Capítulo 18

Captura y análisis

Al terminar este capítulo, entenderás:

- Algunas de las opciones que `tshark` y `wireshark` ofrecen para captura y analisis de tráfico de red.
- Qué son los filtros de captura y cómo se usan.
- Qué son los filtros de visualización y qué posibilidades ofrecen.
- Cömo generar estadísticas y resultados a medida.
- Cómo recuperar la carga útil de un flujo UDP o conexión TCP.

Ya has realizado muchas capturas de tráfico con `tshark` desde los primeros capítulos. Pero todas esas veces ha sido con un uso muy básico del programa, `tshark` es capaz de mucho más: puede diseccionar mensajes, interpretar el significado de cada campo, mostrar la información de diferentes formas, generar estadísticas... entre otras cosas. En este capítulo verás de qué más es capaz `tshark`, y su hermano mayor `wireshark`, una versión con interfaz gráfica más potente y versátil.

Este tipo de programas se llaman «analizadores de protocolos» o *sniffers*, y resultan extremadamente útiles para detectar problemas en la red, optimizar protocolos y depurar aplicaciones. Permiten capturar tráfico de una interfaz de red específica o de todas, visualizarlo en tiempo real, o almacenarlo en un archivo para poder analizarlo después, y estudiar capturas que han hecho otros.

Además, permiten aplicar filtros de captura para procesar únicamente los mensajes que coinciden con el criterio que te interese, y también filtros de visualización, para elegir qué mensajes de la captura se muestran en cada momento. Es importante entender esta diferencia. El filtro de captura es permanente, los mensajes descartados por el filtro pierden y no formarán parte de la captura. El filtro de visualización, en cambio, se puede aplicar, modificar o eliminar en cualquier momento y no afecta a la captura, solo a lo que se muestra.

18.1. tshark

A partir de las múltiples capturas que hemos realizado con `tshark` seguro que ya tienes bastante clara su utilidad. `tshark` es un analizador similar al veterano `tcpdump`, aunque con mucha más funcionalidad y posibilidades. Retomemos una captura sencilla de tráfico ICMP como las que ya has visto:

```
$ ping ietf.org &
```

```
$ sudo tshark -i eno1 -f icmp
Capturing on eno1
192.168.0.37 -> 64.170.98.30 ICMP 98 Echo (ping) request  id=0x405d, seq=10/2560, ttl=64
64.170.98.30 -> 192.168.0.37 ICMP 98 Echo (ping) reply    id=0x405d, seq=10/2560, ttl=60
192.168.0.37 -> 64.170.98.30 ICMP 98 Echo (ping) request  id=0x405d, seq=11/2816, ttl=64
64.170.98.30 -> 192.168.0.37 ICMP 98 Echo (ping) reply    id=0x405d, seq=11/2816, ttl=60
192.168.0.37 -> 64.170.98.30 ICMP 98 Echo (ping) request  id=0x405d, seq=12/3072, ttl=64
64.170.98.30 -> 192.168.0.37 ICMP 98 Echo (ping) reply    id=0x405d, seq=12/3072, ttl=60
6 packets captured
```

Listado 18.1: `tshark` capturando tráfico ICMP

Como muchos otros programas de consola habituales en los sistemas POSIX, `tshark` acepta una gran variedad de opciones y argumentos que permiten modificar su comportamiento. En el ejemplo anterior, el significado de sus argumentos es el siguiente:

`-i eth0` Capturar tráfico de la interfaz `eth0`.
`-f icmp` Capturar solo mensajes ICMP y descartar el resto.
`-c 6` Capturar únicamente 6 paquetes y terminar.

La información de salida está organizada en columnas:

1. Instante de captura, relativo al primer mensaje[1].
2. Dirección IP origen.
3. Dirección IP destino.
4. Protocolo de la carga útil.
5. Tamaño total del mensaje.
6. Tipo de mensaje.
7. Identificador ICMP.
8. Número de secuencia ICMP.
9. TTL.

La información que se muestra depende del tipo de mensaje capturado. Por ejemplo, si capturas tráfico ARP verás algo muy distinto después de la columna de tamaño:

[1] el instante de captura se ha omitido en este ejemplo.

```
$ sudo tshark -i eno1 -f arp -N m
Capturing on eno1
SamsungE_db:d6:45 → Micro-St_30:69:7b ARP 60 Who has 192.168.1.121? Tell 192.168.1.193
Micro-St_30:69:7b → SamsungE_db:d6:45 ARP 42 192.168.1.121 is at 2c:f0:5d:30:69:7b
AzureWav_77:e6:c4 → Micro-St_30:69:7b ARP 60 Who has 192.168.1.121? Tell 192.168.1.181
Micro-St_30:69:7b → AzureWav_77:e6:c4 ARP 42 192.168.1.121 is at 2c:f0:5d:30:69:7b
```

Mediante los argumentos es posible afinar la captura con un alto grado de detalle. Además de capturar tráfico de una interfaz de red específica e imprimir un resumen como el anterior, `tshark` también puede cargar una captura desde un archivo (opción `-r`) y, por supuesto, también crear este tipo de archivos para su análisis posterior (opción `-w`).

18.1.1. Acceso privilegiado

Quizá hayas advertido el uso de `sudo` en los comandos anteriores. Esto se debe a que para capturar tráfico «crudo» de la interfaz de red se requieren privilegios especiales. Puedes utilizar `sudo` para ejecutar un programa con los privilegios de otro usuario (por defecto de `root`). Sin embargo, ejecutar cualquier programa con privilegios de administrador es siempre un riesgo de seguridad, de hecho, así lo advierte el propio `tshark` en su salida.

Existe una alternativa más conveniente para ejecutar `tshark` o programas similares. El núcleo Linux permite asignar «capacidades» (*capabilities*) específicas a un programa (un archivo binario). Las capacidades necesarias para capturar tráfico son `CAP_NET_ADMIN` y `CAP_NET_RAW`. Para aplicarlas al programa que realiza la captura de tráfico para `tshark` (`dumpcap`) basta ejecutar:

```
$ sudo setcap cap_net_raw,cap_net_admin=eip /usr/bin/dumpcap
```

Esta solución otorga esas capacidades a cualquier usuario que ejecute el programa `dumpcap`. Una alternativa más conservadora es otorgarlas sólo a los miembros de un grupo de usuarios específico (*p. ej.* grupo `wireshark`)[2].

18.1.2. Selección de la interfaz de captura

Como en el ejemplo anterior, se puede elegir de qué interfaz de red capturar (con la opción `-i`). Si no se especifica esta opción, `tshark` elegirá la interfaz que se esté utilizando para conectar a Internet. Recuerda que para obtener una lista de interfaces puedes utilizar el comando `ip link` o el veterano `ifconfig`. También el propio `tshark` puede mostrar un lista de interfaces disponibles, que además incluye algunas pseudo-interfaces que él mismo define y que en realidad no existen en el sistema.

[2]Consulta `http://wiki.wireshark.org/CaptureSetup/CapturePrivileges` para más detalles

```
$ tshark -D
1. eno1
2. any
3. lo (Loopback)
4. bluetooth-monitor
5. nflog
6. nfqueue
7. dbus-system
8. dbus-session
9. veth2728659
10. veth1362a83
11. br-6018fd38a646
12. br-7e03d0bfa152
13. ciscodump (Cisco remote capture)
14. randpkt (Random packet generator)
15. sdjournal (systemd Journal Export)
16. sshdump (SSH remote capture)
17. udpdump (UDP Listener remote capture)
```

Demos un repaso rápido a las interfaces que ha encontrado en este caso. Es probable que muchas de ellas también aparezcan cuando ejecutes este comando en tu propio sistema.

eno1
: La primera NIC Ethernet, y única en este caso.

any Interfaz virtual que permite capturar tráfico de todas las interfaces.

lo La interfaz *loopback*.

bluetooth-monitor
: Captura de tráfico Bluetooth.

nflog y nfqueue
: Captura de paquetes de Netfilter, el sistema de cortafuergos de Linux.

dbus-system y dbus-session
: Interfaz de captura de mensajes del bus de mensajes local (D-BUS) que utilizan muchos sistemas GNU/Linux.

veth-<id>y br-<id>
: Interfaces Ethernet virtuales y *bridge* respectivamente, comúnmente utilizadas por aplicaciones de virtualización como como Virtualbox, VMWare, KVM, docker, etc. para conexión a red de máquinas virtuales.

ciscodump
: Captura remota de Cisco.

spjournal
: Captura de mensajes del sistema `systemd`.

randpkt
: Generación de paquetes aleatorios.

sshdump
: Captura remota basado en un túnel SSH.

udpump
Captura remota de paquetes basado en un flujo UDP.

18.1.3. Limitando la captura

Si no se le indica lo contrario, `tshark` continuará capturando tráfico indefinidamente, ya sea mostrando la salida en consola o almacenándolo en un archivo.

Es posible limitar la captura de varias formas:

-c N
un número de paquetes (que cumplen los filtros).

-a duration:N
durante un número de segundos.

-a filesize:N
un tamaño de archivo de la captura resultante, indicando en KiB.

-a files:N
un número de archivos.

El siguiente comando captura todo el tráfico de red de la interfaz `wlan0` y lo almacena en un archivo llamado `wlan0.pcap` hasta un máximo de 1 MiB.

```
$ tshark -i wlan0 -w wlan0.pcap -a filesize:1024
Capturing on 'wlan0'
529
```

Durante el proceso, `tshark` mostrará en consola la cantidad actual de paquetes capturados.

18.1.4. Filtros de captura

Como hemos visto, el argumento `-f` sirve para especificar un filtro de captura. El formato de estos filtros constituye todo un lenguaje en sí mismo. La tabla 18.1 contiene unos cuantos ejemplos extraídos de `http://wiki.wireshark.org/CaptureFilters`. Puedes encontrar información detallada en la página de manual de `pcap-filter`.

18.1.5. Formato de salida

Por defecto el programa muestra una línea que resume cada mensaje, como en los ejemplos anteriores. Pero existen otras muchas posibilidades para obtener información detallada de cada campo. La opción `-V` muestra los valores de los campos de cada mensaje (por capa) en un formato de texto

Filtro	Significado
ip	tráfico IP
ip or arp	IP o ARP
tcp and not dst port 80	cualquier protocolo sobre TCP excepto el dirigido al puerto 80 (HTTP)
host 192.168.0.1	hacia o desde la IP indicada
net 192.168.0.0/24 net 192.168	hacia o desde la red indicada
src host 192.168.0.1 src net 192.168	procedente del host o red indicado
port 22	TCP o UDP hacia o desde el puerto 22
tcp dst port 22	hacia el puerto 22 TCP (SSH)
dst host 10.0.0.1 and port 443 and http	tráfico HTTP con destino al puerto 443 del host 10.0.0.1
not ether dst FF:FF:FF:FF:FF	tramas Ethernet no broadcast

CUADRO 18.1: `tshark`: ejemplos de filtros de captura

legible (Listado 18.2). Ya vimos una parte de esto en la captura 5.1 dónde introdujimos brevemente la información que ofrece. Aparte de los valores directos que se obtienen al diseccionar las cabeceras, `tshark` también muestra información derivada o calculada que no aparece en los mensajes, como por ejemplo, los intervalos de tiempo entre mensajes o el resultado de la comprobación de los checksums. Esa información aparece entre corchetes.

```
$ tshark -f icmp -c 1 -V
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
    Interface id: 0
    WTAP_ENCAP: 1
    Arrival Time: Feb 12, 2013 16:59:13.856520000 CET
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1360684753.856520000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:icmp:data]
Ethernet II, Src: Intel_ef:d4:96, Dst: Cisco_3a:c9:40
    Destination: Cisco_3a:c9:40 (00:64:40:3a:c9:40)
        Address: Cisco_3a:c9:40 (00:64:40:3a:c9:40)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: Intel_ef:d4:96 (c4:85:08:ef:d4:96)
        Address: Intel_ef:d4:96 (c4:85:08:ef:d4:96)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
```

```
Internet Protocol Version 4, Src: 120.12.17.214, Dst: 12.22.58.30
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not
    ↪  ECN-Capable Transport))
    Total Length: 84
    Identification: 0x0000 (0)
    Flags: 0x02 (Don't Fragment)
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x415c [correct]
    Source: 120.12.17.214 (120.12.17.214)
    Destination: 12.22.58.30 (12.22.58.30)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x1fb8 [correct]
    Identifier (BE): 27502 (0x6b6e)
    Identifier (LE): 28267 (0x6e6b)
    Sequence number (BE): 10557 (0x293d)
    Sequence number (LE): 15657 (0x3d29)
    Timestamp from icmp data: Feb 12, 2013 16:59:13.000000000 CET
    [Timestamp from icmp data (relative): 0.856520000 seconds]
    Data (48 bytes)

0000  8c 11 0d 00 00 00 00 00 10 11 12 13 14 15 16 17   ................
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
       Data: 8c110d0000000001011121314151617181 91a1b1c1d1e1f...
       [Length: 48]
```

LISTADO 18.2: Modo «verboso» de `tshark`

Normalmente esto es demasiada información y lo interesante es aislar específicamente los campos relacionados con la cuestión que te interese en cada caso. Para lograrlo se puede indicar el formato de salida por campos (`-T fields`) y la lista con los campos concretos, que pueden depender de las condiciones de filtrado. El ejemplo del Listado 18.3 filtra los mensajes TCP dirigidos al puerto 80 (presuntamente HTTP) y muestra únicamente las direcciones MAC e IP origen y destino.

```
$ tshark -f "tcp and dst port 80" -T fields -e eth.src -e ip.src -e eth.dst -e ip.dst
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    198.252.206.25
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    65.121.208.106
c4:85:08:ef:d4:96    120.12.17.214    08:20:12:3d:f4:32     120.12.27.170
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    198.252.206.25
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    173.194.34.196
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    173.194.34.196
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    173.194.34.196
```

```
c4:85:08:ef:d4:96    120.12.17.214    00:64:40:3a:c9:40    108.160.160.160
```

Listado 18.3: `tshark` permite elegir los campos a mostrar

Los nombres de los campos posibles son los mismos que para los «filtros de visualización» (que se tratan más adelante).

Además, es posible generar una representación de la captura en otros formatos, como PostScript XML, por si necesitaras hacer un tratamiento automatizado de los valores capturados.

18.1.6. Filtros de visualización

A partir de la captura obtenida (o cargada desde archivo) puedes aplicar un filtro que determina qué mensajes se muestran, es decir, un filtro de visualización (*display filter*) altera únicamente lo que el usuario puede ver en cada momento, pero a diferencia de los filtros de captura, estos ses pueden editar obteniendo un subconjunto diferente de entre los mensajes capturados. El formato de estos filtros es distinto al de los filtros de captura, y mucho más potente.

Como ejemplo, el siguiente listado muestra las peticiones HTTP que soliciten una URI que contenga la cadena `'favicon.ico'`, es decir, el icono que los navegadores asocian al guardar una página como favorito (*bookmark*).

```
$ tshark -R 'http.request.uri contains "favicon.ico"'
Capturing on wlan0
192.168.0.37 -> 93.184.220.111 HTTP 380 GET /i/favicon.ico?m=1317424629g HTTP/1.1
192.168.0.37 -> 217.148.71.165 HTTP 365 GET /favicon.ico HTTP/1.1
192.168.0.37 -> 23.37.161.157 HTTP 633 GET /favicon.ico HTTP/1.1
11 packets dropped
3 packets captured
```

El programa proporciona literalmente miles de filtros. Puedes consultarlos en la referencia en línea[3], en la página de manual para `wireshark-filter`[4] o con el comando `tshark -G fields`. Estos filtros están organizados jerárquicamente siendo el primer componente el nombre del protocolo al que aplican. En el ejemplo anterior, `http.request.uri` se refiere a la URI que aparece en la petición (GET) de los mensajes HTTP[5]. La tabla 18.2 muestra algunos ejemplos representativos similares a `http://wiki.wireshark.org/DisplayFilters`.

FiXme Fatal!

FiXme Fatal: : URL chuleta, copiar en el repo del libro? (ver licencia)

[3] `http://www.wireshark.org/docs/dfref/`

[4] `man wireshark-filter`

[5] `http://www.wireshark.org/docs/dfref/t/tcp.html`

Filtro	Significado
icmp.type == 8	Peticiones ICMP Echo
tcp.port == 25	Segmentos TCP hacia o desde el puerto 25
tcp.dstport == 25	Únicamente los dirigidos al puerto 25
arp or icmp	ICMP or ARP de cualquier tipo
ip.addr == 192.168.2.0/24	Generado/dirigido por/hacia un computador de la red indicada
!(ip.dst == 127.0.0.1)	No dirigido a la interfaz *loopback*

CUADRO 18.2: `tshark`: ejemplos de filtros de visualización

chuleta

Filtros de visualización[6]

Al igual que los filtros de captura, también es posible utilizar operadores lógicos y relacionales que permiten escribir expresiones muy elaboradas[7].

18.1.7. Estadísticas

`tshark` genera una gran variedad de estadísticas útiles[8]. En esta sección se incluyen solo algunas a modo de ejemplo.

proto,colinfo,*filter*,*field*

Añade columnas a la salida habitual. Por ejemplo:

```
$ tshark -z proto,colinfo,tcp.srcport,tcp.srcport
  5.508997 108.160.160.160 -> 192.168.0.37 HTTP 245 HTTP/1.1 200 OK
          (text/plain)  tcp.srcport == 80
  8.928316 173.194.78.125 -> 192.168.0.37 TCP 471
          [TCP segment of a reassembled PDU]  tcp.srcport == 5222
 17.660298 173.194.78.125 -> 192.168.0.37 TCP 199
          [TCP segment of a reassembled PDU]  tcp.srcport == 5222
```

icmp,srt[,*filter*]

Calcula las estadísticas SRT (Service Response Time) del tráfico ICMP Echo de forma similar al comando `ping`. Un ejemplo:

```
$ tshark -z icmp,srt
[...]
10024 packets captured
ICMP Service Response Time (SRT) Statistics (all times in ms):
Filter: <none>
Requests  Replies   Lost      % Loss
1784      1783      1           0,1%
Minimum   Maximum   Mean      Median    SDeviation     Min Frame Max Frame
```

[7] `http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html`

[8] Puedes ver todas las estadísticas disponibles con `tshark-z help`

```
55,228    818,109   62,742    56,894    37,105         7831      5373
```

io,phs[,*filter*]

Contabiliza mensajes y bytes de todos los mensajes (conforme al filtro) desglosando los resultados según su encapsulación.

```
$ tshark -z io,phs -q -c 100
Capturing on wlan0
100 packets captured
Protocol Hierarchy Statistics
Filter:
eth                                      frames:100 bytes:18503
  ip                                     frames:98 bytes:18289
    icmp                                 frames:32 bytes:3136
    udp                                  frames:34 bytes:3760
      dns                                frames:32 bytes:3408
      db-lsp-disc                        frames:2 bytes:352
    tcp                                  frames:32 bytes:11393
      ssl                                frames:14 bytes:10189
        tcp.segments                     frames:2 bytes:1157
  llc                                    frames:2 bytes:214
    data                                 frames:2 bytes:214
```

io,stat,*interval*,*filter*,*filter*

Contabiliza mensajes y bytes en intervalos del tiempo especificado. Permite indicar una cantidad arbitraria de filtros que serán representados como columnas.

```
$ LANG=C tshark -nzio,stat,0.5,udp,"tcp.dstport==80",
[...]
794 packets captured
================================================================
| IO Statistics                                                |
|                                                              |
| Interval size: 0.5 secs                                      |
| Col 1: udp                                                   |
|     2: tcp.dstport==80                                       |
|     3: Frames and bytes                                      |
|--------------------------------------------------------------|
|            |1              |2              |3               |
| Interval   | Frames | Bytes | Frames | Bytes | Frames | Bytes |
|--------------------------------------------------------------|
| 0.0 <> 0.5 |      2 |   211 |      2 |   132 |     30 |  9012 |
| 0.5 <> 1.0 |      0 |     0 |      9 |   498 |     18 |  1044 |
| 1.0 <> 1.5 |      2 |   211 |      0 |     0 |      4 |   407 |
| 1.5 <> 2.0 |      0 |     0 |      7 |   378 |     14 |   798 |
| 2.0 <> 2.5 |      6 |   806 |     11 |  1597 |     24 |  3376 |
| 2.5 <> 3.0 |     17 |  1780 |     61 | 11138 |    133 | 42969 |
================================================================
```

io,stat,*interval*,*func*

Aplica funciones de agregación (`COUNT`, `SUM`, `MIN`, `MAX`, `AVG` y `LOAD`) que se pueden aplicar a campos cualesquiera para generar columnas en la tabla de resultados. El siguiente ejemplo cuenta el número de paquetes IP, y calcula la media y la suma de los tamaños de esos paquetes en intervalos de 2 segundos.

```
$ LANG=C tshark -nzio,stat,2,"COUNT(ip)ip","AVG(ip.len)ip.len","SUM(ip.len)ip.len"
[...]
803 packets captured
==================================
| IO Statistics                  |
|                                |
| Interval size: 2 secs          |
| Col 1: COUNT(ip)ip             |
|     2: AVG(ip.len)ip.len       |
|     3: SUM(ip.len)ip.len       |
|--------------------------------|
|          |1      |2    |3       |
| Interval | COUNT | AVG |   SUM  |
|--------------------------------|
|  0 <>  2 |     7 |  61 |    432 |
|  2 <>  4 |     0 |   0 |      0 |
|  4 <>  6 |   107 | 115 |  12354 |
|  6 <>  8 |   486 | 267 | 130039 |
|  8 <> 10 |   126 | 497 |  62628 |
| 10 <> 12 |    20 |  45 |    904 |
| 12 <> 14 |    38 |  40 |   1544 |
==================================
```

follow,*proto*,*mode*,*filter*[,*range*]

Muestra el contenido de un flujo de mensajes entre dos nodos, es decir, concatena la carga útil de los segmentos sucesivos que corresponden a la misma sesión o conexión. De este modo, si por ejemplo se está transmitiendo un archivo podría recuperarse a partir de la captura. Los protocolos soportados son TCP y UDP. Y puede hacer el volcado en tres modos diferentes: ascii, hex y raw.

```
$ tshark -z follow,tcp,hex,192.168.2.12:47147,129.42.58.216:80
```

conv,*type*[,*filter*]

Muestra las «conversaciones» o flujos, es decir, nodos que intercambian mensajes entre sí de forma recurrente. El parámetro *type* puede ser uno de: eth, fc, fddi, ip, ipv6, ipx, tcp, tr, udp.

```
$ tshark -z conv,tcp
TCP Conversations
```

```
Filter:<No Filter>                          |  <-  |  ->   | Total  | Start   | Durat |
                                            | F  B | F  B  | F   B  |         |       |
192.168.0.37:38216 <-> 73.233.104.123:80    1  74  2  140  3   214  0,002912  0,2126
192.168.0.37:58949 <-> 92.184.220.111:80    1  66  2  128  3   194  0,002801  0,1134
192.168.0.37:58948 <-> 92.184.220.111:80    1  66  2  128  3   194  0,002754  0,1118
192.168.0.37:58947 <-> 92.184.220.111:80    1  66  2  128  3   194  0,002704  0,1027
192.168.0.37:58946 <-> 92.184.220.111:80    1  66  2  128  3   194  0,002660  0,1015
192.168.0.37:56569 <-> 92.100.127.144:80    1  74  2  140  3   214  0,002398  0,0727
192.168.0.37:38838 <-> 95.172.94.11:80      1  62  2  128  3   190  0,002319  0,0952
192.168.0.37:38837 <-> 95.172.94.11:80      1  62  2  128  3   190  0,002270  0,0935
```

18.2. wireshark

`wireshark` es una herramienta gráfica que amplía las posibilidades de `tshark`. Permite visualizar el tráfico capturado de un modo más flexible, potente e intuitivo, y además ofrece herramientas de análisis más sofisticadas.

18.2.1. Interfaz gráfica

Al arrancar `wireshark` aparece una ventana (Figura 18.1) que muestra la lista de interfaces disponibles para captura (las mismas que hemos visto con `tshark` en §18.1.2). Desde aquí puedes especificar un filtro de captura (opcional), seleccionar una interfaz y pulsar el botón de captura:  .

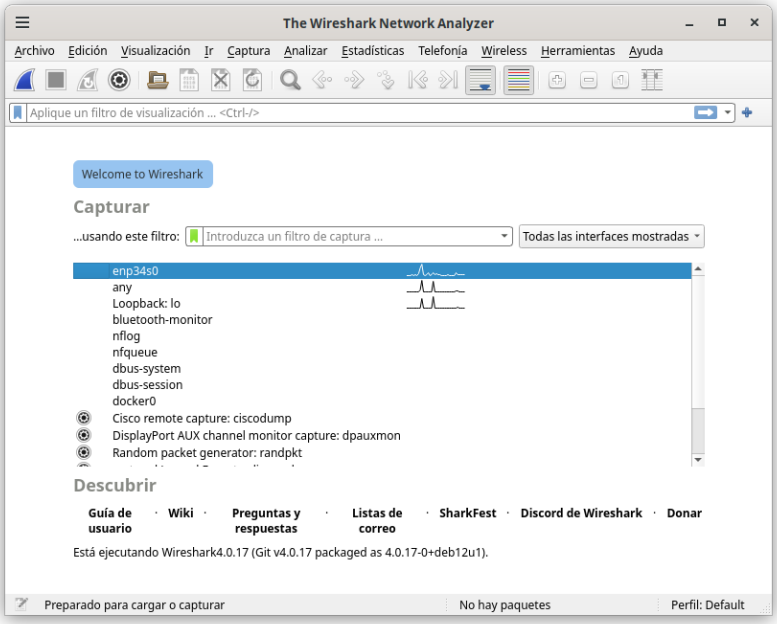

FIGURA 18.1: `wireshark`: interfaz inicial

Inmediatamente la ventana muestra 3 paneles:

1. El panel superior muestra la lista de mensajes capturados, con una línea por mensaje, que se va actualizando en tiempo real conforme van llegando nuevos mensajes. Se muestran por defectos las columnas No., Time, Source, Destination, Protocol, Length e Info.

2. Abajo a la izquierda, aparece un árbol expandible que incluye todos los detalles de los campos de cada una de las cabeceras de los distintos protocolos encapsulados en cada mensaje.

3. Abajo a la derecha, el contenido completo del mensaje en formato hexadecimal en un formato prácticamente idéntico al que hemos visto con `tshark` al indicar la opción `-V` en § 5.1.

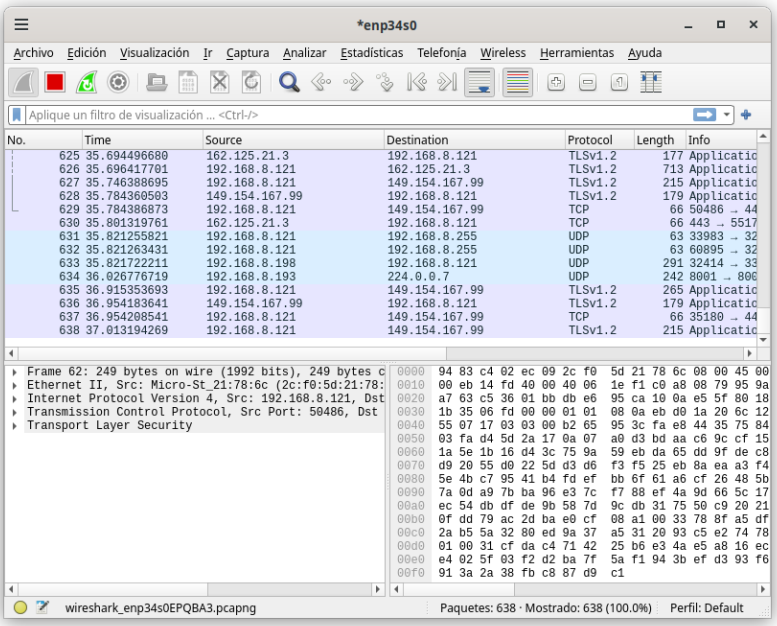

FIGURA 18.2: `wireshark`: interfaz de captura por defecto

Esta vista por defecto no parece la más útil. Puedes cambiar entre distintas disposiciones en el menú Edición→Preferencias→Apariencia→Diseño y elegir el diseño de 3 filas (el primero) que corresponde con lista de mensajes, detalles y contenido (bytes) tal como muestra la Figura 18.3. De ese modo la interfaz se verá como en la Figura 18.4.

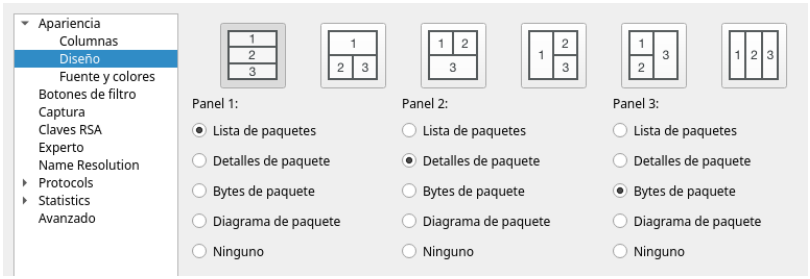

FIGURA 18.3: `wireshark`: configuración de paneles

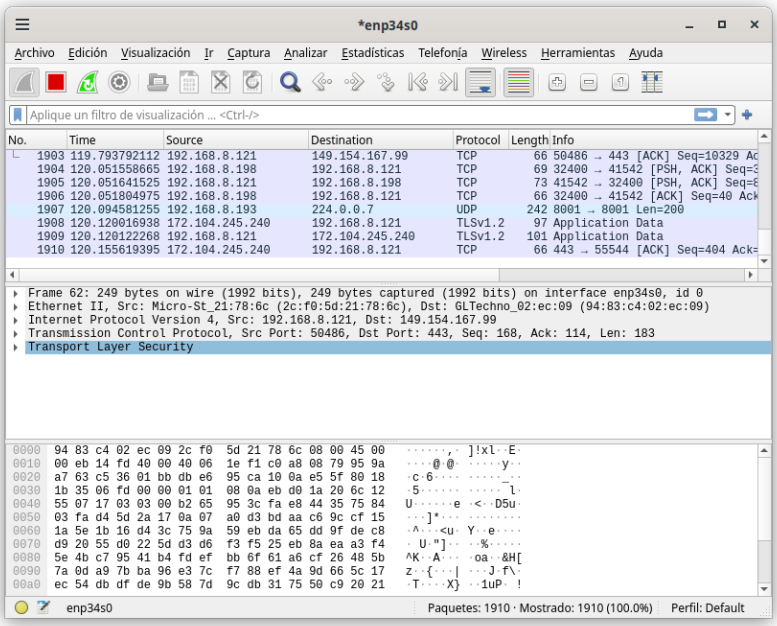

FIGURA 18.4: `wireshark`: interfaz con las 3 filas: lista, detalle y bytes

Habrás comprobado que el programa sigue capturando tráfico indefinidamente, hasta que pulses el botón de parada: ■.

18.2.2. Captura y filtrado

Como ejemplo, vamos a repetir la captura que hicimos con `tshark` en § 5.9.2 para ilustrar una conexión TCP. En la ventana inicial escribe `tcp port 2000` como filtro de captura, selecciona la interfaz *loopback* y lanza la captura (Figura 18.5).

Comprobarás que no aparece nada... es normal, por el momento no hay tráfico que coincida con el filtro de captura. Vamos a generarlo recreando la misma sesión TCP con `ncat`, es decir, en una consola ejecuta el servidor con `ncat -l -p 2000` y en otra el cliente con `echo hola | ncat 127.0.0.1 2000`. Inmediatamente verás que aparecen 8 mensajes en el panel superior, como en la Figura 18.6.

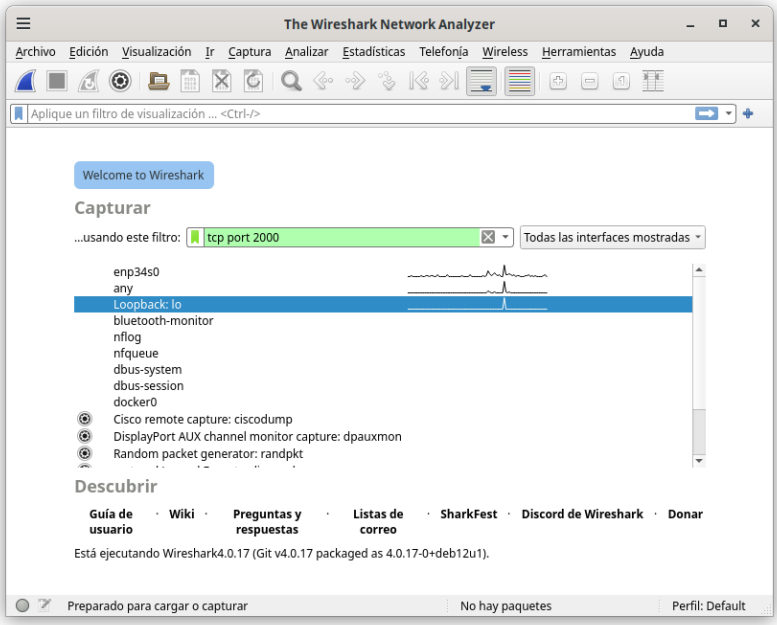

FIGURA 18.5: `wireshark`: estableciendo una captura en la interfaz *loopback* para el puerto TCP 2000

El primero de los mensajes está seleccionado y eso permite explorar todos sus campos en el panel central. En la figura, la parte correspondiente a la cabecera TCP se muestra expandida y aparece el valor de cada campo. En esta segunda sección, el campo seleccionado es «flags» con el valor `0x002`, que corresponde a la activación del flag SYN. Efectivamente se trata del segmento de establecimiento de conexión enviado por el cliente. A su vez, esto provoca que los bytes del mensaje que corresponden a ese campo de la cabecera aparezcan resaltados en el panel inferior.

Aquí puedes establecer un filtro de visualización para, por ejemplo, quedarte únicamente con los segmentos TCP que tienen carga útil. Ese filtro será `tcp.len >0` y como puedes ver en Figura 18.7 se escribe en la barra que queda resaltada con fondo verde (ese color confirma que el filtro es correcto).

El único mensaje que cumple el filtro es el que lleva la cadena «hola\n», y lo puedes ver resaltado en los paneles central e inferior.

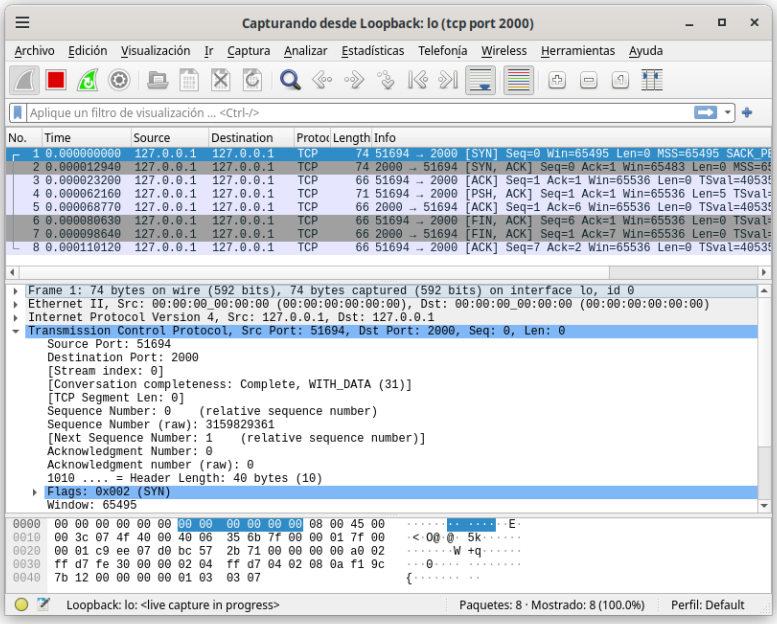

FIGURA 18.6: `wireshark`: captura de una conexión TCP simple

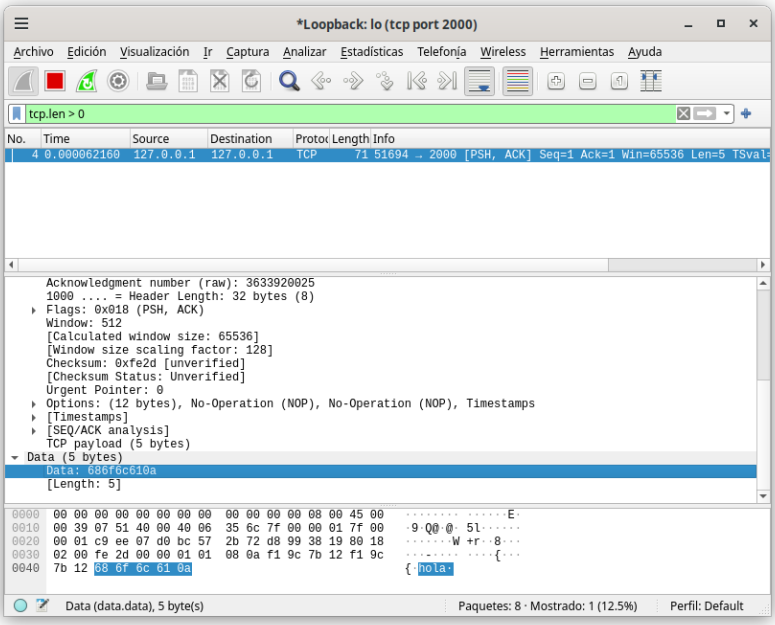

FIGURA 18.7: `wireshark`: filtro de visualización (mensajes con carga útil)

18.3. Captura de flujos

`wireshark` ofrece la posibilidad de capturar la carga útil de una conexión TCP o de un flujo UDP en uno o en ambos sentidos. Esto significa que, por ejemplo, es posible recuperar y reconstruir un archivo que se está transmitiendo a través de la red, asumiendo, por supuesto, que has sido capaz de capturar todos los mensajes.

Para empezar debes utilizar `tshark` para capturar el tráfico dirigido al puerto TCP 2000 y lo puedes guardar en un archivo. Después analizaremos ese archivo para recuperar la carga útil de la conexión completa tanto con `wireshark` como con herramientas de línea de comandos. Por supuesto, con `tshark` por ejemplo, aunque por supuesto lo puedes hacer igualmente con `wireshark`.

```
$ tshark -i lo -f "tcp port 2000" -w captura.pcap
```

Ahora envía un archivo (un `.mp3` concretamente, aunque sirve cualquier cosa) a través de una conexión TCP que puedes conseguir con un servidor `ncat` a la escucha en el puerto 2000 y un cliente (también `ncat`) con el que enviar el archivo:

Cliente

```
$ cat original.mp3 | ncat localhost 2000
```

Servidor

```
$ ncat -l -p 2000 > recibido.mp3
```

Figura 18.8: Transfiriendo un `.mp3` con `ncat`

Una vez terminado el envío, puedes parar la captura con `Ctrl+C`. Y la puedes cargar en `wireshark` con la opción Archivo→Abrir o desde consola simplemente con:

```
$ wireshark captura.pcap
```

Ahora verás la lista de todos los mensajes que forman la captura. Puedes abrir el menú contextual pulsando con el botón derecho del ratón en cualquier de los mensajes. Elige Seguir→TCP Stream. Aparecerá una nueva ventana como la de la Figura 18.9.

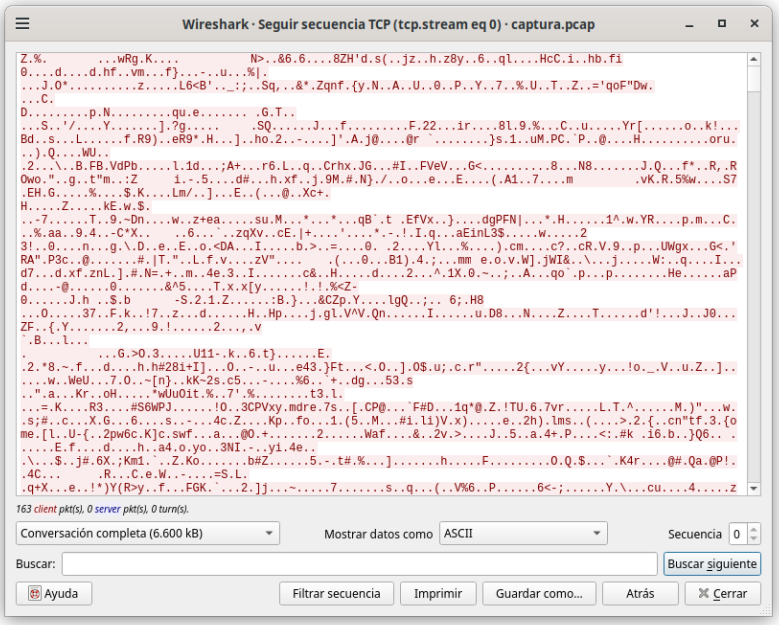

Figura 18.9: `wireshark`: reconstrucción de un flujo TCP

En los controles que aparecen en la parte inferior elige «Mostrar como: raw» y pulsa el botón «Guardar como...». Guarda el archivo con el nombre captura.mp3».

Por supuesto, puedes abrirlo con cualquier reproductor multimedia, pero también puedes comprobar que el archivo recuperado es idéntico al original utilizando el comando `diff` o calculando el MD5 de ambos archivos.

```
$ md5sum *
c2b91faddfec1d01c4f81a8a6d34d537  original.mp3
c2b91faddfec1d01c4f81a8a6d34d537  captura.mp3
```

Puedes conseguir lo mismo desde línea de comandos con:

```
$ tcpflow -r captura.pcap
```

El programa genera un archivo con la carga útil para cada uno de los flujos y sentidos contenidos en la captura. En este caso solo crea un archivo, ya que el sentido servidor a cliente de la conexión TCP no ha transportado ningún dato. El archivo se llama en este caso `127.000.000.001.58008-127.000.000.001.02000` que está formado por las direcciones IP y puertos de origen y destino de la conexión. Igual que antes, puedes comprobar que el archivo es idéntico al original.

```
$ md5sum *
c2b91faddfec1d01c4f81a8a6d34d537  original.mp3
c2b91faddfec1d01c4f81a8a6d34d537  captura.mp3
c2b91faddfec1d01c4f81a8a6d34d537  127.000.000.001.58008-127.000.000.001.02000
```

Puedes ver que resulta sorprendente sencillo recuperar un archivo completo que se transfiere por la red si se dan las condiciones y se dispone de las herramientas adecuadas. Es fácil entender las consecuencias que esto tiene para la privacidad. Por supuesto, si el archivo se transmite sobre un canal cifrado (como TLS) esto no será posible, o al menos, no tan sencillo.

Y ¿qué más?

En este capítulo hemos dado un vistazo a algunas de las posibilidades de captura, visualización, filtrado, análisis y generación de estadísticas que ofrecen `tshark` y `wireshark`. Echando un vistazo a su documentación es fácil percatarse de que se trata de herramientas extraordinariamente potentes, que se utilizan habitualmente en ámbitos profesionales de la seguridad informática, la administración de redes, el desarrollo de software de comunicaciones, etc. A partir de aquí, invitamos al lector a explorar su gran potencial. Hay mucho que descubrir y aprender.