora hasta **parar el programa** con el segundo boton de la barra de herramientas.



No.	Time	Source	Destination	Protocol	Length	Info
9954	827.915504	192.168.0.1	239.255.255.250	SSDP	390	NOTIFY * HTTP/1.1
9955	827.925472	192.168.0.1	239.255.255.250	SSDP	394	NOTIFY * HTTP/1.1
9956	827.935489	192.168.0.1	239.255.255.250	SSDP	386	NOTIFY * HTTP/1.1
10249	857.943807	192.168.0.1	239.255.255.250	SSDP	342	NOTIFY * HTTP/1.1
10250	857.953718	192.168.0.1	239.255.255.250	SSDP	398	NOTIFY * HTTP/1.1
10251	857.963722	192.168.0.1	239.255.255.250	SSDP	326	NOTIFY * HTTP/1.1
10252	857.973696	192.168.0.1	239.255.255.250	SSDP	318	NOTIFY * HTTP/1.1
10253	857.983681	192.168.0.1	239.255.255.250	SSDP	362	NOTIFY * HTTP/1.1
10254	857.993686	192.168.0.1	239.255.255.250	SSDP	338	NOTIFY * HTTP/1.1
10255	858.003782	192.168.0.1	239.255.255.250	SSDP	392	NOTIFY * HTTP/1.1
10256	858.013703	192.168.0.1	239.255.255.250	SSDP	390	NOTIFY * HTTP/1.1

Acá me llevé la pequeña sorpresa de que el servicio **SSDP no estaba deshabilitado** y que la red estaba inundada de estos paquetes, que podrían mantener una elevación pequeña del ping de la red. En fin, **si es un servicio que no se usa, por seguridad y practicidad debe darse de baja.**

El protocolo SSDP -resumido- es un servicio de detección de dispositivos aprovechando el uPnP (universal plug and play) y que entre sí puedan compartir servicios de manera más rápida. Está diseñado para redes caseras. Ahora, marco uno de los paquetes con un click y me fijo en la sección inferior los datos del paquete.

```

> Frame 9954: 390 bytes on wire (3120 bits), 390 bytes captured (3120 bits) on interface 0
v Ethernet II, Src: ArrisGro_f1:03:b2 (64:55:b1:f1:03:b2), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
  > Destination: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
  > Source: ArrisGro_f1:03:b2 (64:55:b1:f1:03:b2)
    Type: IPv4 (0x0800)
v Internet Protocol Version 4, Src: 192.168.0.1, Dst: 239.255.255.250
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 376
  Identification: 0xdead (57005)
  > Flags: 0x00
  Fragment offset: 0
  Time to live: 4
  Protocol: UDP (17)
  > Header checksum: 0x2624 [validation disabled]
  Source: 192.168.0.1
  Destination: 239.255.255.250
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
v User Datagram Protocol, Src Port: 1901 (1901), Dst Port: 1900 (1900)
  Source Port: 1901
  Destination Port: 1900
  Length: 356
  > Checksum: 0xabf1 [validation disabled]
  [Stream index: 0]
> Hypertext Transfer Protocol

```

Abro las 3 secciones del medio y nos encontramos con toda esta información:

1. En la primer sección, nos encontramos con las direcciones **MAC** y la **versión de IP usada** (IPv4 en este caso).
2. En la segunda sección, **vemos las IPs de destino y origen, el TTL del paquete, el protocolo** que utiliza (UDP o TCP), el **checksum**, y varias cositas más.
3. En la tercer sección, tenemos los **puertos origen y destino**.

Y verán que hay varios **“length”** repartidos. Esto es porque el paquete es **disecionado**, abierto como una **mamushka** y cada vez que vamos sacando partes, cambia el largo del paquete y el checksum.



Otro de los interminables filtros que podemos aplicar es el de **“IP origen”**. En mi caso voy a elegir la IP que tiene mi pc en la red, y ver qué tipo de comunicación está teniendo.

`ip.src==X.X.X.X`

En la red tenía la **192.168.0.4** así que allí va lo que saqué:

No.	Time	Source	Destination	Protocol	Length	Info
1572	595.486924	192.168.0.4	64.233.186.1...	TLSv1.2		571 Client Hello
1573	595.487023	192.168.0.4	64.233.190.95	TLSv1.2		107 Application Data
1574	595.487067	192.168.0.4	64.233.190.95	TLSv1.2		104 Application Data
1575	595.487109	192.168.0.4	64.233.190.95	TLSv1.2		96 Application Data

No muestro una lista larga porque de verdad que hice muchas cosas en ese tiempo y para analizar todo necesitaríamos una larga lista de clases. Por eso, ahora nos quedaremos con pocos **paquetes**. Fijémonos en estos 4 que aparecen aquí. Es algo muy interesante porque vienen de **una misma comunicación** (o por lo menos, eso parece). Empieza con un **Client Hello** de una comunicación segura en **TLS 1.2** (como lo vimos en la clase pasada) y luego está enviando datos. Lo importante es... **¿De dónde corcho es esa IP “64.233.blablalbla”?**

Así que me metí en <https://who.is> y puse la IP en el buscador. **Who Is**, es un servicio

de búsqueda de IPs y dominios para brindar datos. Veamos:

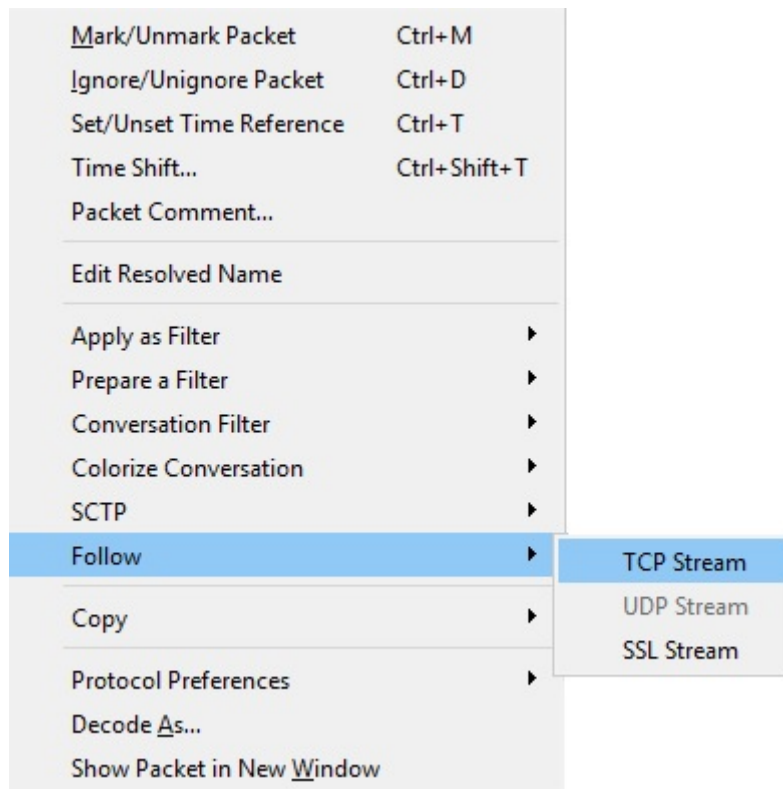
```
NetRange:      64.233.160.0 - 64.233.191.255
CIDR:          64.233.160.0/19
NetName:       GOOGLE
NetHandle:     NET-64-233-160-0-1
Parent:        NET64 (NET-64-0-0-0-0)
NetType:       Direct Allocation
OriginAS:
Organization:  Google Inc. (GOGL)
RegDate:       2003-08-18
Updated:       2012-02-24
Ref:           https://whois.arin.net/rest/net/NET-64-233-160-0

OrgName:       Google Inc.
OrgId:         GOGL
Address:        1600 Amphitheatre Parkway
City:          Mountain View
StateProv:     CA
PostalCode:    94043
Country:       US
RegDate:       2000-03-30
Updated:       2015-11-06
Ref:           https://whois.arin.net/rest/org/GOGL

OrgTechHandle: ZG39-ARIN
OrgTechName:   Google Inc
OrgTechPhone:  +1-650-253-0000
```

No profundizaré sobre esto porque más adelante lo veremos con detalles. Entonces, finalmente **la IP pertenecía a Google Inc.** Me quedo tranquilo de que no es una comunicación extraña. De igual manera vamos a ver qué nos trae este protocolo y si es cierto lo que vimos la clase anterior.

Le damos clic derecho al paquete del Client Hello y hacemos → Follow → TCP Stream. Con esto, le decimos al Wireshark que solamente nos muestre los **paquetes relacionados a esa comunicación.**



Nos salta una **ventana emergente**, que en mi caso es **ilegible** y por lo tanto la voy a cerrar.

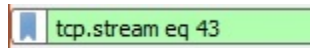


Y acá tenemos toda la **serie de paquetes** entre ambas partes.

No.	Time	Source	Destination	Protocol	Length	Info
1556	595.453581	192.168.0.4	64.233.186.1	TCP	66	51859 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
1569	595.483166	64.233.186.1	192.168.0.4	TCP	66	443 → 51859 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=1430 SACK_PERM=1 WS=128
1570	595.483236	192.168.0.4	64.233.186.1	TCP	54	51859 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1572	595.486924	192.168.0.4	64.233.186.1	TLSv1.2	571	Client Hello
1597	595.517103	64.233.186.1	192.168.0.4	TCP	64	443 → 51859 [ACK] Seq=1 Ack=518 Win=44032 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
1598	595.517314	64.233.186.1	192.168.0.4	TLSv1.2	214	Server Hello, Change Cipher Spec, Hello Request, Hello Request
1601	595.519520	192.168.0.4	64.233.186.1	TLSv1.2	270	Change Cipher Spec, Hello Request, Hello Request, Hello Request
1602	595.524781	192.168.0.4	64.233.186.1	TLSv1.2	107	Application Data
1603	595.524832	192.168.0.4	64.233.186.1	TLSv1.2	104	Application Data
1604	595.524865	192.168.0.4	64.233.186.1	TLSv1.2	96	Application Data
1615	595.548967	64.233.186.1	192.168.0.4	TLSv1.2	110	Application Data
1616	595.549163	192.168.0.4	64.233.186.1	TLSv1.2	92	Application Data
1617	595.549195	64.233.186.1	192.168.0.4	TLSv1.2	96	Application Data
1618	595.553706	64.233.186.1	192.168.0.4	TCP	64	443 → 51859 [ACK] Seq=259 Ack=879 Win=45056 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
1619	595.553928	64.233.186.1	192.168.0.4	TLSv1.2	92	Application Data
1620	595.553959	192.168.0.4	64.233.186.1	TCP	54	51859 → 443 [ACK] Seq=917 Ack=297 Win=65280 Len=0
1636	595.618401	64.233.186.1	192.168.0.4	TCP	64	443 → 51859 [ACK] Seq=297 Ack=917 Win=45056 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
2349	640.619438	192.168.0.4	64.233.186.1	TCP	55	[TCP Keep-Alive] 51859 → 443 [ACK] Seq=916 Ack=297 Win=65280 Len=1
2350	640.648837	64.233.186.1	192.168.0.4	TCP	66	[TCP Keep-Alive ACK] 443 → 51859 [ACK] Seq=297 Ack=917 Win=45056 Len=0 SLE=916 SRE=917
2656	685.661106	192.168.0.4	64.233.186.1	TCP	55	[TCP Keep-Alive] 51859 → 443 [ACK] Seq=916 Ack=297 Win=65280 Len=1
2657	685.690414	64.233.186.1	192.168.0.4	TCP	66	[TCP Keep-Alive ACK] 443 → 51859 [ACK] Seq=297 Ack=917 Win=45056 Len=0 SLE=916 SRE=917
8258	730.690592	192.168.0.4	64.233.186.1	TCP	55	[TCP Keep-Alive] 51859 → 443 [ACK] Seq=916 Ack=297 Win=65280 Len=1
8266	730.719754	64.233.186.1	192.168.0.4	TCP	66	[TCP Keep-Alive ACK] 443 → 51859 [ACK] Seq=297 Ack=917 Win=45056 Len=0 SLE=916 SRE=917
9320	775.720243	192.168.0.4	64.233.186.1	TCP	55	[TCP Keep-Alive] 51859 → 443 [ACK] Seq=916 Ack=297 Win=65280 Len=1
9321	775.748993	64.233.186.1	192.168.0.4	TCP	66	[TCP Keep-Alive ACK] 443 → 51859 [ACK] Seq=297 Ack=917 Win=45056 Len=0 SLE=916 SRE=917
9821	820.754623	192.168.0.4	64.233.186.1	TCP	55	[TCP Keep-Alive] 51859 → 443 [ACK] Seq=916 Ack=297 Win=65280 Len=1
9822	820.782741	64.233.186.1	192.168.0.4	TCP	66	[TCP Keep-Alive ACK] 443 → 51859 [ACK] Seq=297 Ack=917 Win=45056 Len=0 SLE=916 SRE=917
9994	835.541723	64.233.186.1	192.168.0.4	TLSv1.2	117	Application Data
9995	835.541724	64.233.186.1	192.168.0.4	TCP	64	443 → 51859 [FIN, ACK] Seq=360 Ack=917 Win=45056 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]

No se ve muy bien, así que haré **zoom**:

1. Aquí el **filtro** cambió. Esto es para mostrar lo que le ordenamos que haga. El **follow al stream TCP**.



2. En éste, vemos que es un **error**. En la información nos indica que el **numero de secuencia es incorrecto**. Sucede, es una conexión TCP así que necesita todos los paquetes y seguramente **lo volverá a mandar**.



3. Vemos que el Server manda a mi dispositivo, un **Server Hello** y me indica cual va a ser el **cifrado**. También le hace un **Hello Request**.

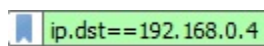


Y ya pronto empieza el **intercambio de datos** entre servidor y cliente.

Sigamos viendo **filtros**. Así como pusimos IP origen, podríamos poner **IP destino**:

ip.dst==X.X.X.X

Volveré a usar la IP del dispositivo en mi mano que era 192.168.0.4.



No.	Time	Source	Destination	Protocol	Length	Info
20908	1032.217672	181.108.6.198	192.168.0.4	DTLSv1.2		111 Application Data
20910	1032.322007	181.108.6.198	192.168.0.4	DTLSv1.2		111 Application Data
20911	1032.332407	192.168.0.2	192.168.0.4	DTLSv1.2		111 Application Data

Sigamos porque no es más que para aprender el tema de los filtros. Otra cosa que podríamos hacer es buscar los paquetes que **NO tengan relación** con cierta dirección IP. El formato del filtro sería:

!(ip.addr==X.X.X.X)

!(ip.addr==192.168.0.4)						
No.	Time	Source	Destination	Protocol	Length	Info
12019	978.347080	192.168.0.1	239.255.255...	SSDP		398 NOTIFY
12020	978.357087	192.168.0.1	239.255.255...	SSDP		326 NOTIFY
12021	978.367061	192.168.0.1	239.255.255...	SSDP		318 NOTIFY
12022	978.377043	192.168.0.1	239.255.255...	SSDP		362 NOTIFY
12023	978.387100	192.168.0.1	239.255.255...	SSDP		338 NOTIFY
12024	978.397170	192.168.0.1	239.255.255...	SSDP		392 NOTIFY
12025	978.407083	192.168.0.1	239.255.255...	SSDP		390 NOTIFY
12026	978.417080	192.168.0.1	239.255.255...	SSDP		394 NOTIFY
12027	978.427082	192.168.0.1	239.255.255...	SSDP		386 NOTIFY
12107	982.461751	fe80::10d4:2...	ff02::1:2	DHCPv6		157 Solicit
12296	989.759907			ARP		42 Who has
12310	989.844332			ARP		60 192.168
12526	994.852025			ARP		60 Who has
12527	994.852052			ARP		42 192.168
18691	1008.435513	192.168.0.1	239.255.255...	SSDP		342 NOTIFY
18692	1008.445447	192.168.0.1	239.255.255...	SSDP		398 NOTIFY

Y hasta podemos **concatenar filtros**, poniendo **&&** (que funcionaría como "and"). En este caso usé algo fácil de entender.

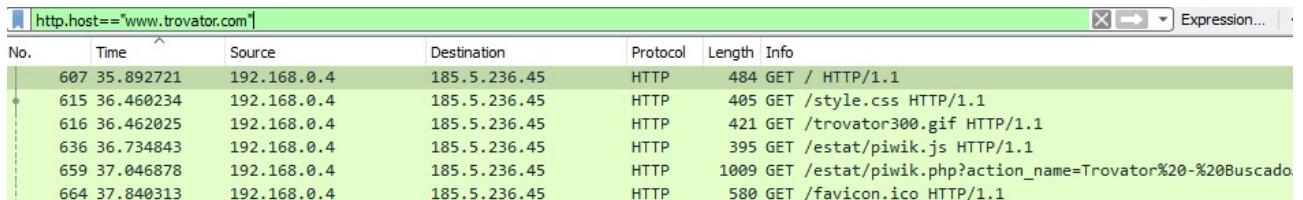
!(ip.src==X.X.X.X) && !(ip.dst==X.X.X.X)

!(ip.src == 192.168.0.1) and !(ip.dst==192.168.0.1)						
No.	Time	Source	Destination	Protocol	Length	Info
11	1.017535	Giga-Byt_ec:f5:58	Broadcast	ARP	60	Who has 192.168.0.8? Tell 192.168.0.3
12	2.017532	Giga-Byt_ec:f5:58	Broadcast	ARP	60	Who has 192.168.0.8? Tell 192.168.0.3
13	2.732424	192.168.0.4	31.13.73.1	TLSv1.2	219	Application Data
14	2.732497	192.168.0.4	31.13.73.1	TLSv1.2	100	Application Data
15	2.864259	31.13.73.1	192.168.0.4	TCP	64	443 → 50070 [ACK] Seq=1 Ack=166 Win=2043 L
16	2.864485	31.13.73.1	192.168.0.4	TCP	64	443 → 50070 [ACK] Seq=1 Ack=212 Win=2043 L
17	2.864679	31.13.73.1	192.168.0.4	TLSv1.2	96	Application Data
18	2.864679	31.13.73.1	192.168.0.4	TLSv1.2	100	Application Data
19	2.864719	192.168.0.4	31.13.73.1	TCP	54	50070 → 443 [ACK] Seq=212 Ack=89 Win=255 L
20	2.917787	31.13.73.1	192.168.0.4	TLSv1.2	177	Application Data
21	2.967951	192.168.0.4	31.13.73.1	TCP	54	50070 → 443 [ACK] Seq=212 Ack=212 Win=254
22	3.017531	Giga-Byt_ec:f5:58	Broadcast	ARP	60	Who has 192.168.0.8? Tell 192.168.0.3
23	3.115654	192.168.0.4	31.13.73.1	TLSv1.2	281	Application Data
24	3.248464	31.13.73.1	192.168.0.4	TLSv1.2	96	Application Data

Vamos por más de estos filtros. Por ejemplo, podríamos hacerlo por **direcciones http de un host específico**. El formato del filtro sería algo así:

http.host=="direccionweb"

¡Ojo! La dirección tiene que ser **exacta**, no acortada. Entré a la página de **Trovator**, un buscador, para que puedan ver el **tráfico HTTP** (porque Google usa HTTPS).

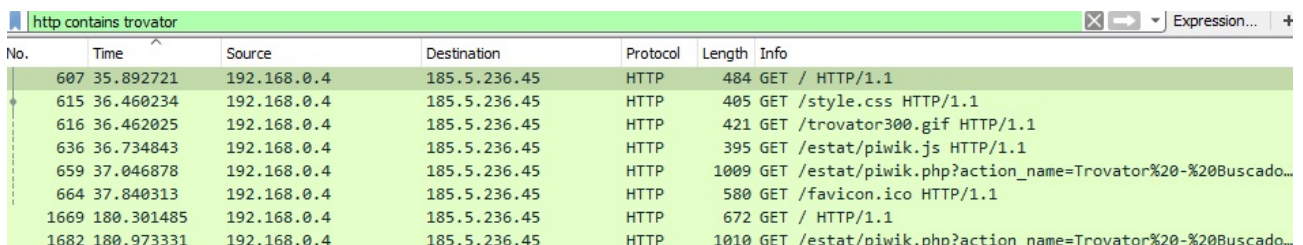


No.	Time	Source	Destination	Protocol	Length	Info
607	35.892721	192.168.0.4	185.5.236.45	HTTP	484	GET / HTTP/1.1
615	36.460234	192.168.0.4	185.5.236.45	HTTP	405	GET /style.css HTTP/1.1
616	36.462025	192.168.0.4	185.5.236.45	HTTP	421	GET /trovator300.gif HTTP/1.1
636	36.734843	192.168.0.4	185.5.236.45	HTTP	395	GET /estat/piwik.js HTTP/1.1
659	37.046878	192.168.0.4	185.5.236.45	HTTP	1009	GET /estat/piwik.php?action_name=Trovator%20-%20Buscado...
664	37.840313	192.168.0.4	185.5.236.45	HTTP	580	GET /favicon.ico HTTP/1.1

En pocas clases estaremos con la manera de aprovechar Wireshark a partir de los filtros y la navegación de una posible víctima o quizás sólo para analizar el tráfico. Ahora supongamos que no sabemos la web entera pero queremos que **contenga cierta palabra**.

Http contains "palabra"

Podríamos buscar casi cualquier cosa. Yo estoy usando esto para que lo entienda cualquiera. Hasta Manolo.



No.	Time	Source	Destination	Protocol	Length	Info
607	35.892721	192.168.0.4	185.5.236.45	HTTP	484	GET / HTTP/1.1
615	36.460234	192.168.0.4	185.5.236.45	HTTP	405	GET /style.css HTTP/1.1
616	36.462025	192.168.0.4	185.5.236.45	HTTP	421	GET /trovator300.gif HTTP/1.1
636	36.734843	192.168.0.4	185.5.236.45	HTTP	395	GET /estat/piwik.js HTTP/1.1
659	37.046878	192.168.0.4	185.5.236.45	HTTP	1009	GET /estat/piwik.php?action_name=Trovator%20-%20Buscado...
664	37.840313	192.168.0.4	185.5.236.45	HTTP	580	GET /favicon.ico HTTP/1.1
1669	180.301485	192.168.0.4	185.5.236.45	HTTP	672	GET / HTTP/1.1
1682	180.973331	192.168.0.4	185.5.236.45	HTTP	1010	GET /estat/piwik.php?action_name=Trovator%20-%20Buscado...

Para los filtros, basta con ser **creativo**. **¿Qué tipo de comunicación podría interesarnos?** Algo que me interesaría siendo un atacante, sería el intercambio de datos en el servicio **FTP**. Una vez puse el Wireshark en funcionamiento, levante un server FTP desde ese dispositivo y con otro, me loguee e hice **get** a un PDF que tenía allí. Desde el dispositivo que hosteaba al server filtré lo que el Wireshark había analizado con:

ip.addr==X.X.X.X && ftp

En realidad **la dirección IP no es obligatoria** pero me ayuda a tener lo que yo quiero y dejar de lado la información basura, y **"ftp"** sirve para lograr **separar el protocolo**.

```
ip.addr==192.168.0.10 && ftp
```

Source	Destination	Protocol	Length	Info
192.168.0.10	192.168.0.4	FTP	197	Response: 220-FileZilla Server 0.9.57 beta
192.168.0.10	192.168.0.4	FTP	118	Response: 202 USER mode is always enabled. No need to send this command.
192.168.0.10	192.168.0.4	FTP	87	Response: 331 Password required for roadd
192.168.0.10	192.168.0.4	FTP	69	Response: 230 Logged on
192.168.0.10	192.168.0.4	FTP	83	Response: 200 Port command successful
192.168.0.10	192.168.0.4	FTP	127	Response: 150 Opening data channel for file download from server of "/prueba.pdf"
192.168.0.10	192.168.0.4	FTP	98	Response: 226 Successfully transferred /prueba.pdf
192.168.0.10	192.168.0.4	FTP	67	Response: 221 Goodbye

Entonces, mi comunicación es esa se supone. Lindo, muy lindo. Podemos ver en uno de esos paquetes cual es el **usuario** con el cual se conectó (**roadd**) y el **archivo que pidió**.

“Eso podría ser interesante para hacer un ataque de fuerza bruta y ya tener el nombre de usuario correcto.”

Exacto. Pero hay algo mas **interesante** aún. Cambiemos **“ftp”** en el filtro por **“ftp-data”**.

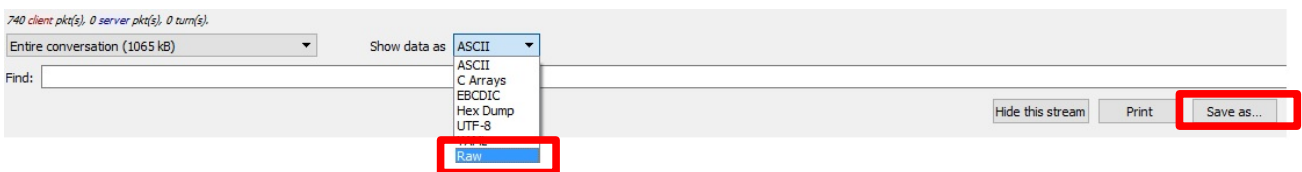
```
ip.addr==192.168.0.10 && ftp-data
```

Source	Destination	Protocol	Length	Info
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes
192.168.0.10	192.168.0.4	FTP-DATA	1514	FTP Data: 1460 bytes

Wow. Un montón de paquetes iguales... **¿o no tan iguales?** Hagamos un **follow al tcp stream**. Nos va a saltar la **ventana emergente**.

```
Wireshark · Follow TCP Stream (tcp.stream eq 7) · wireshark_pcapng_836546
%PDF-1.4
%....
1 0 obj
<<
/BleedBox [ 54 54 585 720 ]
/CropBox [ 54 54 585 720 ]
/MediaBox [ 0 0 639 774 ]
/Parent 2918 0 R
/Resources <<
/ColorSpace <<
/Cs9 2060 0 R
>>
/ExtGState <<
/GS2 2 0 R
/QQAPGS8f98388c 3712 0 R
>>
/Font <<
/QQAPF2a2f501d 3708 0 R
>>
>>
/Rotate 0
/StructParents 624
/TrimBox [ 54 54 585 720 ]
/Type /Page
/Contents [ 3715 0 R 3 0 R 3714 0 R 3716 0 R 3717 0 R ]
/Annots [ 3718 0 R ]
>>
endobj
```

De repente, **el texto no es ilegible sino que es relativamente fácil de leer**. Entonces ¿Qué es esto? Si lo vemos bien, arriba de todo tenemos **la cabecera** de lo que se supone que sería un **archivo PDF**. Primero cambiemos la forma que se ven los datos a **“Raw”** y luego un **“Save as...”** para guardarlo en algun lado.

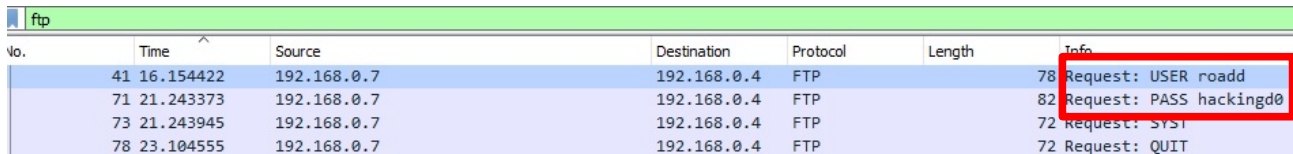


Obviamente lo guardé con **extensión PDF**, y vemos:



El archivo que tomé desde el cliente, es posible de reconstruirse mediante la toma de datos con el analizador de tráfico. Lindo, muy lindo. Con eso, cualquier cosa que se quiera compartir es vulnerado por este sistema.

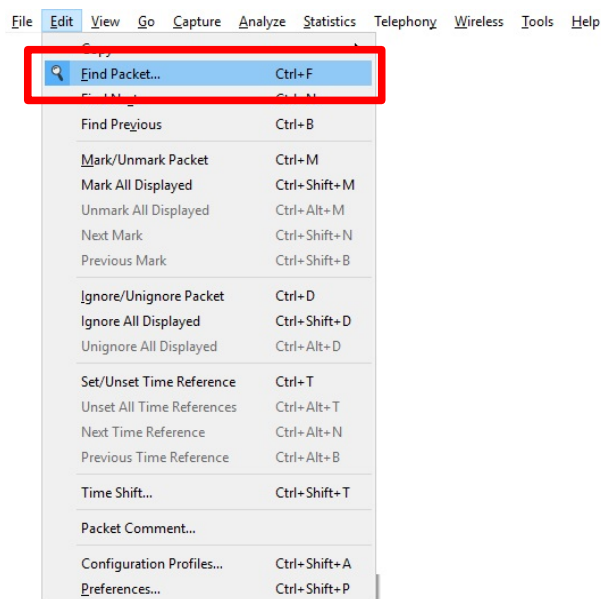
Esto es porque lo vimos desde el lado del cliente, pero si ponemos el wireshark desde el **lado servidor** podremos ver esto:



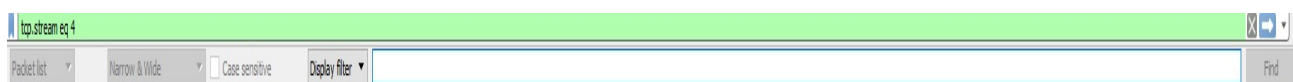
No.	Time	Source	Destination	Protocol	Length	Info
41	16.154422	192.168.0.7	192.168.0.4	FTP	78	Request: USER roadd
71	21.243373	192.168.0.7	192.168.0.4	FTP	82	Request: PASS hackingd0
73	21.243945	192.168.0.7	192.168.0.4	FTP	72	Request: SYST
78	23.104555	192.168.0.7	192.168.0.4	FTP	72	Request: QUIT

Siempre usé el mismo dispositivo desde donde tenía Wireshark para hacer la prueba. Bueno si ven del lado derecho podrán ver que **¡están el usuario y el password en texto plano!** Claro, no está aplicado **ningún tipo de cifrado** y eso es lo que obtenemos.

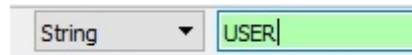
¿Qué pasaría si son muchos paquetes y es difícil o molesto encontrar esas líneas? Simplemente arriba a la izquierda, en el menú, abrimos **Edit → Find Packet.**



Nos abrirá una nueva barra debajo de los filtros.



No se lee muy bien pero en donde dice **Display filter** debemos elegir **String** y usar el **string USER** o **PASS**. Si le damos **enter** o **Find**, nos brindará en la segunda sección cuál es el paquete con esa característica.



Claro, esto nos sirve porque ya sabemos de qué manera se envían los datos en el FTP. Quizás para otros servicios no sería exactamente con ese string.

En fin, eso es más o menos lo básico que deben saber de Wireshark, pero no se olviden de jugar con los filtros e intentar capturar y analizar el tráfico de ustedes en su pc. Si lo hacen, intenten no dejarlo demasiado tiempo porque cuántos mas datos tiene, más lento es el análisis a la hora de filtrar.

Saludos, futuros hackers! :D

Pueden seguirme en Twitter: @RoaddHDC

Contactarse por cualquier duda a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:

1HqpPJbbWJ9H2hAZTmPXnVuoLKkP7RFSvw

También recomiendo que se unan al foro: underc0de.org/foro

Este tutorial puede ser copiado y/o compartido en cualquier medio siempre aclarando que es de mi autoría y de mis propios conocimientos.

Roadd.