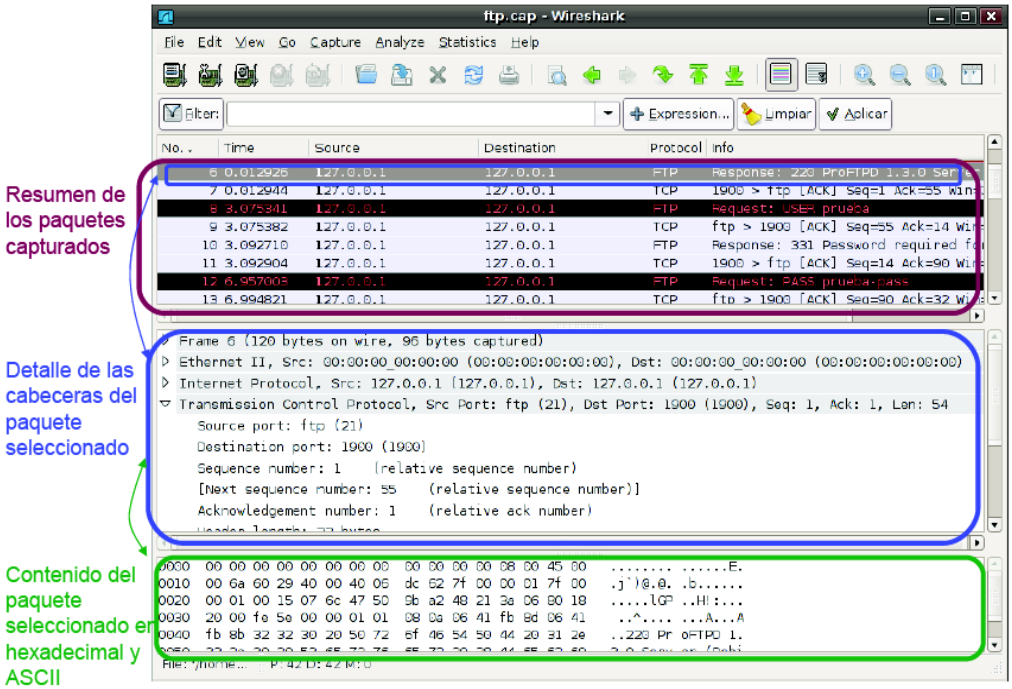
L'analitzador de protocols Wireshark (I)

Introducció:

Wireshark (http://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 o bé 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 aquí: http://www.tcpdump.org/manpages/pcap-filter.7.html

Filtres de captura	
[src|dst] host *dir_IP* **[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* mask *mascara* **[src|dst] net *dir_IP/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 [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).
{ether|ip} broadcast	Filtra els paquets la direcció MAC (*ether*) o la direcció IP (*ip*) dels quals sigui broadcast.
[tcp|udp] [src|dst] port *nºport* **[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 **ip** **icmp** **udp** **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. La referència completa de tots els filtres de pantalla possibles es pot trobar aquí (https://www.wireshark.org/docs/dfref) però 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: **==, !=,** >, <, >= 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*
*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

Com es pot veure al 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). Si es vol comparar, no obstant, amb expressions regulars (case-insensitive i de tipus PCRE!) llavors l'operador a usar és **matches** (o també ~). Per exemple, *http contains "https://www.wireshark.org"* mostrarà 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)"* mostrarà 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

*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"*

Finalment, 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}*)

Wireshark també ofereix un conjunt de funcions que es poden utilitzar dins els filtres de pantalla, entre d'altres:

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...*

Però per utilitzar aquesta notació, primer haurem de conèixer, no obstant, el tamany i l'ordre dels diferents camps de les capçaleres més importants. A continuació mostro l'enllaç 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).

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 primer byte de la capçalera IP en realitat està format per dos camps diferents, cadascun de 4 bits (versió del protocol i tamany de la capçalera -mesurat en número de paraules de 32 bits-, respectivament). Si volguéssim filtrar només per un valor pel segon 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 concretament 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*

EXERCICIS:

1.-a) Esborra la teva catxé ARP (recorda: *ip neigh flush dev enp0s3*) i obre el Wireshark. Aplica-hi el filtre de pantalla *arp || icmp* i seguidament, obre un terminal i fes "ping" durant uns segons a algun ordinador de la teva xarxa local. Atura la captura. ¿Quin tipus de paquets veus al Wireshark?

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?

c) ¿Quina és, en canvi, la direcció MAC d'origen d'un paquet "ARP reply" (arp.opcode == 2)?

d) 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?

e) ¿Quin és, en canvi, el valor del camp Type de la capçalera d'un paquet ICMP reply?

f) ¿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)?

g) ¿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)?

h) 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?

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

j) 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.

k) 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?

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

2.-a) Torna a iniciar una nova captura però ara amb el filtre *bootp* i seguidament executa les comandes *dhclient -r* i, a continuació, *dhclient -v* Hauries de veure al Wireshark 5 paquets: 3 generats per la teva màquina ("DHCP Release", "DHCP Discover" i "DHCP Request") i 2 respostes del servidor ("DHCP Offer" i "DHCP ACK"). Pots consultar https://www.eventhelix.com/RealtimeMantra/Networking/dhcp-flow/dhcp-sequence-diagram.pdf per conèixer el significat de cada paquet. La pregunta és: ¿per què les direccions IP d'origen i 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 i 6 de l'estàndar.

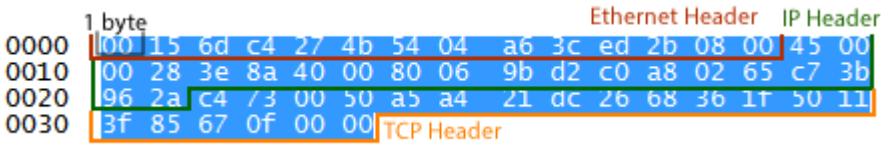
3.-a) Torna a iniciar una nova captura però ara amb el filtre *dns contains lecturas* i seguidament executa la comanda *dig www.lecturas.com* A partir de la informació que el Wireshark t'ofereix, esbrina quin és el significat del filtre *dns.flags=0x0120* .

b) ¿I el significat del filtre *dns.a == 104.27.165.161*, quin és?

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 Avahi 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.

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

5.-a) Digues què estaries buscant si fessis servir cadascun dels següents filtres de pantalla:

!(arp or icmp or dns)
eth.addr[0:3]==00:06:5B

frame.len <128
frame contains un:a:dir:ecc:io:mac
ip contains un:a:dir:ecc:io:mac
udp contains un:a:dir:ecc:io:mac
udp[2:2] > 4000
tcp.seq ==321345
tcp.flags.reset == 1
http.request.uri contains "flv"
http.request.method == "GET"
http.response.code == 404
http.content_type contains "video"
http.content_type[0:5] == "video"

b) ¿Com podria ser un filtre de pantalla que només et mostri el tràfic dirigit de la IP de la teva màquina amb destí 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 l'*etherwake* o el *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/reference-reading/tipsandtricks/353-wireshark-display-filter-macros) i digues què explica

6.-Si executessis la comanda *dig* per esbrinar la direcció IP de les màquines anomenades www.elmundo.es i www.marca.com veuries que és la mateixa: això significa que els dos llocs webs s'allotjen al mateix servidor. Al fitxer "peticion_marca.cap" (descarregable a la web del centre) hi ha capturats els missatges intercanviats en obrir en un navegador la pàgina principal de www.marca.com. Obre aquest fitxer amb el Wireshark i observa els paquets capturats:

*Els 6 primers missatges són els que es generen durant la resolució DNS del nom www.marca.com
*Els missatges 7,8 i 9 són els corresponents a l'establiment de la conexió TCP entre la màquina client i el servidor web (és a dir, el "3-way handshake": SYN, SYN/ACK, ACK)
*El missatge 10 és la petició HTTP de la pàgina d'inici (GET / HTTP/1.1)
*La resta de missatges són els generats per poder rebre tots els objectes de la pàgina web

a) Quan el servidor rep el missatge 10 amb la petició de la pàgina d'inici, ¿com sap quin és el lloc web, d'entre els varis que pot allotjar, del qual ha d'enviar aquesta pàgina inicial? Pista: consulta les capçaleres de la petició; una manera de conèixer les capçaleres d'una petició HTTP és sel.leccionant un missatge concret (en aquest cas el nº10) i escollint llavors l'opció "Follow HTTP Stream" del seu menú contextual

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) del lloc oficial de Wireshark, concretament aquí: https://wiki.wireshark.org/SampleCaptures

b) Aplica el filtre de pantalla *http* i observa que la primera petició obté com a resposta del servidor un codi 301 (important aquí tenir en compte també el valor de la capçalera de servidor "Location") i la segona petició un codi 200. ¿Per què es produeix aquest diàleg i quina és la conseqüència?

c) Vés al menú "File->Export->Objects->HTTP" del Wireshark i selecciona el paquet nº27 per descarregar-te el seu contingut associat (el pots anomenar "marca.html" perquè representa la pàgina web inicial). Un cop descarregat, obre-la amb el navegador. ¿Quin contingut pots deduir que té llavors, a partir del que veus, el paquet nº 141?

NOTA: Això també ho podries deduïr si segueixes l'"HTTP Stream"

d) Esbrina la URL correcta del favicon (https://ca.wikipedia.org/wiki/Icona_de_web) de marca.com observant el "HTTP Stream". Fes-la servir per visualitzar-lo en un navegador

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

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*). 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 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 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.dst == ip.vict.ima*** per facilitar la inspecció).

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.

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 apliqui l'estàndar TCP...podeu trobar més informació a la secció del manual de nmap corresponent al paràmetre -sN). Sabent això, comprova que els paquets enviats per Nmap als diferents ports no tinguin, efectivament, cap flag activada i que els paquets rebuts com a resposta mostrats al Wireshark 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 està tancat fent servir el paràmetre *-sS* de Nmap? ¿I si està obert? ¿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 està tancat fent servir el paràmetre *-sA* de Nmap? ¿I si està obert? ¿Quin/s "flags" té activats el paquet inicial que envia Nmap en aquest tipus d'escaneig?

8.-Llegeix els articles https://seguridadyredes.wordpress.com/2010/04/05/wireshark-captura-conversaciones-voip-protocolo-sip-sdp-y-rtp-extraccion-de-audio i https://seguridadyredes.wordpress.com/2010/03/24/wireshark-tshark-capturando-impresiones-en-red i digues quina és la informació que s'acaba obtenint en cada cas a partir de l'anàlisi dels paquets capturats

L'analitzador de protocols Wireshark (II)

Gràfiques i estadístiques de paquets:

1.-Prova les següents possibilitats que ofereix el Wireshark i digues per a què serveixen:

Menú *File* → *Export specified packets* (i escull allà l'opció "Specify a packet range" tenint en compte si està sel.leccionada la columna "Captured" o bé "Displayed")
Menú *File* → *Export packets dissections* → *as plain text*
Menú *File* → *Export selected packet bytes* (hauràs de sel.leccionar abans alguns bytes del panell inferior)

Menú *Edit* → *Find a packet* (i allà prova les opcions "*Display filter/Hex value/String/Reg Expression*")
Menú *Edit* → *Mark/Unmark packet* (hauràs de sel.leccionar abans un paquet)

Menú *Edit* → *Preferences* → *Appearance* → *Layout*
Menú *Edit* → *Preferences* → *Appearance* → *Columns*
Menú *Edit* → *Preferences* → *Appearance* → *Font & Colors*
Menú *Edit* → *Preferences* → *Capture*
Menú *Edit* → *Preferences* → *Name resolution*

Menú *View* → *Time display format*
Menú *View* → *Coloring rules* (fixa't en els filtres de pantalla corresponents!)

Menú *Go* → *Next packet in conversation*

Menú *Analyze* → *Follow TCP/UDP/HTTP stream* (hauràs de sel.leccionar abans un paquet)
Menú *Analyze* → *Expert information*

PISTA: Aquesta opció (a la qual també es pot accedir fent clic sobre la rodona de color que apareix a la cantonada inferior esquerra de la finestra principal de Wireshark) serveix per detectar anomalies a la xarxa. Apareix una finestra-resum on s'indiquen els comportaments inusuals de la xarxa que Wireshark ha sigut capaç de detectar analitzant tota la captura realitzada fins llavors, com ara retransmissions, fragmentacions o bé tècniques usades per evadir IDS o enganyar sistemes en general.

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 sel.leccionar-ne un protocol, quines columnes es mostren)
Menú *Statistics* → *Endpoints* (i allà mira els protocols "*Ethernet"*, *"Ipv4"*, *"UDP"* o *"TCP*". Fixa't, en sel.leccionar-ne un protocol, quines columnes es mostren)
Menú *Statistics* → *Packet lenghts* (fixa't en quines columnes es mostren)
Menú *Statistics* → *Ipv4* (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* (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)

Menú *Wireless* → *WLAN traffic* (fixar't en quines columnes es mostren)

2.-Descarrega't el fitxer pcap corresponent a aquest exercici disponible a 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) Utilitza el menú *File → Export objects → HTTP* per tal guardar al teu disc dur aquesta imatge

b) 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 Selected Packet Bytes*

> NOTA: Per a què el apartats anteriors funcionin cal que l'opció Edit → Preferences → Protocols ->TCP → "Allow subdissector to reassemble TCP streams." estigui activada (cosa que per defecte ja ho està)

c) Ara torna a fer el mateix però seguint aquests passos 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 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) 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* : extrau a la carpeta actual el/s fitxer/s binari la descripció del qual 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

Si, un cop obert el Wireshark i capturant, 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 (s'ha de seleccionar prèviament quins paquets volem tractar -tots o només els visualitzats- i quins protocols volem incloure -tots o només els de tipus TCP-). 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 conexió) , el temps en el què ha ocorregut cada flux, els ports involucrats (mostrats entre parèntesis), els número de seqüència i acks (mostrats per cada fluxe a la columna de la dreta, titulada "Comments")

3.- Tenint obert al Wireshark el mateix fitxer pcap utilitzat a l'exercici anterior, vés ara al menú *Statistics → Flow graph* (i sel.lecciona l'opció "TCP Flow") per obtenir els números SEQ i ACK de cada paquet de la conversa TCP. Seguidament, respon:

a) ¿Quin valor tenen els camps *seq* i *ack* (no confondre aquest últim amb el flag "ack") d'aquest paquet SYN? ¿Per què el Wireshark mostra pel camp *seq* un valor diferent segons es miri al panell horitzontal del mig (que seria el mateix valor que el mostrat al diagrama "Flow") o bé si s'obté del panell inferior? Pista: La resposta està en aquest paràgraf: "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?

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"?

d) ¿Quin valor tenen els camps *seq* i *ack* (no confondre aquest últim amb el flag "ack") del primer paquet que envia "dades" del client al servidor després de l'establiment de la connexió (en concret, la petició GET que es vol realitzar)?

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

e) Busca dins de la capçalera TCP del paquet nº4 el valor "TCP Segment Length" i dedueix, a partir de la informació del paràgraf blau anterior, per què el valor *ack* del paquet *nº*5 és 726. En general, ¿què representen els valors del camp *ack* dels paquet TCP?

f) 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 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".

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

4.- Tenint obert al Wireshark el mateix fitxer pcap utilitzat a l'exercici anterior, utilitza el menú *Analyze → Follow HTTP Stream* per conèixer les capçaleres i peticions HTTP del client i les capçaleres i respostes HTTP del servidor. Concretament, respon:

a) Digues quina és la URL sencera de la petició realitzada, observant la petició GET i la capçalera client *Host:* . 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

b) Digues quin navegador va realitzar la petició observant el valor de la capçalera client *User-Agent*

c) Digues quin programa servidor web va contestar i quan observant el valor de les capçaleres servidor *Server* i *Date*

d) Digues quin és el significat de les capçaleres servidor *Content-Type* i *Content-Length*

e) Digues què és el que es veu a partir de la línia ".PNG"

f) Digues quants bytes ha enviat en total el client al servidor i quants ha enviat el servidor al client

5.- Tenint obert al Wireshark el mateix fitxer pcap utilitzat a l'exercici anterior, vés al menú *View → Coloring rules* i copia el filtre de pantalla corresponent al color "Bad TCP" per tal de convertir-lo en un nou filtre de pantalla disponible per usar amb el nom "malo". Prova-ho.

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 X" (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 "Eix Y" 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 "Camp Y", quin serà el camp de capçalera a usar com a font de dades per la funció agregada .

*Una altra columna molt interessant per cada gràfica és la de "SMA interval", 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

*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.

6.-Descarrega't el fitxer pcap corresponent a aquest exercici disponible a la web del centre i obre'l amb el Wireshark. Aquesta captura 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"). D'aquesta manera observaràs una gràfica mostrant-te la ràtio "Bits/s".

b) Afegeix a la primera gràfica que apareix a la llista mostrada a la meitat inferior de la finestra (l'anomenarem "Graph1") el filtre "icmp" i fes que tingui un estil "Bar". Veuràs que apareixen forats al llarg del temps que no haurien d'aparèixer.

c) Modifica el filtre de "Graph1" per a què ara sigui "icmp.type==8" (echo request) i afegeix un nou filtre a "Graph2" (és a dir, la segona gràfica que apareix llistada) que sigui "icmp.type==0" (echo reply). Fes que l'estil de totes dues gràfiques sigui ara "Line". ¿Què veus? ¿Quin extrem sembla l'origen del problema?

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.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 "Graph4" tingui estil de barres blaves i que mostri els paquets TCP retransmessos. ¿Què veus i què en pots deduir?

e) Fes ara que "Graph1", "Graph2" i "Graph3" (totes amb estil Bar i amb el filtre "http" activat) mostrin en l'eix Y, respectivament, els valors *AVG(frame.time_delta)*, *MIN(frame.time_delta)* i *MAX(frame.time_delta)*. Amb això hauríem de veure la mitja, el mínim i el màxim 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 quin n.º de paquet es correspon exactament cada pic?

f) Desactiva "Graph2" i "Graph3" i ara fes 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) Llegeix aquest article (https://www.cellstream.com/reference-reading/tipsandtricks/366-real-life-wireless-wireshark-troubleshooting-example) i digues quin és el problema que descriu i quina és la seva solució.

Una gràfica molt interessant és la que es mostra si hom, després d'haver sel.leccionat un determinat paquet TCP pertanyent a un extrem concret d'una determinada conversa, va al menú **Statistics → TCP Stream Graphs → Time-Sequence Graph (tcptrace)** . Una explicació molt acurada sobre el significat dels valors allà mostrats (bàsicament, el número de seqüència dels paquets enviats per l'extrem sel.leccionat -és a dir dels paquets del "fluxe" sel.leccionat- respecte el temps que dura la conversa) es troba a http://packetbomb.com/understanding-the-tcptrace-time-sequence-graph-in-wireshark En condicions ideals la representació del fluxe sel.leccionat hauria de mostrar una línia creixent amb el temps que indicaria un rendiment eficient de la conexió TCP. No obstant, en algunes ocasions trobarem forats que interrompeixen la continuïtat de la línia; els problemes més importants que es poden detectar en aquesta gràfica són bàsicament dos: les retransmissions de paquets (apareixen vàries barres verticals en diferents instants totes amb el mateix número de seqüència i acks) y els ack duplicats (apareix una línia horitzontal; són un símptoma de desordre de paquets i pèrdua de dades).

> NOTA: Podeu veure una relació de tecles interessants per moure's més còmodament a través de la gràfica si feu clic amb el botó dret del ratolí sobre ella. D'altra banda, altres gràfiques interessants que es poden veure a partir del menú Statistics-> TCP Stream Graphs són **Window Scaling, Throughput** i **Round Trip Time**

Esnifar HTTPS (i altres):

Si intenteu capturar els paquets pertanyents a una conversa HTTPS amb el Wireshark veureu que no obtindreu cap paquet HTTPS sinò tot un seguit de paquets TCP i TLS. Això és perquè la comunicació HTTPS viatja encriptada (gràcies precisament al protocol SSL/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 això implica haver de fer una de les següents accions:

> NOTA: De fet, 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

a) Si el servidor HTTPS del qual volem espiar el seu trànsit és de la nostra propietat, sempre podrem exportar la clau privada d'aquest servidor (en forma de fitxer en format PEM o PKCS#12...si està en algun altre format caldrà usar la comanda *openssl* amb els paràmetres adients per transformar-la) i importar-la al Wireshark anant al menú **Edit →**

Preferences → Protocols →TLS i clicant allà sobre el botó **"Edit"** rera l'opció "RSA keys list". En el quadre que apareix haurem d'introduir les següents dades:

*Direcció IP del servidor (es pot escriure "any" també).
*Port on escolta el servidor (normalment serà 443).
*Protocol (cal escriure "http")
*Fitxer de clau privada (s'agafa d'allà on la tinguem guardada)
*Contrasenya de la clau privada (només en el cas de claus PKCS#12)

NOTA: Si la clau està en format PEM, en realitat conté no només la clau sinò també el certificat del servidor. Com que només necessitem la clau, el que hem de fer és obrir la clau amb un editor de texte qualsevol i quedar-nos amb l'interior de la secció que comença per PRIVATE KEY (incloent capçalera i peu) i crear un nou fitxer només amb aquest contingut, que representarà la clau privada que hem d'indicar

b) Si el servidor HTTPS no és de la nostra propietat haurem d'aprofitar la capacitat que té tot navegador per comunicar-se amb ell (fent servir en aquest cas la seva clau pública). El procediment és el següent:

1.-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) on el navegador a emprar tingui permisos d'escriptura. Per exemple, executant: ***export SSLKEYLOGFILE=~/hacker.txt*** És recomanable afegir aquesta comanda dins de l'arxiu ~/.bashrc per tal de què s'executi automàticament sempre que s'obri un terminal. La idea és que en arrencar el navegador, 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.
NOTA: El contingut d'aquest fitxer ve donat pel format detallat aquí:
https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

2.-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.-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 "Developer Edition", disponibles a https://www.mozilla.org/en-US/firefox/developer Un cop descarregat i descomprimit el paquet firefox*.tbz, 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: Tant l'acció a) com l'acció b) funcionaran sempre i quan Wireshark hagi estat compilat amb la llibreria GnuTLS (no pas OpenSSL). D'altra banda, l'acció a) obliga a més a què les claus a emprar per l'intercanvi inicial siguin de tipus RSA (però no pas DHE, també molt habituals!). Afortunadament, la primera condició es compleix a la majoria de distribucions importants (es pot comprovar fent *wireshark -v*) i la segona es compleix a molts servidors segurs del món. Si aquesta segona no es complís, però, una "solució" seria deshabilitar al client la capacitat de treballar amb claus d'intercanvi que no fossin RSA per tal d'obligar al servidor, en una cascada de prova-i-error a acabar finalment utilitzant una clau RSA. Per exemple, al Firefox això es fa anant a la direcció *about:config* i establint a false totes les directives amb "dhe" i "ecdhe" en el seu nom.

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

Un cop feta una de les dues accions anteriors, ja podrem veure al Wireshark el tràfic HTTPS (i HTTP2!) com si fos tràfic HTTP tradicional

7.-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.

Tshark:

Tshark és la versió en línia de comandes de Wireshark (a Ubuntu s'instal.la mitjançant el paquet homònim, a Fedora s'instal.la amb el paquet "wireshark-cli"). Per fer-lo servir, primer s´ha d´executar amb el paràmetre *-D* per llistar les interfícies disponibles, i seguidament, per començar a capturar, s´ha d´escriure la comanda amb el paràmetre *-i* seguit del número d´interfície observat abans amb *-D* (també es pot indicar com a valor de *-i* el nom de la tarja a fer servir per capturar)

*Per especificar filtres de captura cal fer-ho amb el paràmetre *-f "filtreCaptura"*.
*Per especificar filtres de pantalla cal fer-ho amb el paràmetre *-Y "filtrePantalla"*
*Per obrir un fitxer amb paquets capturats s'usa el paràmetre *-r nomFitxer*.
*Per guardar els paquets capturats en un fitxer s'usa el paràmetre *-w nomFitxer* (no es veurà res en pantalla llavors, a no ser que s'indiqui també el paràmetre *-P*).

Per defecte Tshark mostra una línia per paquet detectat amb la següent informació:

```
marca temps, IP origen, IP destí, protocol, port origen, port destí, informació paquet
```

Aquesta informació es pot personalitzar afegint els següents paràmetres:

-T {fields|json|ps|psml|pdml|text} : Indica el format de sortida de les dades: camps específics (cal fer servir a més el paràmetre *-e* per indicar-los, fent servir notació de filtres de pantalla), JSON, Postcript, XML ("psml" o "pdml"), o texte (per defecte), entre d'altres.

-e nom.camp1 -e nom.Camp2 : Si s'indica el paràmetre *-T fields*, aquest paràmetre cal escriure'l per especificar (fent servir notació de filtres de pantalla) cada camp que es vol veure. Per exemple, si es vol la IP d'origen d'un paquet s'escriurà el paràmetre *-e ip.src*

-E headers=y/n : Indica si s'imprimeix la capçalera dels camps indicats amb *-e*
-E separators={ /t | /s | caracter }: Indica si el separador dels camps indicats amb *-e* és el tabulador, l'espai en blanc o un altre caracter a indicar.

Altres paràmetres interessants de Tshark són:

-n : Deshabilita la resolució de noms (de hosts i de ports)

-V : Mostra per cada paquet la seva dissecció, similar al panell intermig de Wireshark. Una altra opció similar és *-O protocol*, com ara *-O icmp*

-x : Mostra directament en hexadecimal el contingut de cada paquet (i la seva correspondència en ASCII), similar al panell inferior de Wireshark

-q : No mostra per pantalla els paquets capturats en temps real (útil per si només volem realitzar estadístiques amb les dades finals o les volem guardar en un fitxer sense "embrutar" la pantalla)

--color : Mostra els paquets en colors segons el seu tipus (a partir de la versió 3.5)

-c nº : Para la captura quan s'hagin capturat nº paquets
-a duration:nº : Para la captura quan hagin passat nº segons
-a filesize:nº : Para la captura quan s'hagi arribat a nºKB de paquets capturats

-s nº : Indica la quantitat de bytes a capturar de cada paquet (començant pel principi). Aquesta opció seria equivalent a la del menú "Capture->Options->Limit each packet n bytes" del Wireshark. Si nº fos **14** només veuríem la capçalera Ethernet de cada paquet; si fos **34**, la capçalera IP; si fos **42** la capçalera UDP i si fos **54** la TCP.

-t {a|ad|u|ud|r|d|dd} : Especifica el format de la columna de temps del paquet capturat. "a" indica temps absolut local, "ad" indica el mateix però afegint la data, "u" indica temps absolut UTC, "ud" indica el mateix però afegint la data, "r" indica el temps relatiu (és a dir, la diferència) entre el temps del primer paquet capturat i l'actual, "d" indica el temps delta (és a dir, la diferència) entre el temps del paquet capturat anterior i l'actual, "dd" indica el temps delta entre el temps del paquet mostrat anterior i l'actual. El valor per defecte és "r".

-v : Mostra la versió del programa i les opcions de compilació

El Tshark també permet obtenir gràfiques i estadístiques tal com ja hem vist amb el seu "germà gran" gràfic, el Wireshark. Concretament, els següents paràmetres es poden fer servir:

-z io,stat,interval, *filtre,filtre,...* **:** Al final de la captura mostra la quantitat total de paquets i bits capturats per cada interval de temps (l'indicat), aplicant-hi opcionalment varis filtres de pantalla.

NOTA: Com a filtre també es poden usar funcions "d'agregació" d'aquesta manera: *"AVG(capçalera.camp)capçalera.camp","MIN(capçalera.camp)capçalera.camp","MAX(capçalera.camp)capçalera.camp","COUNT(capçalera.camp)capçalera.camp"* o *"SUM(capçalera.camp)capçalera.camp"* sempre que *capçalera.camp* contingui un valor numèric. Per exemple: *-z io,stat,1,MIN(ip.ttl)ip.ttl*

-z io,phs, *filtre* : Al final de la captura mostra la jerarquia de protocols capturats, indicant per cadascun la quantitat total de paquets i bits capturats. Es pot indicar un filtre de pantalla.

-z conv,tipus,*filtre* : Al final de la captura mostra la llista de "conversacions" entre els hosts (ips dels extrems, quantitat de paquets i bytes rebuts, transmesos i total, duració de la conversa i el seu ordre relatiu, etc). El "tipus" pot ser, entre altres: *eth*, *ip*, *tcp* o *udp*.

-z endpoints,tipus,*filtre* : Al final de la captura mostra la llista d'"endpoints" entre els hosts (ips dels extrems, quantitat de paquets i bytes rebuts, transmesos i total, duració de la conversa i el seu ordre relatiu, etc). El "tipus" pot ser, entre altres: *eth*, *ip*, *tcp* o *udp*.

-z http,tree,*filtre* : Ofereixen informes estadístics de transaccions HTTP, codis de resposta, redireccions, enllaços trencats, etc. També es poden veure només les estadístiques de peticions (***-z http_req,****filtre*) o de respostes (***-z http_srv,****filtre*).
-z dns,tree*,filtre* : Ofereixen informes estadístics dels paquets DNS capturats
-z bootp,tree*,filtre* : Ofereixen informes estadístics dels paquets DHCP capturats
-z dests,tree, *filtre* : Mostra informació estadística (número de paquets, mijtana, etc) sobre les IPs, protocols de transport i ports de destí existents a la captura
-z ptype,tree : Mostra informació estadística sobre els protocols IPs existents a la captura
-z ip_hosts,tree : Mostra informaci estadística sobre els hosts que intervenen a la captura

-z follow,tcp,{hex|ascii|raw},ipOrigen:port,ipDesti:port : Al final de la captura mostra la conversa TCP entre els dos extrems indicats. També es pot mostrar una conversa indicant el seu número (nº) mitjançant ***-z follow,tcp,{hex|ascii|raw},nº***

8.-a) Utilitza els paràmetres adeqüats del Tshark per capturar durant 10 segons només les direccions IP de destí dels paquets UDP que surtin de la teva màquina amb destí al port 53 i guarda aquesta informació en un fitxer. Executa en un altre terminal durant aquests deu segons la comand *dig* un parell de cops per tal de generar algun tràfic que coincideixi amb el filtre indicat. Un cop finalitzada la captura, obre aquest fitxer amb el Wireshark. ¿Què veus?

b) Obre un terminal i executa la comanda *mtr 8.8.8.8* . Obre un altre terminal i executa la comanda *tshark -f "icmp" -T fields -e frame.number -e frame.time_relative -e ip.src -e ip.dst -e ip.ttl -e _ws.col.Protocol -e _ws.col.Info -E occurrence=f* ¿Què veus?

NOTA:Les columnes que tenen un nom que comença per "_ws.col" són columnes que proporciona el propi Wireshark com a comoditat però no es corresponen a cap camp concret dels paquets. Per veure tots els filtres possibles que ofereix tshark incloent aquests "especials", es pot fer *tshark -G fields*
NOTA:El paràmetre *-E occurrence=f* serveix per no mostrar dades duplicades

c) Sabent que els missatges DHCP són un tipus concret de missatges BOOTP identificats pel tipus 53 i que les diferents opcions d'un missatge DHCP indiquen els diferents valors que un servidor DHCP pot assignar a un client (1=màscara, 3=porta d'enllaç, 6=servidors DNS, etc), ¿què és el que veus si executes aquesta comanda: *tshark -ni enp0s8 -Y "bootp.option.type==53" -T fields -e bootp.option.hostname -e eth.src_resolved* ?

d) ¿Quin és el significat de la informació que es mostra a pantalla en executar aquesta comanda?: *tshark -n -T fields -E separator=';' -e ip.src -e tcp.srcport -e ip.dst -e tcp.dstport -Y "(tcp.flags.syn == 1 and tcp.flags.ack == 0)"*

e) Digues què significa cadascuna de les columnes mostrades per la comanda següent i prova-la: *tshark -Y "ip.src==la.te.va.IP" -T fields -e tcp.stream -e tcp.seq -e tcp.ack -e tcp.flags.str -e tcp.flags*

f) ¿Quin és el significat de la informació que es mostra a pantalla en executar cadascuna de les següents comandes (i navegar mentrestant per Internet)?

tshark -Y "http.request.method == 'GET'" -T fields -e http.request.uri
tshark -Y "http.request.method == POST" -T fields -e http.file_data
tshark -Y "http.response.code == 200" -T fields -e data

g) Digues quines dades apareixen per pantalla en finalitzar l'executació de les següents comandes (també es podrien fer servir sobre la informació obtinguda en un fitxer de captura prèviament generat):

tshark -q -z io,phs
tshark -q -z io,stat,60,ip.addr==la.te.va.IP
tshark -q -z io,stat,30,"COUNT(tcp.analysis.retransmission) tcp.analysis.retransmission"
tshark -q -z conv,tcp
tshark -q -z conv,tcp,http
tshark -q -z dns,tree -z http,tree

h) ¿Per a què serveix el paràmetre -o de tshark i quina funció concreta creus que tindria en aquesta comanda?:
tshark -o "ssl.keylog_file: sslkeys.txt" -x -Y "http.content"

i) Suposant que tens un fitxer "captura.pcap" (obtingut en haver executat en algun moment la comanda *tshark -w captura.pcap*), digues per a què serviria aquest script:

```
#!/bin/bash
for stream in $(tshark -r captura.pcap -T fields -e tcp.stream | sort -n | uniq)
do
        echo $stream
        tshark -r captura.pcap -w stream-$stream.pcap -Y "tcp.stream=$stream"
done
```

iII) De forma similar, digues per a què serviria aquest bucle:
for acapfile in (.pcap) do tshark -r $acapfile -Y "dns" -w $acapfile-dns*

j) Consulta la pàgina de manual de *tshark* (o també l'article https://www.cellstream.com/reference-reading/tipsandtricks/328-wireshark-ring-buffer-capture-from-the-command-line-using-t-shark per tal d'esbrinar per a què serveix el seu paràmetre *-b files* , *-b filesize* i *-b duration*

Altres:

9.-a) Imagina que vols investigar el trànsit de paquets que viatja per una màquina remota anomenada "miservidor" amb el Tshark i un servidor SSH funcionant però sense entorn gràfic. Si a la teva màquina local tens instal.lat el Wireshark i executes allí la següent comanda, ¿què estaràs fent? (fixeu-vos en els valors dels paràmetres -f i -w de tshark i en els paràmetres -k i -i de Wireshark): *ssh usuari@miservidor "tshark -f 'port !22' -w -" | wireshark -k -i -*

b) ¿Per a què creus que serveix la interfície "sshdump" que apareix en iniciar el Wireshark? ¿Què és el que hauries d'escriure a l'apartat "Remote command" del quadre emergent que es mostra en sel.leccionar aquesta interfície? (per més informació pots consultar https://www.wireshark.org/docs/man-pages/sshdump.html)

10.- a) Resum en poques paraules el que s'explica a https://wiki.wireshark.org/CaptureSetup/USB

b) Llegeix el següent fòrum i digues si és fàcil/barat (o no) capturar paquets LTE (4G) amb Wireshark: https://osqa-ask.wireshark.org/questions/36931/capturing-lte-packets Llegeix primer https://es.wikipedia.org/wiki/Radio_definida_por_software i, a continuació, aquests dos articles per deduir si és fàcil/barat (o no) capturar paquets GSM (2G) amb Wireshark:
https://ferrancasanovas.wordpress.com/cracking-and-sniffing-gsm-with-rtl-sdr-concept i
https://payatu.com/passive-gsm-sniffing-software-defined-radio

c) ¿Per a què serveix aquest programa: https://rftap.github.io

11.-Llegeix les pàgines del manual de les següents comandes i digues quin paper juga cadascuna dins de la suite Wireshark:

dumpcap -i 2 -w captura.pcap
mergecap captura.pcap unaaltra.pcap
capinfos captura.pcap
editcap captura.pcap