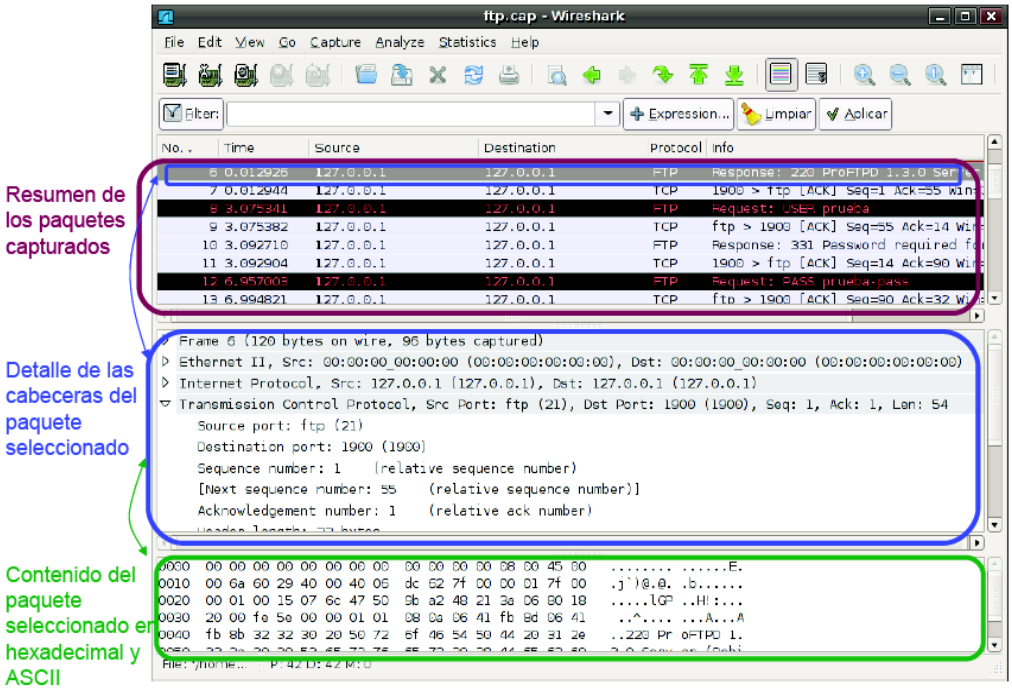
# L'analitzador de protocols Wireshark (I)

## Introducció

Wireshark (https://www.wireshark.org) és un capturador ("sniffer") i analitzador de paquets. Amb aquesta eina podrem "veure" com funcionen a la realitat la majoria de protocols (de tots els nivells) estudiats sobre el paper. Per exemple: podrem reconèixer establiments i finalitzacions de connexions TCP, peticions i respostes ARP, peticions i respostes DNS, missatges ICMP...i en general, qualsevol cosa que "passi pel cable". A més, de cada paquet capturat podrem estudiar el seu contingut, còmodament presentat en diferents seccions: el Wireshark ens mostra tots els valors dels diferents camps de la capçalera Ethernet del paquet (o el protocol físic que sigui) -com per exemple les direccions MAC d'origen i de destí-, a més de tots els valors dels camps de la capçalera IP -com per exemple les direccions IP d'origen i de destí- , a més de tots els valors dels camps de la capçalera TCP -com per exemple, els flags SYN, RST, FIN, etc o els ports d'origen o de destí- ó UDP -aquí només els ports d'origen i de destí- a més de, finalment, tots els camps de la capçalera del protocol d'aplicació concret utilitzat, si es pot reconèixer. Per tant, Wireshark és una eina ideal per saber exactament el tipus i contingut dels paquets que els nostres programes (o els dels veïns) envien i reben quan es comuniquen per la xarxa i, d'aquesta manera, aprendre'n molt més sobre el procés d'encapsulament de la informació, els protocols d'Internet, i en general, del funcionament de les xarxes. A més, ens pot servir per estadístiques del seu rendiment i detectar possibles problemes.

A continuació es mostra una captura d'una finestra típica de Wireshark, on es poden distingir clarament tres seccions diferenciades: la primera mostra la seqüència de paquets rebuts i enviats per la nostra màquina (on apareix un breu resum de les seves característiques més rellevants, com ara la data i hora del seu enviament/recepció, la IP d'origen i de destí i alguna dada més que dependrà del tipus de protocol i paquet detectat (nº de ports involucrats, nº de seqüència de la connexió, etc); la segona mostra, pel paquet que haguem sel.leccionat, a més d'informació general del paquet (tamany total, etc), els detalls de les seves capçaleres convenientment separada (física, IP, TCP/IP, aplicació) i el seu contingut pròpiament dit; la tercera mostra més específicament el seu contingut, tant en format binari com la seva "traducció" a ASCII.



Com poder començar a capturar

Un detall important que cal saber és que, per a què Wireshark pugui capturar correctament els paquets que surten i que arriben a la nostra màquina, cal executar-ho com a "root". Per evitar això, de totes formes, hi ha una alternativa: afegir l'usuari que executarà Wireshark al grup d'usuaris "wireshark" (amb la comanda *sudo usermod -a -G wireshark nomusuari*, per exemple). D'aquesta manera es podrà fer servir el programa sense problemes sense haver de ser "root".

Quan Wireshark s'inicia i l'usuari sel.lecciona de la llista la tarja de xarxa que vol fer servir per capturar, el programa posa automàticament aquesta tarjeta en "mode promiscu" (també es podria fer manualment amb la comanda *ip link set promisc on dev enp0s3*, però gairebé mai serà necessari). Una tarjeta en mode "normal" només tracta les trames en què coincideix l'adreça MAC de destinació amb la seva; en el mode "promiscu", en canvi, tracta totes les trames detectades de la xarxa encara que l'adreça MAC de destinació no sigui la seva. D'aquesta manera, es pot observar molt més tràfic (de fet, tot el tràfic que arriba a la tarjeta encara que no vagi dirigit específicament a ella); el preu a pagar, però, és que es consumeix més CPU i RAM (ja que cal processar i guardar totes les trames que es reben).

De totes formes, si l'ordinador que executa Wireshark està connectat a la xarxa mitjançant un switch (el que se'n diu xarxa commutada), tot i tenir la tarja de xarxa en mode promiscu, no serà capaç de "veure" més que el tràfic enviat a/per ell i el tràfic broadcast o multicast (com l'ARP, per exemple). ¿Per què? Perquè els switchos mantenen una taula a la seva memòria que serveix per associar les direccions MAC de cada màquina connectada a ells amb la boca concreta on estan connectats, de forma que el tràfic vagi directament a la boca que li toca sense "molestar" a la resta. Hi ha diferents maneres de solventar aquest inconvenient, però cap és senzilla i/o neta:

*__*Substituir el switch per un hub*__: Un hub, al contrari que un switch, reenvia el tràfic que li arriba per una boca a totes les demés. D'aquesta manera, ja tindrem sol.lucionat el problema. No obstant, l'ús de hubs actualment està totalment desaconsellat perquè fan saturar la xarxa ràpidament degut a la quantitat de tràfic redundant que no para de viatjar per tots els cables (i, a sobre, l'ample de banda s'ha de compartir).

*__*Fer un enverinament ARP*__: És a dir, fer creure al switch que la direcció MAC de destí de tots els paquets que l'atravessin (o al menys la dels dirigits a un dispositiu en concret) és la de la nostra màquina Wireshark (https://en.wikipedia.org/wiki/ARP_spoofing). Si en aquesta màquina disposem de l'eina adequada (per exemple, Bettercap), aquest enverinament hauria de passar desapercebut pels destinataris vertaders perquè rebrien igualment el missatge després de passar pel Wireshark. No obstant, aquesta solució és molt desaconsellable perquè es pot desestabilitzar la xarxa i crear majors problemes

*__*Fer una inundació de MACs ("MAC flooding")*__: És a dir, saturar el switch generant moltes respostes ARP aleatòries (que en realitat no contesten a cap petició ARP prèvia) amb l'objectiu d'omplir la taula ARP del switch (lloc on s'emmagatzema el mapeig de direccions MACs<->boca física). En aquesta situació, alguns switchos entren en mode "failopen" i passen a comportar-se com a hubs. Bettercap és un programa que es pot fer servir per a aquest tipus "d'atacs" No obstant, a l'igual que l'enverinament ARP, aquesta solució és molt desaconsellable perquè es pot desestabilitzar la xarxa i crear majors problemes.

*__*Ubicar-se en una zona centralitzada de la xarxa*__: Per exemple, es podria executar el Wireshark directament a la màquina que estigui funcionant com gateway de la nostra xarxa o firewall de sortida a Internet. Aquesta solució és molt més elegant però a vegades difícil d'implementar a la pràctica

*__Ubicar-se entre el switch i la "víctima"__: Una altra solució similar a l'anterior però no tan general seria afegir a la nostra màquina (la que executarà el Wireshark) una segona tarjeta de xarxa, connectar una al switch i l'altra a la màquina "víctima" a analitzar. Es necessita llavors que la nostra màquina tingui les tarjetes prèviament configurades en el mode "bridge" (https://wiki.archlinux.org/index.php/Network_bridge)

*__Activar el "Port-Mirroring" en el switch__: La majoria de switchs administrables tenen una característica anomenada "Port Mirroring" o també SPAN ("Switch Port Analysis") que, si s'activa, copia el tràfic entre dues (o més) boques -fins i tot una VLAN sencera- a una altra (on tindrem conectat el nostre Wireshark). A l'igual que la sol.lució anterior, doncs, només serviria per "espia" una víctima concreta, no tota la xarxa. La versió "per a pobres" del Port-Mirroring (en el cas de què tinguem un switch no administrable o que no tingui aquesta opció) és connectar un hub a una boca del switch i llavors connectar al hub tant la "víctima" a espiar com la nostra màquina amb el Wireshark funcionant.

*__Fer servir un dispositiu hardware específic anomenat "Network TAP"__, com per exemple https://www.profitap.com/profishark-1g) o https://greatscottgadgets.com/throwingstar , entre molts d'altres. Per més informació, veieu https://en.wikipedia.org/wiki/Network_tap o https://www.gigamon.com/products/taps-aggregators/network-taps.html.

Per més informació, podeu consultar https://wiki.wireshark.org/CaptureSetup/Ethernet

## Filtres (de captura i de pantalla)

A la xarxa viatjen moltíssims paquets. A la pràctica, si volem treballar eficientment amb el Wireshark, serà necessari realitzar un filtratge de paquets per a què només ens mostri/guardi aquells en els quals estem realment interessats, ignorant la resta de "soroll".

Podem utilitzar dos tipus de filtres, no excloents entre sí: els filtres de captura ("capture filters") i els filtres de pantalla ("display filters"). Si usem algun dels primers, quan arriba un paquet al Wireshark, aquest comprova si s'ajusta o no als criteris establerts pel filtres; si sí, el paquet és acceptat i mostrat a pantalla, si no, el paquet es descarta completament. Si usem algun dels segons, el paquet sempre és capturat sense restricció però el filtre serveix per mostrar a la pantalla només aquells que s'ajustin als criteris del filtre activat en aquell moment.

Els filtres de pantalla es poden especificar en qualsevol moment dins d'una caixa de text que apareix sempre just sota la barra de botons. Els filtres de captura es poden especificar només en el moment de sel.leccionar la tarja de xarxa a utilitzar per començar una nova captura (això pot ser a la pantalla inicial del Wireshark o bé en el quadre que apareix en clicar sobre *Capture->Options...*).

Per desactivar els filtres de pantalla podem pulsar el botó "x" que apareix a la dreta de la caixa de texte (o escriure un filtre buit -una línia en blanc- en aquesta caixa de texte). De fet, podem canviar el filtre de pantalla en qualsevol moment mentre dura la captura simplement escrivint el filtre desitjat en la caixa i pulsant "Enter" (o bé el botó amb el dibuix de la fletxeta que apareix a la dreta de la caixa). Els filtres de captura, en canvi, no es poden desactivar a no ser que aturem la captura i reiniciem una de nova sense cap filtre de captura assignat.

NOTA: Tant els filtres de pantalla com els de captura permeten la creació d'"àlies" per tal d'indicar més ràpida o còmodament filtres sofisticats. Per això cal anar, respectivament, al quadre *Analyze->Display filters* (per crear filtres de pantalla) o al quadre *Capture->Capture filters* (per crear filtres de captura). En qualsevol d'aquests quadres es pot observar que, de fet, ja existeixen uns quants "àlies" ja creats per defecte. En qualsevol cas, per utilitzar els "àlies" definits només cal clicar, respectivament sobre la icona de la bandereta que apareix

a l'esquerra de la caixa de text on s'escriuen els filtres de pantalla o de la caixa de text on s'escriuen els filtres de captura (o bé a la finestra inicial del Wireshark o a la finestra que apareix en fer *Capture->Options*...). En el cas concret dels filtres de pantalla, també hi ha l'opció de *Analyze-> Display filter macros* , la qual permet escriure "àlies" de construccions més sofisticades (per més informació, https://www.wireshark.org/docs/wsug_html_chunked/ChDisplayFilterMacrosSection.html ).

La sintaxis dels dos tipus de filtre és lleugerament diferent però tenim la possibilitat d'usar un assistent per especificar-los. A continuació es mostren alguns exemples de filtres de captura (els quals segueixen una sintaxis estàndar en altres programes anomenada "BPF", de "Berkeley Packet Filter") però la referència completa de tots els filtres de captura possibles es pot trobar a *man pcap-filter*

NOTA: Per tal de filtrar paquets més ràpid, el que fa Wireshark quan establim un filtre BPF (és a dir, de captura) en realitat és generar un codi "Assembly" equivalent a aquest filtre (a través de determinades crides al sistema com ara *pcap_compile()*) i "injectar-lo" (a través d'altres crides al sistema com ara *pcap_setfilter()*) dins del propi kernel, de manera que el paquet a inspeccionar no hagi de passar del kernel (on es detecta) al propi Wireshark (que és un programa funcionant en l'espai d'usuari, amb la latència i el processament que això suposa) sinó que sigui el propi kernel qui el filtri directament. Aquest "Assembly" no és nadiu de cap sistema/arquitectura sinó que es correspon a una "màquina virtual" que tots els kernels Linux moderns incorporen al seu interior. D'aquesta manera, tots els filtres BPF es "traduïran" d'igual manera a un mateix "Assembly" estigui funcionant el kernel allà on sigui.

| **Filtres de captura** | |
|---|---|
| **ether [src\|dst] host *dir_MAC*** | Filtra paquets per direcció MAC. Opcionalment, entre la paraula *ether* i la paraula *host* pot anar la paraula *src* o *dst* per indicar que només es vol tenir en compte la direcció MAC si aquesta dada correspon a la de l'origen o a la del destí, respectivament (si no s'indica res, se sel.leccionaran els paquets on la direcció MAC aparegui indistintament com origen o com a destí). |
| **[src\|dst] host *dir_IP***<br>**[src\|dst] host *nomHost*** | Filtra paquets per direcció IP o nom de host. Opcionalment la paraula *host* pot anar precedida de *src* o *dst* per indicar que només es vol tenir en compte la direcció IP/nom si aquesta dada correspon a la de l'origen o a la del destí, respectivament (si no s'indica res, se sel.leccionaran els paquets on la direcció IP/nom aparegui indistintament com origen o com a destí) |
| **[src\|dst] net *dir_IP_xarxa* mask *mascara***<br>**[src\|dst] net *dir_IP_xarxa/bits*** | Filtra paquets per direcció IP **de xarxa**. La màscara es pot indicar amb el mode "clàssic" o amb notació CIDR. Opcionalment la paraula *net* pot anar precedida de *src* o *dst* per indicar que només es vol tenir en compte la direcció IP si aquesta dada correspon a la de l'origen o a la del destí, respectivament (si no s'indica res, se sel.leccionaran els paquets on la direcció IP aparegui indistintament com origen o com a destí). |
| **{ether\|ip} broadcast** | Filtra els paquets la direcció MAC (*ether*) o la direcció IP (*ip*) dels quals sigui broadcast. |
| **vlan [*nº*]** | Filtra els paquets que continguin una capçalera IEEE802.1Q (o dit d'una altra manera, que pertanyin a alguna VLAN). Si s'especifica un nº, llavors es filtraran només els paquets que pertanyin a la VLAN concreta que tingui com a nº identificatiu l'indicat. |

| | |
|---|---|
| **[tcp\|udp] [src\|dst] port** ***nºport***<br>**[tcp\|udp] [src\|dst] portrange** ***nºp1-nºp2*** | Filtra paquets pel número de port TCP o UDP (o per un rang de ports). Opcionalment la paraula *port* pot anar precedida de *src* o *dst* per indicar que només es vol tenir en compte el número de port si aquesta dada correspon al de l'origen o al de destí, respectivament (si no s'indica res, se sel.leccionaran els paquets on el número de port aparegui indistintament com origen o com a destí). D'altra banda, si no s'indica el protocol (*tcp* o *udp*) se sel.leccionaran els paquets d'ambdós. |
| **{less\|greater}** ***longitud*** | Filtra els paquets que tinguin una longitud en bytes menor o major, respectivament, que la indicada. |
| **arp**<br>**ip**<br>**icmp**<br>**udp**<br>**tcp** | Filtra els paquets que continguin una capçalera pertanyent al protocol especificat (en aquest sentit, els filtres *icmp*, *udp* i *tcp* serien casos concrets del filtre *ip*, més general). Cal tenir en compte que no existeixen filtres de captura per protocols del nivell d'aplicació, havent-se d'utilitzar llavors el protocol del nivell de transport i el número de port adient per tal de capturar el tràfic corresponent a un protocol d'aplicació concret. |
| Tots aquests filtres es poden combinar per formar filtres més complexes amb els operadors lògic ***and***, ***or*** i ***not*** (i fent ús dels parèntesis si calgués agrupar-los). Per exemple, un filtre com "*not arp and not dst port 53*" agafaria tots els paquets que no fossin ni ARP ni peticions DNS; un filtre com "*(udp or icmp) and dst host 158.42.148.3*" agafaria tots els paquets que fossin UDP o ICMP i que a més anessin dirigits a la direcció IP indicada. | |

Ja hem comentat que els filtres de pantalla són molt més flexibles, versàtils i complets que els filtres de captura. Entre altres coses, per exemple, no cal reiniciar la captura per canviar de filtre de pantalla (es poden anar canviant "on the fly") i permeten inspeccionar multitud de protocols a un nivell de detall molt petit, incloent la majoria de protocols de nivell d'aplicació. Es pot trobar un resum de la seva sintaxi a *man wireshark-filter* però la referència completa de tots els filtres de pantalla possibles es pot trobar https://www.wireshark.org/docs/dfref; en tot cas, alguns dels exemples més comuns són:

| **Filtres de pantalla** |
|---|
| A l'hora de realitzar comparacions (és a dir, per filtrar "el valor de tal camp de la capçalera qual quan sigui igual, major, menor ...que tal número"), els filtres de pantalla ofereixen els mateixos operadors de comparació que els del llenguatge C: **==** (similar a **any_eq** en cas que el filtre afecti a diversos camps del mateix paquet, com podrien ser el filtres -estudiats a continuació- *eth.addr*, *ip.addr*, *tcp.port*, etc), **!=** (similar a **all_ne** en cas que el filtre afecti a diversos camps del mateix paquet)**,** **===** (similar a **all_eq** en cas que el filtre afecti a diversos camps del mateix paquet), **!==** (similar a **any_ne** en cas que el filtre afecti a diversos camps del mateix paquet), **>, <,** >= i <=. Igualment, podem fer servir, en comptes del operadors lògics ***and, or*** o ***not*** (que també), els operadors equivalents al llenguatge C: **&&, \|\|** i **!**, respectivament (també existeix l'operador *xor* o la seva equivalència ^^) |
| *Per filtrar per direccions MAC tenim els filtres **eth.src** (d'origen), **eth.dst** (de destí) o **eth.addr** (indistintament), entre altres (de fet, es pot filtrar per qualsevol camp de la capçalera Ethernet: **eth.type**, etc).. Per exemple: *eth.src == 12:34:56:78:90:ab* També existeix el filtre **vlan.id** (o **vlan**) |
| *Per filtrar per direccions IP tenim els filtres **ip.src** (d'origen), **ip.dst** (de destí) o **ip.addr** (indistintament), entre altres (de fet, es pot filtrar per qualsevol camp de la capçalera IP: **ip.ttl** , etc). Per exemple: *ip.src != 10.1.2.3 or ip.dst != 10.4.5.6* |

*Per filtrar per número de port tenim els filtres **{udp|tcp}.srcport** (d'origen), **{udp|tcp}.dstport** (de destí) o **{udp|tcp}.port** (indistintament), entre altres (de fet, es pot filtrar per qualsevol camp de les capçaleres UDP/TCP). Per exemple, un filtre interessant seria, per exemple, el que comprova els valors (0 ó 1) d'un flag en concret, com ara **tcp.flags.syn** o **tcp.flags.reset** ,etc. També es pot indicar simplement si es volen filtrar paquets d'un tipus o un altre (**udp** o **tcp**).

*Els paquets poden filtrar per molts altres protocols. Un exemple seria **arp** o també **icmp** (incloent per valors concrets de les seves capçaleres, com per exemple **icmp.code**, etc) . També es poden indicar protocols de la capa d'aplicació, com **dhcp** o **dns** o **ssh** o **imap** o **mysql** , etc, etc. En concret, ens interessarà sobre tot el protocol **http**, el qual té una sèrie de camps interessants, com:

**http.request.method**==*"GET"*
**http.request.user_agent** *contains "Mozilla"*
**http.request.uri** *matches "[0-9]\.swf$"*
**http.response.code** == *200*
**http.content_type**==*"text/html"*
**http.server**==*"Apache"*

Per saber la llista completa de protocols que Wireshark és capaç d'entendre, podeu consultar **https://wiki.wireshark.org/ProtocolReference**

*Convé destacar també el filtre "frame" , el qual serveix per comprovar característiques pròpies del paquet com a tal, com ara el seu tamany total, la seva posició dins dels paquets capturats, el temps en el què s'ha detectat el paquet, etc. Per exemple (i respectivament): **frame.len** > *10000* , **frame.number** >= *40 o* **frame.time** > *"Sep 13, 2017 11:56:25"*

Com es pot veure en un quadre anterior, un altre operador molt útil és **contains**, el qual permet filtrar paquets que continguin (al nivell del protocol que s'especifiqui) la cadena exacta indicada (o també l'array de bytes exacte indicat) entre cometes. Si es vol comparar amb expressions regulars (case-insensitive i de tipus PCRE!), l'operador a usar és **matches** (o també ~). Per exemple, *http contains "https://www.wireshark.org"* mostra els paquets que continguin en la seva capçalera HTTP o en el seu payload la cadena indicada); en canvi *http matches "wireshark\.(org|com|net)"* mostra els paquets que continguin en la seva capçalera HTTP o en el seu payload les cadenes "wireshark.org", "wireshark.com" o "wireshark.net".

NOTA: Per forçar a què les expressions regulars siguin "case-sensitive", cal precedir l'expressió en qüestió amb els símbols *(?-i)*, així per exemple: *wsp.user_agent matches "(?-i)cldc"*. Per obtenir més informació sobre aquest i altres símbols possibles que es poden indicar, consulteu http://perldoc.perl.org/perlre.html

D´altra banda, també disposem de la notació *nom.camp#nº* per indicar, si el camp en qüestió aparegués més d'un cop en el paquet (és el cas, per exemple, del camp *ip.src* en un paquet tunnel·lat via GREP o una VPN), contra quina ocurrència d'aquest camp es vol contrastar el filtre. Per exemple, indicar *ip.src**#1*** implica només contrastar la primera IP d'origen que aparegui dins del paquet (començant des del seu començament, és a dir, a la capçalera més exterior), i així. Si no s'indica cap número, es contrastarà amb totes les ocurrències del camp existents.

D'altra banda, fer notar que es poden indicar llistes de valors i rangs amb la notació **{ valor1 valor2 }** i **{ valor1 .. valor2 }** , respectivament. Per exemple, *tcp.port in { 80 443 8080 }* mostrarà els paquets que tinguin com a port d'origen o de destí alguns dels tres de la llista; en canvi *tcp.port in { 80 .. 443 8080 }* mostrarà els paquets que tinguin com a port d'origen o de destí qualsevol entre el nº80 i el nº443 (ambdós inclosos) a més del 8080. Aquesta notació també es pot fer servir per cadenes (*http.request.method in {"HEAD" "GET"}* ) , Ips (*ip.addr in {10.0.0.5 .. 10.0.0.9 192.168.1.1 .. 192.168.1.9}* ) o temps ( *frame.time_delta in {10 .. 10.5}*)

Finalment, cal saber que en un filtre de pantalla es poden realitzar operacions aritmètiques bàsiques (+,-,*,/,%) amb la notació *{ nom.camp op valor }*. Per exemple *{tcp.port % 1000 } == 443*

Wireshark també ofereix un conjunt de funcions que es poden utilitzar dins els filtres de pantalla, com:
**upper():** converteix a mayúscules el valor indicat entre parèntesis (normalment serà el d'un camp de capçalera)
**lower():** converteix a minúscules el valor indicat entre parèntesis (normalment serà el d'un camp de capçalera)
**len():** retorna la longitud (en bytes) del valor indicat entre parèntesis (normalment serà el d'un camp de capçalera)
Per exemple, *lower(http.server) contains "apache"* buscarà en els paquets HTTP la cadena "apache" només en mínúscules; *len(http.request.uri) > 100* buscarà tots els paquets HTTP de peticions llargues

Filtres de pantalla a nivell de bytes

Amb el filtre ***protocol[x:y] == valor*** podem "afinar" una mica més i mirar només un byte en concret (o conjunt de bytes) pertanyent/s a algun camp de la capçalera d'un determinat protocol. El valor "x" indica el byte a partir del qual es mirarà (comença per 0) i el valor "y" és la quantitat de bytes que es miraran (per defecte val 1). El protocol pot ser: *ether, arp, ip, icmp, udp, tcp...*

NOTA:Per usar aquesta notació, haurem de conèixer, no obstant, la mida i l'ordre dels diferents camps de les capçaleres més importants. Dins del Wireshark saber això és fàcil si mostrem la secció "Packet Diagram" (per exemple al "panell 3", en lloc de la secció "Packet Bytes") anant el menú *Edit → Preferences → Appearance → Layout* En tot cas, a continuació mostrem enllaços a la Wikipedia per tenir aquesta informació més a l'abast:
https://en.wikipedia.org/wiki/Ethernet_frame#Structure : Capçalera Ethernet II / 802.3
https://en.wikipedia.org/wiki/Address_Resolution_Protocol#Packet_structure : Capçalera ARP
https://es.wikipedia.org/wiki/Cabecera_IP : Capçalera IP
https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#ICMP_datagram_structure
https://en.wikipedia.org/wiki/User_Datagram_Protocol#Packet_structure : Capçalera UDP
https://es.wikipedia.org/wiki/Segmento_TCP#Formato_del_segmento_TCP : Capçalera TCP

*Per exemple, si volem veure els paquets ICMP de tipus "echo request" (és a dir, els que tenen el camp "tipus" -primer byte de la capçalera- igual al valor 8) en comptes d'utilitzar el filtre predefinit *icmp.type == 8* podríem escriure de forma equivalent el següent: *icmp[0:1] == 8* (o *icmp[0] == 8* ).

*Un altre exemple: el filtre *ip[9:1] == 1* (o *ip[9] == 1*) filtra tots els paquets ICMP. ¿Per què? Perquè el byte nº9 de la capçalera d'un paquet IP indica el protocol "superior" que està encapsulat dins del paquet IP mitjançant un codi numèric que pot ser 1 (ICMP), 6 (TCP) o 17 (UDP), entre altres.

*Un altre exemple: el filtre *tcp[13:1] == 2* (o *tcp[13] == 2*) filtra tots els paquets TCP amb el flag SYN activat. ¿Per què? Per què els flags d'un paquet TCP (SYN, FIN, RST, etc) es troben al byte nº13 i cadascun es correspon a un bit concret que valdrà 1 si el flag "està activat" i 0 si no. En concret, el flag SYN és el segon començant per la dreta, així que un paquet TCP amb (només) aquest flag activat tindrà en el byte nº13 el valor 00000010 (és a dir, 2 en decimal). És fàcil deduir a partir d'aquí que, per exemple, per detectar un paquet SYN/ACK hauríem d'establir el filtre *tcp[13] == 18* (el flag ACK és el cinquè començant per la dreta, així que el valor del byte seria 00010010, és a dir 16+2=18).

NOTA: Ens podem trobar en situacions que volguem filtrar paquets en base al valor no d'un byte o conjunt de bytes sinó el d'una part d'un byte. Per exemple, el 1r byte de la capçalera IP en realitat està format per 2 camps diferents, cadascun de 4 bits (versió del protocol i mida de la capçalera -mesurat en nº de paraules de 32 bits-, respectivament). Si volguéssim filtrar només per un valor pel 2n camp (suposarem el valor decimal 5), ¿com ho podríem fer?. Doncs amb l'operació AND a nivell de bits fent servir l'operador & per aplicar una màscara binària, els valors de la qual hauran de valer 1 per tots els valors que volguem "conservar" del byte en qüestió i 0 pels que no. En l'exemple indicat, això es faria així: *ip[0] & 0x0f == 5*; si volguéssim en canvi "quedar-nos" només amb els primers 4 bits (per veure si valen, per exemple, 6 en decimal), llavors seria: *ip[0] & 0xf0 == 6*
Teniu més exemples a https://blog.wains.be/2007/2007-10-01-tcpdump-advanced-filters o *man pcap-filter*

## EXERCICIS

A la màquina real de l'aula ja tens instal·lat el Wireshark, així que per realitzar els exercicis podràs fer servir aquest programa sense haver d'arrencar cap màquina virtual

**1.-a)** Obre el Wireshark a la màquina real, tria del menú de dispositius de captura la (única) tarja de xarxa real de la màquina (no pas el dispositiu "any", mira la segona nota següent!) i comença a capturar. Tot seguit aplica-hi el filtre de pantalla *arp || icmp* i obre un terminal per fer "ping" durant uns segons a algun ordinador de la teva xarxa local al qual no hagis contactat com a mínim fa deu minuts ; finalment, atura la captura. ¿Quin tipus de paquets veus al Wireshark?

> NOTA: El requisit de no haver contactat en un temps prudencial és perquè es vol forçar a que la teva màquina, abans d'enviar efectivament el "ping", faci peticions ARP a la xarxa per trobar el destí (i així aconseguir que es vegin al Wireshark). Si ja es va contactar amb el destí fa poc temps, la seva MAC la tindríem guardada a la taula ARP i, per tant, la teva màquina no faria cap petició ARP i passaria directament a fer el "ping" (amb la qual cosa, doncs, no veuríem cap petició ARP prèvia als paquets ICMP). Si no tinguéssim més remei que recontactar amb una màquina, la MAC de la qual estigués present a la nostra taula ARP, per fer l'exercici tal com es demana primer hauríem d'esborrar manualment la taula ARP (recorda que això es pot fer amb la comanda *sudo ip neigh flush dev enp0s3* però, tal com es veu, cal ser "root" per executar-la)
> NOTA: El requisit de no triar el dispositiu "any" és degut a què llavors no podríem accedir en detall a les dades corresponents al nivell 2 (és a dir a les dades Ethernet). Això és perquè el dispositiu "any" és un dispositiu genèric de captura que permet obtenir tràfic de múltiples orígens heterogenis però justament per això, es queda en una captura de dades minimes comunes. Es pot identificar aquest fet observant, al panell central del Wireshark que no apareix la capa "Ethernet" sinó una altra "d'equivalent" (però menys complerta) anomenada "Linux cooked capture v1". Teniu més informació sobre aquest mode de captura genèric a https://wiki.wireshark.org/SLL

**b)** Localitza, d'entre els paquets capturats anteriors, un que sigui de tipus "ARP request" (*arp.opcode == 1*). ¿Quina direcció MAC té com a destí? ¿Quina és, en canvi, la direcció MAC d'origen d'un paquet "ARP reply" (*arp.opcode == 2*)?

**c)** Localitza, d'entre els paquets capturats anteriors, un que sigui de tipus ICMP request. ¿Quin és el valor del camp Type de la capçalera d'un paquet ICMP request? ¿Quin és, en canvi, el valor del camp Type de la capçalera d'un paquet ICMP reply?

**d)** ¿Quin és el valor del camp Type de la capçalera Ethernet per tots els paquets que siguin de tipus ARP (request o reply, és igual)?

**dII)** ¿Quin és, en canvi, el valor del camp Type de la capçalera Ethernet per tots els paquets que estiguin encapsulats dins d'un paquet IP (com són els paquets ICMP, TCP o UDP)?

**e)** Torna a iniciar una nova captura amb el mateix filtre *arp || icmp* (no cal que salvis l'anterior captura) i torna a executar el mateix "ping" d'abans ¿Veus ara els mateixos tipus de paquets?

**f)** ¿Tindria sentit haver aplicat en els apartats anteriors un filtre com *arp && icmp* ?

**g)** Executa un altre cop la comanda ping però canviant el TTL dels paquets (això es fa amb el seu paràmetre -t) per a què sigui 234. ¿Quin hauria de ser el filtre de pantalla que mostri només els paquets amb aquest TTL o major? Prova-ho.

**h)** Torna a iniciar una nova captura però ara amb el filtre *icmp || dns* i executa la comanda *ping www.hola.com* ¿Quants i quins paquets DNS ha enviat i rebut la teva màquina abans d'enviar el primer paquet ICMP?

**i)** Només observant la informació que t'aporta el Wireshark a l'apartat anterior, ¿podries dir quina és l'adreça IP de www.hola.com?

**2.-a)** Torna a iniciar una nova captura però ara amb el filtre *dhcp* i seguidament executa en un terminal de la màquina real les comandes *nmcli con down nomConnexio && nmcli con up nomConnexio*

> NOTA: "nomConnexio" és el nom de la connexió a la qual la tarja de xarxa estigui associada en aquest moment, la qual es pot conéixer prèviament fent *nmcli con show* Les dues comandes anteriors serien equivalents a *sudo dhclient -r nomTarja && sudo dhclient nomTarja* però sense haver de ser "root"

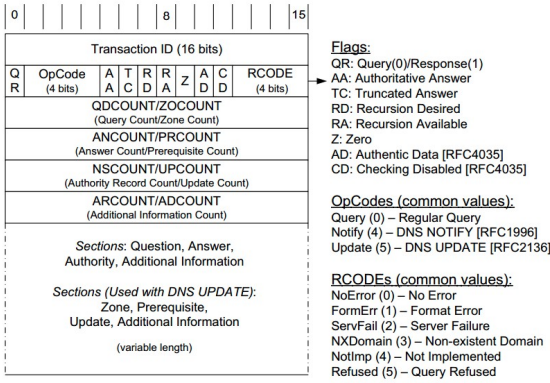
En executar les dues comandes anteriors hauries de veure al Wireshark varis paquets: alguns generats per la teva màquina ("DHCP Release", "DHCP Discover" i "DHCP Request") i altres que són respostes del servidor ("DHCP Offer" i "DHCP ACK"). Després de consultar el PDF de teoria sobre el procés "DORA" (vist quan vam estudiar el nivell nº3 a la UF anterior) per saber el significat de cada paquet (o,l'alternativa https://www.eventhelix.com/RealtimeMantra/Networking/dhcp-flow/dhcp-sequence-diagram.pdf ), digues per què les direccions MAC i IP (tant d'origen com de destí) a cadascun d'aquests 5 paquets són les que són

**b)** Ja sabem que un servidor DHCP dóna molt més que direccions IP. Observa el contingut del paquet "DHCP ACK" rebut i digues quina informació és la que es correspon amb les opcions 1, 3, 6 i 51 de l'estàndard.

**3.-a)** Torna a iniciar una nova captura però ara amb el filtre *dns contains lecturas* i seguidament executa la comanda *ping www.lecturas.com* ¿Què veus?

**aII)** ¿I si escrius el filtre *dns contains lecturas && (dns.flags==0x0100 || dns.flags == 0x0110)* ? ¿És el mateix que si escrius el filtre *dns contains lecturas && dns.flags.response == 0* ?

> NOTA: Tot i que el propi Wireshark ja t'ho mostra, pots consultar l'estructura d'un paquet DNS en aquest gràfic:




**aIII)** ¿I si escrius el filtre *dns contains lecturas && dns.flags==0x8180* ? ¿És el mateix que si escrius el filtre *dns contains lecturas && dns.flags.response == 1* ? ¿Si escrius el filtre *dns.resp.name == www.lecturas.com* veus alguna diferència?

**aIV)** Dedueix quin tipus de paquets mostra el filtre *dns.qry.name == www.lecturas.com* a partir del que et mostra el Wireshark. ¿Què passa si escrius el filtre *dns.qry.name contains lecturas.com* ? ¿I el filtre *dns.qry.name matches ".*lecturas.[(com|es)]"* ?

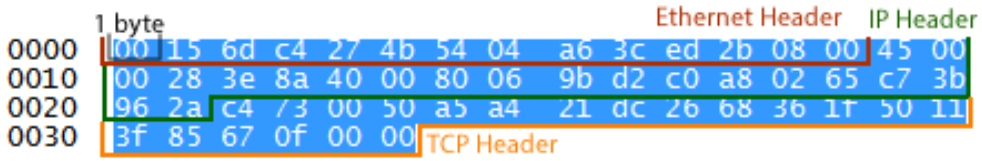
**b)** ¿Quin és el significat del filtre *dns.a == 212.230.153.51*? D'altra banda, ¿apareix alguna cosa si escrius el filtre *dns.aaaa* ? Per què?

**c)** ¿Per què si apliques el filtre *dns.count.answers == 2* es mostra (només) el paquet de resposta DNS però si apliques el filtre *dns.count.answers == 3* no n'apareix cap?

**d)** Ja sabem que el protocol mDNS fa servir per comunicar el nom que autoimposa a la màquina on s'està executant una variant del protocol DNS anomenat mDNS, el qual funciona al port UDP *5353* Digues quin seria un filtre adient per observar aquest tipus de tràfic i aplica'l per comprovar si aquest apareix a la teva aula en aquest moment.

**e)** Sense executar la comanda *dig* (ni cap altra que resolgui noms per terminal; és a dir, només fent servir el Wireshark i un navegador), comprova (i digues com ho has fet) que la direcció IP de les màquines anomenades "www.elmundo.es" i "www.marca.com" són la mateixa (això significa que els dos llocs webs s'allotjen al mateix servidor)

**4.-**Ha arribat a les teves mans la següent foto, la qual mostra el contingut binari d'un paquet. A partir d'ella (i de consultar l'article https://www.hackingarticles.in/network-packet-forensic-using-wireshark) , dedueix les següents informacions:




**a)** Dedueix la direcció MAC de destí (primers 6 bytes), la d'origen (següents 6 bytes) i el protocol de nivell 3 amb què està encapsulat la resta del paquet (últims 2 bytes de la capçalera Ethernet, com a xul.leta pots consultar l'arxiu /etc/ethertypes).

**b)** Dedueix el protocol de nivell 4 amb què estan encapsulades les dades (byte nº10 de la capçalera IP, com a xul.leta pots consultar l'arxiu /etc/protocols però vigila que a la foto els valors estan en hexadecimal)

**c)** Dedueix la direcció IP d'origen i la de destí sabent que són els valors presents als bytes nº13, 14, 15 i 16 i nº 17, 18, 19 i 20 de la capçalera IP , respectivament

**d)** Dedueix el port d'origen i el de destí sabent que són els valors presents al primer i segon byte i al tercer i quart, de la capçalera TCP, respectivament.

**e)** Dedueix les flags d'aquest paquet sabent que es troben dins del byte nº14 de la capçalera TCP.

> PISTA: En aquest cas, en comptes de transformar el valor hexadecimal en decimal (tal com es demanava als apartats anteriors) el que cal fer és transformar-lo en binari per tal de trobar la posició dels bits concrets que valen 1. En aquest sentit, recorda que la posició de les flags és en aquest byte és : CWR-ECN-URG-ACK-PSH-RST-SYN-FIN

> NOTA: Un eina online que fa la tasca de decodificar el contingut binari d'un paquet de xarxa de forma automàtica és http://sadjad.me/phd

**5.-a)** Raona què estaries buscant si fessis servir cadascun dels següents filtres de pantalla i provoca algun tipus de tràfic per a comprovar si el teu raonament és correcte (recorda que la referència dels filtres de pantalla es troba aquí: https://www.wireshark.org/docs/dfref):

| | |
|---|---|
| *frame.len <128*<br>*frame contains un:a:dir:ecc:io:mac* | *tcp.len < 128*<br>*tcp.srcport > 4000*<br>*tcp.seq ==321345*<br>*tcp.flags == 0x0002*<br>*tcp.flags.reset == 1* (seria similar a *tcp.connection.rst*)<br>*tcp.flags.fin == 1* (seria similar a *tcp.connection.fin*)<br>*tcp.payload contains "hola"* |
| *!(arp or icmp or dns)* | *tls.handshake*[1]<br>*tls.handshake.type == 1*[2] |
| *eth.addr[0:3]==08:00:27*<br>*eth.dst=ff:ff:ff:ff:ff:ff* | *http.request*<br>*http.request.uri contains "avi"*<br>*http.request.method == "GET"*<br>*http.response.code > 299 && http.response.code < 400*<br>*http.content_type contains "video"*<br>*http.content_type[0:5] == "video"* |
| *ip.len < 128*<br>*ip contains un:a:dir:ecc:io:mac*<br>*ip.addr == 1.1.1.1*<br>*ip.src == 1.1.1.1* | |
| *udp.len < 128*<br>*udp contains un:a:dir:ecc:io:mac*<br>*udp[2:2] > 4000* o *udp.port > 4000*<br>*udp.dstport > 4000* | |

[1]Paquets pertanyents a l'establiment del canal TLS segur, previ a la transmissió d'informació xifrada

[2]Concretament, el paquet "Client Hello" (el que inicia el TLS-handshake). Altres valors interessants a buscar pel filtre *tls.handshake.type* són 2 (la resposta "Server Hello" a l'inici del TLS-handshake demanat pel client), 4 (entrega d'un nou ticket de sessió TLS per part del servidor al client), 11 (entrega del certificat del servidor al client), 13 (demanda per part del servidor d'un certificat de client, en el cas que aquest hagi de validar la seva identitat) o 14 (finalització del TLS-handshake mitjançant el missatge "Server Hello Done"), entre altres

**b)** ¿Com podria ser un filtre de pantalla que només et mostri el tràfic provinent de la IP de la teva màquina i dirigit a la MAC de la teva porta d'enllaç? Si l'apliques, ¿quins paquets veus?

**c)** Aplica el filtre *wol* i seguidament utilitza algun programa d'enviament de "paquets màgics" (com *etherwake*, *awake*, *wakeonlan*,...) per veure aquest tipus de paquets al Wireshark. ¿Quin és el valor del camp Type de la capçalera Ethernet? ¿Quin és el contingut d'un "paquet màgic" (és a dir, les dades incloses dins de qualsevol de les trames Ethernet mostrades)?

**d)** Llegeix aquest article (https://www.cellstream.com/2017/06/24/wireshark-display-filter-macros) i digues què explica

**6.-**Descarrega't el fitxer "Conversa_HTTP_completa.pcap" de la web del centre i obre'l amb el Wireshark. Aquesta captura representa tota una conversa TCP (incloent el 3-way handshake i la seqüència de desconexió també) dins de la qual es realitza una petició HTTP (GET) per tal de descarregar una imatge. A partir d'aquí:

**a)** ¿Quins són els paquets que formen el 3-way handshake? ¿Com els has reconegut?

**b)** Sabent que "174-143-213-184.static.cloud-ips.com" és el nom del "hosting" que allotja múltiples llocs web, ¿en quin lloc de la petició GET s'indica el lloc web concret al qual va dirigida? Pista: consula les capçaleres de la petició HTTP al panel horitzontal intermig del Wireshark (o, com veurem en un apartat posterior, amb l'opció "Analyze->Follow HTTP Stream")

**c)** Aplica el filtre de pantalla *http* i observa si hi ha hagut alguna redirecció (amb resposta del servidor amb codi 301) o si, en canvi, la petició ha sigut directa (amb resposta del servidor amb codi 200).

**d)** Selecciona un paquet HTTP visible qualsevol del panell superior del Wireshark i tot seguit utilitza el menú *Analyze → Follow HTTP Stream* per conèixer les capçaleres i peticions HTTP del client (en vermell) i les capçaleres i respostes HTTP del servidor (en blau). ¿Què representen els símbols "estranys" que es veuen en la resposta del servidor a partir de la línia ".PNG"?

**dII)** Encara a la pantalla de "Follow HTTP Stream", digues quina és la URL sencera de la petició (la URL sencera està formada pel nom DNS indicat a la capçalera client *Host:* seguit del "path" indicat a la petició GET) Si escrius aquesta URL directament en un navegador, ¿què veus?

> NOTA: La URL bona ha canviat; ara és http://packetlife.net/static/img/logo.png

**dIII)** Encara a la pantalla de "Follow HTTP Stream", digues quin navegador va realitzar la petició observant el valor de la capçalera client *User-Agent*

**dIV)** Encara a la pantalla de "Follow HTTP Stream", digues quin programa servidor web va contestar i quan observant el valor de les capçaleres servidor *Server* i *Date*

**dV)** Encara a la pantalla de "Follow HTTP Stream", digues quin és el significat de les capçaleres servidor *Content-Type* i *Content-Length*

**dVI)** Digues quants bytes ha enviat en total el client al servidor i quants ha enviat el servidor al client (aquesta informació es troba disponible al desplegable ubicat a la cantonada inferior esquerra de la pantalla de "Follow HTTP Stream")

**e)** ¿Què mostra el quadre que apareix en sel.leccionar l'opció del menú "Statistics->HTTP->Requests"? ¿I "Statistics->HTTP->Load distribution"? ¿I "Statistics->HTTP->Packet counter"?

**f)** Utilitza el menú "File → Export objects → HTTP" per tal guardar al teu disc dur la imatge descarregada en la conversa HTTP (és l'únic objecte reconegut)

**g)** Ara torna a fer el mateix però seguint aquest passos: aplica el filtre *http.content_type contains "png";* sel.lecciona la capçalera "Portable Network Graphics" de l'únic paquet visible (la qual apareix sota les dades HTTP en el quadre horitzontal del mig) i ves al menú "File → Export Packet Bytes"

> NOTA: Per a què tant l'apartat f) com el g) funcionin cal que l'opció Edit → Preferences → Protocols ->TCP → "Allow subdissector to reassemble TCP streams." estigui activada (cosa que per defecte ja ho està)
> NOTA: La capacitat del Wireshark de reconéixer fitxers fins del tràfic capturat prové de la seva capacitat de reconéixer moltes de les "file signatures" hbituals. En podeu obtenir un llistat de les més comunes a https://www.garykessler.net/library/file_sigs.html i https://en.wikipedia.org/wiki/List_of_file_signatures Si no recordeu què és una "file signature", llegiu els següents paràgrafs:
>
> > *A file signature is a unique identifying number located at the beginning of a file which identifies the type of file, giving information about the data contained within. This information can be used when the file extension has misidentified the file as an incorrect type. File signatures are located at the file header, a block of data at the beginning of a file that defines the parameters of how information is stored in the file, in the form of a sequence of bytes which defines if actual file is an image file, a document from a specific program, etc, etc. The file header does not use a defined standard, though; it, instead, is proprietary to each different format, meaning a program or operating system needs a file signature database to determine the type of an unknown file. Anyway, two file signatures shouldn't be the same*

**h)** Ara torna a fer el mateix però seguint aquests passos encara més "artesanals":

*Aïlla la conversa TCP que conté les dades a extreure. Per fer això, selecciona el paquet corresponent a la petició GET i escull l'opció "*Follow TCP Stream*" del menú contextual. Veuràs que Wireshark aplica llavors un filtre de pantalla (del tipus *tcp.stream=nº*) per mostrar només els paquets pertanyents a la conversa seleccionada (en aquest cas només n'hi ha una, la nº0, però en un tràfic normal de xarxa n'hi poden haver moltes amb els paquets intermesclats entre sí) i que apareix una finestra amb el contingut d'aquesta conversa

*En aquesta finestra les dades transmeses estan marcades en vermell i les rebudes en blau. Com que només ens interessen aquestes últimes, cal indicar que només es volen veure aquestes seleccionant l'opció adient del quadre desplegable que hi ha a la cantonada inferior esquerra de la finestra.

*Per extraure totes aquestes dades "blaves", cal assegurar-se de seleccionar el valor "Raw" al desplegable de "Show and save data as" i seguidament fer clic al botó "Save as". El que obtindrem és un fitxer binari que conté tant la resposta i capçaleres HTTP com la imatge pròpiament dita. Per tant...

*...hem d'extreure aquesta imatge de l'interior del fitxer binari, excloent la resta de dades que no ens interessa. Per fer això podem utilitzar la comanda *binwalk* , la qual és capaç de reconèixer estructures binàries individuals (com pot ser la d'una foto, un executable, un arxiu zip, etc) dins d'un fitxer binari "embolcall". La manera de fer-lo servir és simplement:

***binwalk fitxer.raw*** : llista les estructures binàries internes reconegudes

***binwalk -D "part_descripcio:ext" fitxer.raw*** : extreu a la carpeta actual el/s fitxer/s binari/s la descripció del/s qual/s apareix a la columna de més a la dreta en fer el llistat d'estructures existents. Es pot escriure qualsevol part de la descripció a partir del principi i pot ser una expressió regular, però en tot cas ha de en minúscules. L'extensió indicada serà la que es s'afegirà al/s fitxer/s binari/s extret/s

NOTA: Una alternativa a la comanda binwalk -D anterior és fer servir directament la comanda *dd* amb el paràmere *bs=1* , el paràmetre *skip* indicant la posició a partir d'on extreure (aquest número es mostra al llistat oferit per *binwalk* ) i el paràmetre *count* valent la diferència entre la posició del següent element binari que apareix a la llista menys el valor indicat a *skip* . És a dir, així:
*dd if=fitxer.raw of=imatge.png bs=1 skip=68264 count=2760* , per exemple

NOTA: Es poden obtenir molts altres fitxers de captura ja preparats (ideals per tant per poder estudiar protocols que potser no els podem tenir a l'abast a la nostra xarxa o també patrons de tràfic maliciós conegut) de diferents llocs webs, com ara:

https://wiki.wireshark.org/SampleCaptures
http://www.malware-traffic-analysis.net
https://asecuritysite.com/forensics/pcap
https://tshark.dev/search/pcaptable
https://packetlife.net/captures
https://www.pcapr.net
https://www.netresec.com/?page=PcapFiles
http://icir.org/enterprise-tracing/download.html

**7.-a)** Realitza un escaneig de ports entre el 1 i el 100 a un ordinador qualsevol de l'aula utilitzant l'Nmap amb el mètode per defecte (paràmetre *-sT*). Recorda que aquest mètode intenta realitzar la conexió TCP estàndar 3-way handshake de manera que si, després d'enviar el paquet SYN, rep una resposta SYN/ACK, dedueix que el port en qüestió està obert (i envia llavors un ACK per acabar d'establir connexió) però si rep una resposta RST/ACK, dedueix que el port està tancat. Sabent això, comprova que els tipus de paquets enviats i rebuts que mostra el Wireshark es corresponen, efectivament, amb els ports oberts (3-way handshake finalitzat) i tancats (3-way handshake interromput per un RST) indicats a la sortida de l'Nmap (hauràs de fer servir algun filtre com ***tcp.flags.ack == 1*** *i/o* ***tcp.flags.syn == 1*** combinat amb ***ip.src == ip.vict.ima*** (per facilitar la inspecció) i/o ***tcp.flags.reset == 1*** combinat també amb ***ip.src == ip.victima***.

**b)** Realitza ara un escaneig també del rang de ports entre 1 i 100 a la mateixa màquina però ara fent servir conexions UDP (paràmetre *-sU*). En aquest cas, en teoria la resposta a un port tancat és un paquet ICMP de tipus "Destination unreachable". Comprova amb el Wireshark que sigui així (fent servir algun filtre com ***icmp.type == 3*** combinat amb ***ip.src == ip.victima***

**c)** Realitza ara un escaneig també del rang de ports entre 1 a 100 a la mateix màquina però utilitzant ara el mètode "Null Scan" (paràmetre *-sN*), el qual consisteix en enviar paquets TCP sense cap flag activat. Si el port investigat està tancat la víctima hauria de retornar un paquet RST i si està obert no hauria de tornar res (això sempre i quan la màquina destí apliqui l'estàndar TCP, que no sempre passa). Sabent això, comprova que els paquets enviats per Nmap als diferents ports no tinguin, efectivament, cap flag activada (fent servir per exemple el filtre ***tcp.flags == 0x0***) mentre que els paquets rebuts com a resposta mostrats al Wireshark (és a dir, els visibles pel filtre ***tcp.flags.reset == 1*** combinat amb ***ip.src == ip.victima***) es corresponen amb els ports oberts que indica Nmap

**d)** ¿Quin és el paquet de resposta que es veu en el Wireshark si un determinat port de la màquina víctima (el que tu vulguis) està tancat fent servir el paràmetre *-sS* de Nmap? (per saber-ho, observa el tràfic aparegut en temps real al Wireshark mentre executes la comanda *nmap* adient) ¿I si està obert? En aquest sentit, ¿quina diferència hi ha entre els paràmetres *-sT* i *-sS* en relació a l'intercanvi de paquets en el 3-way handshake?

**e)** ¿Quin és el paquet de resposta que es veu en el Wireshark si un determinat port de la màquina víctima (el que tu vulguis) està tancat fent servir el paràmetre *-sA* de Nmap? (per saber-ho, observa el tràfic aparegut en temps real al Wireshark mentre executes la comanda *nmap* adient) ¿I si està obert? ¿Quin/s "flags" té activats el paquet inicial que envia Nmap en aquest tipus d'escaneig?

Esnifar HTTPS (i altres)

Si intentes capturar els paquets pertanyents a una conversa HTTPS amb el Wireshark veuràs que no obtindràs cap paquet HTTPS sinò tot un seguit de paquets TCP i TLS. Això és perquè la comunicació HTTPS viatja encriptada (gràcies precisament al protocol TLS subjacent, el qual aporta aquesta capa de seguretat al protocol HTTP planer) i, per tant, Wireshark no és capaç d'observar-hi a dins què hi circula. L'única manera d'"esnifar" una comunicació basada en TLS és desencriptar-la, trencant la seva seguretat; i la manera més fàcil de fer-ho és interceptant al vol les claus utilitzades durant la comunicació entre el navegador, el tràfic del qual es vol espiar, i el servidor HTTPS de l'altre extrem. El procediment concret és el següent:

NOTA: Recorda que el protocol SSL/TLS és un protocol genèric que se situa per sobre del protocol TCP i sota la capa d'aplicació. Això vol dir que es pot utilitzar (i de fet, s'utilitza) per afegir seguretat no només al protocol HTTP sinó a molts d'altres protocols d'aplicació, com ara l'FTP (FTPS), el SMTP (SMTPS), etc

**1.-**Cal crear una variable d'entorn del sistema anomenada SSLKEYLOGFILE que tingui com a valor la ruta d'un arxiu qualsevol (no cal que existeixi prèviament i el seu nom pot ser inventat) on el navegador a emprar tingui permisos d'escriptura. Per exemple, executant: *export SSLKEYLOGFILE=~/sslkeys.txt* La idea és que en arrencar el navegador (o la comanda *curl*!) des del mateix terminal on està ja definida aquesta variable, aquest automàticament utilitzarà l'arxiu indicat en aquesta variable (si aquesta existeix) per guardar-hi allà els valors interns usats en la generació de les seves claus de sessió TLS. És per això que és recomanable afegir aquesta comanda dins de l'arxiu ~/.bashrc per tal de què s'executi automàticament sempre que s'obri un terminal.

NOTA: El contingut d'aquest fitxer ve donat pel format detallat aquí:
https://tlswg.org/sslkeylogfile/draft-ietf-tls-keylogfile.html

**2.-**Cal obrir el Wireshark i associar-li la ruta de l'arxiu indicat al pas anterior. Això es fa anant al menú **Edit → Preferences → Protocols → TLS → (Pre)-Master-Secret log filename .** En aquest pas el que estem fent és aconseguir que Wireshark pugui accedir a les claus de sessió emmagatzemades a l'arxiu indicat i, per tant, que pugui resseguir el mateix trànsit de dades que el navegador veu.

**3.-**Cal descarregar una versió del navegador Firefox que reconegui i utilitzi la variable SSLKEYLOGFILE. Antigament qualsevol versió podia però des de fa temps només ho permeten les versions disponibles a https://www.mozilla.org/en-US/firefox/developer , anomenades "Developer Edition". Un cop descarregat i descomprimit el paquet comprimit, repetim que cal executar el binari "firefox" que hi ha a dins en el mateix terminal on s'hagi executat la comanda *export* del primer punt

NOTA: Per saber més sobre el per què d'aquest punt, consulteu la documentació sobre NSS_ALLOW_SSLKEYLOGFILE present a
https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Reference/NSS_environment_variables

NOTA: El procediment anterior funcionarà sempre i quan Wireshark hagi estat compilat amb la llibreria GnuTLS (no pas OpenSSL). Afortunadament, aquesta condició es compleix a la majoria de distribucions importants (es pot comprovar fent *wireshark -v*)
NOTA: Un altre article més complet i tècnic sobre el tema el pots trobar aquí:
http://joji.me/en-us/blog/walkthrough-decrypt-ssl-tls-traffic-https-and-http2-in-wireshark#6e30
NOTA: En lloc de realitzar el pas 3, una alternativa seria utilitzar el mateix navegador de sempre però llavors fer passar tot el seu tràfic per un proxy HTTP que reconegués i utilitzés la variable SSLKEYLOGFILE, com pot ser Mitmproxy (https://mitmproxy.org)
NOTA: Com es pot deduir de tot el que s'ha explicat, desxifrar el tràfic HTTPS amb Wireshark només és possible si es té accés a l'arxiu "sslkeys.txt" explícitament generat per la víctima. Per tant, no és una solució per desxifrar qualsevol tràfic HTTPs indiscriminadament (i afortunadament)

Un cop feta una de les dues accions anteriors, ja podrem veure al Wireshark el tràfic HTTPS com si fos tràfic HTTP tradicional

**8.-a)** Visita diferents llocs webs HTTPS mentre captures el tràfic generat amb el Wireshark. Comprova que, efectivament, aquest no sigui capaç d'entendre aquest tràfic (només hauràs de veure paquets TCP i TLS sense poder saber què hi transporten a dins)

**b)** Segueix els passos indicats a la teoria per tal de poder capturar paquets HTTPS amb el Wireshark

**c)** Torna a visitar diferents llocs webs HTTPS mentre captures el tràfic generat amb el Wireshark. Comprova que, efectivament, ara sí es capaç d'entendre aquest tràfic mostrant com toca el protocol HTTP i amb el contingut i capçaleres de tots els paquets reconeguts.

Gràfiques i estadístiques de paquets

**9.-** Comença una captura de tràfic qualsevol amb el Wireshark i, al cap d'uns quants segons, atura-la. Tot seguit, vés als següents menús del programa i digues quina informació mostra cadascun:

| | |
|---|---|
| Menú *Statistics* → *Capture file properties* | (i allà les seccions "*File", "Time", "Capture"* i *"Display*") |
| Menú *Statistics* → *Resolved addresses* | |
| Menú *Statistics* → *Protocol Hierarchy* | (la informació mostrada depén del filtre de pantalla actiu) |
| Menú *Statistics* → *Conversations* | (i allà mira els protocols "*Ethernet", "Ipv4", "UDP"* o *"TCP*". Fixa't, en triar-ne un protocol, quines columnes es mostren) |
| Menú *Statistics* → *Endpoints* | (i allà mira els protocols "*Ethernet", "Ipv4", "UDP"* o *"TCP*". Fixa't, en triar-ne un protocol, quines columnes es mostren) |
| Menú *Statistics* → *Packet lenghts* | (fixa't en quines columnes es mostren) |
| Menú *Statistics* → *Ipv4 Statistics* | (en les seves quatre variants: "*All addresses", "Destinations & ports", "IP Protocol types"* i *"Source & destination addresses*"; fixa't en quines columnes es mostren) |
| *Menú Statistics* → *DHCP Statistics* | (fixa't en quines columnes es mostren) |
| Menú *Statistics* → *DNS* | (fixa't en quines columnes es mostren) |
| Menú *Statistics* → *HTTP* → ... | (fixa't en quines columnes es mostren a cada subapartat) |

Si, un cop obert el Wireshark i fetes algunes captures, hom va al menú "**Statistics -> Flow Graph"** es pot veure una representació gràfica del flux de dades entre les diferents connexions realitzades entre els diferents hosts. En aquesta gràfica es pot veure el sentit del flux de dades (representat amb fletxes que, a més, indiquen entre quines màquines s'estableix la connexió) , el temps en el què ha ocorregut cada flux, els ports involucrats (mostrats entre parèntesis) i els número de seqüència i acks (mostrats per cada fluxe a la columna de la dreta, titulada "Comments"). A les opcions inferiors del quadre es pot seleccionar quins paquets volem tractar -tots o només els visualitzats amb un eventual filtre de pantalla- i/o quins protocols volem incloure -tots o només els de tipus TCP, ICMP, etc.

**10.-** Obre amb el Wireshark el fitxer "Conversa_HTTP_completa.pcap" utilitzat a l'exercici 6 i tot seguit vés al menú *Statistics → Flow graph* . Al quadre emergent, sel.lecciona l'opció "TCP Flows" del desplegable "Flow type" per obtenir els números SEQ i ACK de cada paquet de la conversa TCP. Seguidament, respon:

**a)** ¿Quin valor té el camp *seq* del paquet SYN?

**aII)** Després de llegir el següent paràgraf, ¿pots dir si el valor dels camps *seq* mostrat dels diferents paquets al "Flow graph" (i també al panell horitzontal del mig de la pantall principal del Wireshark) és "real", o no?

"When a host initiates a TCP session, its initial sequence number is effectively random; it may be any value between 0 and 4,294,967,295, inclusive. However, protocol analyzers like Wireshark will typically display *relative* sequence and acknowledgement numbers in place of the actual values. These numbers are relative to the initial sequence number of that stream. This is handy, as it is much easier to keep track of relatively small, predictable numbers rather than the actual numbers sent on the wire."

**b)** ¿Quant de temps triga el servidor web en respondre al primer paquet SYN amb el corresponent SYN/ACK? ¿Quin valor tenen els camps *seq* i *ack* (no confondre aquest últim amb el flag "ack") d'aquest paquet SYN/ACK?

> NOTA: El paquet SYN que inicia el "3-way handshake" té 0 com a valor del camp *ack* (per això no apareix mostrat al "Flow graph")

**c)** ¿Quin valor tenen els camps *seq* i *ack* (no confondre aquest últim amb el flag "ack") del paquet ACK que remata el el "3-way handshake"?

Els valors vistos fins aquí dels camps *seq* i *ack* són sempre els mateixos en qualsevol connexió TCP. A partir d'aquí, però, aniran canviant segons la quantitat de dades (bytes) que es transmetin entre els dos extrems. Concretament, el primer paquet que envia dades del client al servidor després de l'establiment de la connexió es correspon, en aquest exemple, a la petició GET realitzada i es pot distingir -en aquest cas- perquè té activat el flag "psh".

**d)** Observa la quantitat de dades que es transmet en la petició (és el valor "Len" mostrat sobre les fletxes) i dedueix, després d'haver llegit el paràgraf blau anterior, per què el valor *ack* del paquet *nº5* és 726. En general, ¿què representen els valors del camp *ack* dels paquet TCP?

> NOTA: Una manera alternativa de saber la quantitat de dades transmeses en un paquet TCP és observar el valor anomenat "TCP Segment Length" mostrat dins el panell horitzontal intermedi de la pantalla principal del Wireshark pel paquet en qüestió

**e)** Troba la relació entre el sentit de les fletxes de la conversa mostrada a "Flow graph" i els valors dels camps *seq* dels diferents paquets. ¿Pots trobar la regla que estableix aquests valors? PISTA: Els valors dels camps *seq* estan relacionats amb els valors *ack* de l'últim paquet anterior provinent de l'altre extrem

> NOTA: Per saber més sobre els números de seqüència i els d'acknowledge podeu llegir aquest post: http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/

Recorda que els passos "canònics" per fer una desconnexió TCP de forma correcta són els següents:

1.-El client es desconnecta del servidor enviant-li un paquet FIN/ACK.
2.-(Opcional) El servidor li respon amb un paquet ACK conforme ha sigut notificat
3.-El servidor es desconnecta del client enviant-li un paquet FIN/ACK
4.-El client li respon amb un paquet ACK final conforme ha sigut notificat. Obligatori.

Com podeu veure, hi ha un intercanvi de 4 paquets (quan a l'establiment de la connexió "només" eren 3). Mentre s'estan intercanviant aquests paquets entre els extrems, la conexió entre client i servidor entra en diferents estats transitoris que són els famosos FIN_WAIT, CLOSE_WAIT, etc. L'extrem que ja s'hagi desconnectat no podrà enviar més dades a l'altre extrem, encara que sí que pot continuar rebent paquets, fins que l'altre extrem també tanqui la connexió per la seva banda. També podria passar que fos el servidor que prengués la iniciativa de tancar la connexió (en aquest cas l'intercanvi de paquets seria similar però "al revés").

**f)** Dedueix, a partir de la informació del paràgraf blau anterior, si la captura mostra una desconnexió TCP correcta (o no).

Si, un cop obert el Wireshark i capturant, hom va al menú "**Statistics -> IO Graph**" es pot veure una gràfica on es mostra la quantitat de paquets detectats pel programa per segon. En aquesta gràfica es poden configurar els següents aspectes:

*A la zona de dades es poden visualitzar múltiples gràfiques (per defecte apareixen dues: una corresponent a tots els paquets i una altra corresponent als errors TCP), cadascuna d'un color diferent i un determinat estil, que es poden canviar. Cada gràfica està associada a un determinat filtre de pantalla per tal de què mostrin només la quantitat de paquets/s que es corresponguin al filtre en qüestió. Per afegir una nova gràfica cal pulsar el botó "+" (i per eliminar-la, el botó "-").

*La resolució de l'eix horitzontal (el del temps) es pot ajustar amb l'opció "Interval" de la barra inferior. En aquesta barra també podem configurar el comportament del ratolí ("drag" o "zoom"), o establir si volem veure els temps de forma relativa o bé atenent al temps de la captura amb "Time of day".

*A la columna "Y Axis" es pot indicar que es vol veure el tràfic capturat per quantitat de paquets, bytes o bits però si s'indica una funció agregada (com ara SUM(), MIN(), MAX(), AVG() entre d'altres), s'haurà d'indicar a més, a la columna "Y Field", quin serà el camp a usar com a font de dades per la funció agregada .

*Una altra columna molt interessant per cada gràfica és la de "SMA period", la qual serveix per esmorteir els canvis bruscos de pics i valls de la gràfica (i així veure l'evolució del trànsit en perspectiva) gràcies a què calcula per cada punt Y el valor mitjà dels 10, 20, 50, etc valors Y que l'envolten.

*El botó "Save as" grava la part actualment visible de la gràfica com a fitxer d'imatge (png) D'altra banda, el botó "Copy" copia a porta-papers els valors de les gràfiques seleccionades en format CSV per tal de poder importar-les i graficar-les, per exemple, en un full de càlcul.

*Es poden accedir a diferents opcions de visualització de la gràfica mitjançant el menú contextual que apareix quan es clica amb el botó dret del ratolí sobre la gràfica mateixa.

**11.-**Descarrega't el fitxer "Analisi_problemes.pcap" de la web del centre i obre'l amb el Wireshark. Aquest fitxer conté una captura que representa una descàrrega HTTP que pateix pèrdua de paquets mentre s'està fent pings en paral.lel al servidor web en qüestió. Vés al menú *Statistics -> IO Graph* i a partir d'aquí segueix les següents instruccions:

**a)** Canvia el valor de les unitats de l'eix X per a què sigui "1 sec" i el de les de l'eix Y per a què sigui "Bits" (en comptes de "Paquets") a les dues gràfiques mostrades per defecte ("Tots els paquets" i "TCP errors" -equivalenta a tenir el filtre "*tcp.analysis.flags*"); d'aquesta manera observaràs les gràfiques mostrant-se amb la ràtio "Bits/s". Oculta de moment la gràfica "TCP errors" (que anomenarem "Graph2" a partir d'ara).

**b)** Afegeix a la gràfica de "Tots els paquets" (que anomenarem "Graph1" a partir d'ara) el filtre "icmp" i fes que tingui un estil "Bar" (li pots canviar el nom també, per a què sigui acorde amb el que mostra). Veuràs que apareixen forats al llarg del temps que no haurien d'aparèixer ja que aquests forats indiquen que el tràfic "ping" s'ha interromput.

**c)** Duplica (amb el botó adient, que està al costat dels botons "+" i "-") "Graph1" (a aquest duplicat l'anomenarem "Graph3"). Canvia-li el color a "Graph3" per distingir-la de "Graph1" i tot seguit modifica el filtre de "Graph1" per a què ara sigui "icmp.type==8" (echo request) i assigna un filtre a "Graph3" que sigui "icmp.type==0" (echo reply). ¿Quin problema es veu a la gràfica resultant?

Es pot realitzar un anàlisis més exhaustiu dels problemes mitjançant un conjunt de filtres de pantalla especialitzats com els següents:

> **tcp.analysis.lost_segment** : Indicates we've seen a gap in sequence numbers in the capture. Packet loss can lead to duplicate ACKs, which leads to retransmissions
>
> **tcp.analysis.duplicate_ack** : Displays packets that were acknowledged more than one time. A high number of duplicate ACKs is a sign of possible high latency between TCP endpoints
>
> **tcp.analysis.retransmission** : Displays all retransmissions in the capture. A few retransmissions are OK, excessive retransmissions are bad. This usually shows up as slow application performance and/or packet loss to the user. There's another similar filter called **tcp.analysis.fast_retransmission**
>
> **tcp.analysis.window_update :** This will graph the size of the TCP window throughout your transfer. If you see this window size drop down to zero(or near zero) during your transfer it means the sender has backed off and is waiting for the receiver to acknowledge all of the data already sent. This would indicate the receiving end is overwhelmed.
>
> **tcp.analysis.bytes_in_flight** : The number of unacknowledged bytes on the wire at a point in time. The number of unacknowledged bytes should never exceed your TCP window size (defined in the initial 3 way TCP handshake) and to maximize your throughput you want to get as close as possible to the TCP window size. If you see a number consistently lower than your TCP window size, it could indicate packet loss or some other issue along the path preventing you from maximizing throughput.
>
> **tcp.analysis.ack_rtt** : Measures the time delta between capturing a TCP packet and the corresponding ACK for that packet. If this time is long it could indicate some type of delay in the network (packet loss, congestion, etc)
>
> **tcp.analysis.initial_rtt** : Measures the total time spent in the 3-way handshake of a TCP connection
>
> **tcp.analysis.out_of_order** : Measures the number of disordered received TCP packets, which often forces to request a retransmission

**d)** Modifica els eixos X i Y per a què tornin a mostrar la ràtio "Packets/s". Fes ara que "Graph1" tingui estil de línia negra i que mostri el tràfic HTTP, que "Graph2" tingui estil de barres vermelles i que mostri els paquets TCP perduts ("lost segments"), que "Graph3" tingui estil de barres verdes i que mostri els paquets TCP amb ACK duplicats i que una nova "Graph4" tingui estil de barres blaves i que mostri els paquets TCP retransmessos. ¿Què veus i què en podries deduir?

**e)** Oculta totes les gràfiques excepte "Graph1" i fes ara que aquesta tingui l'estil de barres, el filtre "http" activat i que mostri en l'eix Y el valor *MIN(frame.time_delta)*. Amb això hauríem de veure el mínim de temps mesurat entre paquets (això és útil per veur la latència que apareix entre paquets individuals). ¿Què passa als 65s, 80s, 90s...aproximadament? ¿A quins paquets concrets es corresponen els pics anteriors?

**f)** Fes ara que "Graph1" segueixi mostrant només tràfic HTTP però ara acompanyant-ho del valor *COUNT FRAMES(tcp.analysis.retransmission)*. ¿Què representa la gràfica que veus?

**g)** Fes que "Graph1" (ara en estil Line) segueixi mostrant només tràfic HTTP però ara acompanyant-ho del valor *SUM(tcp.seq)*. En una connexió fluïda s'hauria de veure una línia ascendent regular degut a l'aument estable del número de seqüència; si no fos així (és a dir, que es veiessin pics i forats) voldria dir que hi ha problemes en la retransmissió TCP. ¿Què veus a la gràfica que apareix?

**h)** Fes que "Graph1" (un altre cop en estil Bar) segueixi mostrant només tràfic HTTP però ara exclusivament el sortint de la IP 192.168.1.4 (pensa el filtre que ha de ser) i fes que al seu eix Y es vegi el valor *SUM(tcp.len)*. Afegeix de nou "Graph2" (també en estil Bar) que mostri també només tràfic HTTP però exclusivament l'entrant a la IP 192.168.1.4, i fes que al seu eix Y es vegi el valor *SUM(tcp.len)*. El resultat final a cada gràfica haurà de ser la quantitat total de bytes enviats i rebuts, respectivament. ¿Quina de les dues gràfiques mostra un valor més elevat? Per què?

**i)** Amb un paquet TCP qualsevol sel.leccionat, vés al menú *Analyze → Follow → TCP Stream* i digues, a partir de la informació que allà hi apareix, quin és el fitxer la descàrrega del qual ha generat la captura estudiada.

**j)** Fes ara que Wireshark torni a capturar tràfic en temps real i fes algun ping a diversos servidors d'Internet (*ping www.hola.com*, etc). Vés de nou al menú *Statistics -> IO Graph* i construeix les següents gràfiques (totes amb el filtre de pantalla *dns* assignat i d'estil Dot però de colors diferents): "Graph1" amb valor AVG(*dns.time*), "Graph2" amb valor MIN(*dns.time*) i "Graph3" amb valor MAX(*dns.time*). ¿Què veus? (cal que sàpigues què significa el filtre *dns.time*, òbviament)

**k)** Llegeix aquest article (https://www.cellstream.com/2017/11/06/real-life-wireless-wireshark-troubleshooting-example) i digues quin és el problema que descriu i la solució.