

MÁSTER EN REVERSING, ANÁLISIS DE MALWARE Y BUG HUNTING

MÁSTER EN
*ANÁLISIS DE MALWARE Y
REVERSING*

María Sonia Salido Fernández

Módulo 5 - Tarea 2



Campus Internacional
CIBERSEGURIDAD



UCAM
UNIVERSIDAD
CATÓLICA DE MURCIA

- **Entendiendo que pide el ejercicio**
- **1. Análisis del protocolo**
 - **Filtración el tráfico específico**
 - **Localización las conversaciones reales de la app. Sesiones TCP**
- **2. Información intercambiada entre el cliente y el servidor**
 - **Listar directorio - Paquete 550**
 - **Fichero hello.txt - Paquete 560**
 - **Fichero lena.gif - Paquete 571**

Entendiendo que pide el ejercicio

Analizar el tráfico incluido en el fichero actividad2.pcap.

Esa captura de tráfico se ha realizado en la misma máquina (192.168.1.33) donde se ejecuta el servidor de la aplicación a analizar y que corre en el puerto TCP/7000 (que no es el puerto estándar de ese protocolo). El fichero de captura también incluye otro tipo de tráfico, que no está relacionado con la aplicación, por lo que se deberían utilizar los filtros de visualización de Wireshark para encontrar el tráfico de la aplicación.

1. Análisis del protocolo

El objetivo de la práctica es:

- Estudiar el protocolo empleado por la aplicación.
- Identificar el tipo de mensajes intercambiados por el cliente y el servidor.
- Intentar interpretar el contenido y significado de los mismos.
- Obtener la información intercambiada.
- Intentar identificar el protocolo exacto que está siendo empleado por la aplicación.
- Incluir capturas de pantalla de Wireshark:
 - Especificación del protocolo de la aplicación
 - La información intercambiada entre el cliente y el servidor.

Para resolver este ejercicio de análisis de tráfico con Wireshark, necesitamos filtrar el ruido y centrarnos específicamente en la actividad del puerto 7000. Por ello debemos eliminar el tráfico de fondo como TLS e ICMP.

Filtración el tráfico específico

Dado que el enunciado indica que la aplicación corre en el puerto TCP 7000, lo primero es limpiar la vista. En wireshark, aplicamos el filtro: `tcp.port == 7000`:

No.	Time	Source	Destination	Protocol	Length	Info
547	23.658353	192.168.1.110	servidor	TCP	66	62897 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
548	23.658377	servidor	192.168.1.110	TCP	66	7000 → 62897 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
549	23.658573	192.168.1.110	servidor	TCP	54	62897 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
550	23.659351	192.168.1.110	servidor	TCP	56	62897 → 7000 [PSH, ACK] Seq=1 Ack=1 Win=2102272 Len=2
551	23.659418	servidor	192.168.1.110	TCP	54	7000 → 62897 [ACK] Seq=1 Ack=3 Win=64256 Len=0
552	23.662103	servidor	192.168.1.110	TCP	111	7000 → 62897 [PSH, ACK] Seq=1 Ack=3 Win=64256 Len=57
553	23.662161	servidor	192.168.1.110	TCP	220	7000 → 62897 [FIN, PSH, ACK] Seq=58 Ack=3 Win=64256 Len=166

```

554 23.662373 192.168.1.110 servidor      TCP 54 62897 → 7000 [ACK]
Seq=3 Ack=225 Win=2102016 Len=0
555 23.662554 192.168.1.110 servidor      TCP 54 62897 → 7000 [FIN, ACK]
Seq=3 Ack=225 Win=2102016 Len=0
556 23.662565 servidor      192.168.1.110      TCP 54 7000 → 62897 [ACK]
Seq=225 Ack=4 Win=64256 Len=0
    ...
    ...

```

En el primer paquete que aparece tras aplicar el filtro (547), vamos hacer un seguimiento con TCP stream:
Hacemos click derecho en el paquete 547 y seleccionamos Follow (Seguir) → TCP Stream.

No.	Time	Source	Destination	Protocol	Length	Info
547	23.658353	192.168.1.110	servidor	TCP	66	62897 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
548	23.658377	servidor	192.168.1.110	TCP	66	7000 → 62897 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
549	23.658573	192.168.1.110	servidor	TCP	54	62897 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
550	23.659351	192.168.1.110	servidor			
551	23.659418	servidor				
552	23.662103	servidor				
553	23.662161	servidor				
554	23.662373	192.168.1.110				
555	23.662554	192.168.1.110				
556	23.662565	servidor				

```

iWelcome to the 192.168.1.33:7000 server!    fake    (NULL)  0
i  fake    (NULL)  0
0Hello  /hello.txt  192.168.1.33    7000      +
1RFCs   /dir/    192.168.1.33    7000      +
gLena   /lena.gif  192.168.1.33    7000      +
1Quux  Mega Server /  142.4.200.132    7000

```

donde:

- Vemos palabras legibles como "Welcome to the...", "Hello", "Mega Server". Esto parece que es un protocolo basado en texto (ASCII), lo cual descarta protocolos complejos cifrados o puramente binarios.
- Tambien parece una comunicación en la que se piden recursos, como hello.txt, lena.gif

Localización las conversaciones reales de la app. Sesiones TCP

Aunque filtrémos por 7000, es buena práctica confirmar cuántas conexiones hay y quién habla con quién:

- Vamos a Statistics → Conversations → TCP → Ordenamos por Port o usamos el buscador para encontrar 7000.
- Apuntamos:
 - IP/puerto del cliente
 - IP/puerto del servidor (192.168.1.33:7000)
 - Número de bytes transferidos en cada sentido (te ayuda a identificar la sesión “importante”).



Dirección A, Puerto A, Dirección B, Puerto B, Paquetes, Bytes, Stream ID, Paquetes totales, Porcentaje filtrado, Packets A → B, Bytes A → B, Packets B → A, Bytes B → A, Inicio rel, Duración, Bits/s A → B, Bits/s B → A, Flows
 "192.168.1.110", 62897, "192.168.1.33", 7000, 10, 789, 6, 10, 100, 5, 284, 5, 505, 23.658353, 0.00421200000002547, "", "", 3
 "192.168.1.110", 62900, "192.168.1.33", 7000, 10, 589, 7, 10, 100, 5, 294, 5, 295, 25.852265, 0.007728000000000179, 304347, 305383, 2
 "192.168.1.110", 62910, "192.168.1.33", 7000, 82, 231798, 8, 82, 100, 66, 3587, 16, 228
 31.351551, 2.8117599999999996, 10205, 649304, 14
 "192.168.1.110", 62949, "192.168.1.33", 7000, 10, 618, 10, 10, 100, 5, 289, 5, 329, 52.655474, 0.0057520000000109, 401947, 457579, 2

Esta vista de Conversaciones TCP es fundamental porque nos aporta una visión global de cómo interactuó el cliente con la aplicación. Al filtrar por el puerto 7000, vemos que se establecieron 4 flujos o sesiones independientes entre el cliente (192.168.1.110) y el servidor (192.168.1.33).

Y los volúmenes son muy diferentes:

- Stream 6: 10 paquetes / 789 bytes → muy corto
- Stream 7: 10 paquetes / 589 bytes → muy corto
- Stream 8: 82 paquetes / 231,798 bytes → con diferencia, el principal
- Stream 10: 10 paquetes / 618 bytes → muy corto

Además, en el Stream 8:

- A → B (cliente → servidor): 3,587 bytes
- B → A (servidor → cliente): 228,211 bytes

Esto encaja con un patrón típico: el cliente hace una petición pequeña y el servidor responde con un contenido grande, que puede ser una descarga, listado, datos, o un archivo. Es por esto que vamos a apostar por este stream para seguir investigando `tcp.stream == 8`.

Aplicamos el filtro: `tcp.stream == 8`

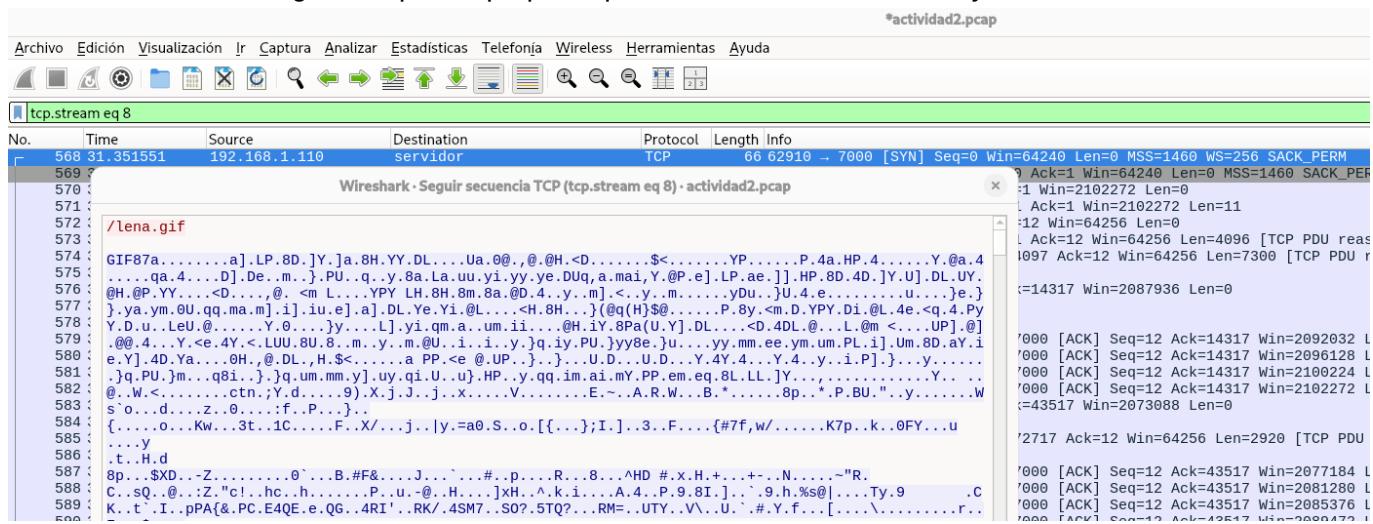
No.	Time	Source	Destination	Protocol	Length	Info
568	31.351551	192.168.1.110	servidor	TCP	66	62910 → 7000 [SYN]
		Seq=0 Win=64240 Len=0	MSS=1460 WS=256 SACK_PERM			
569	31.351573	servidor	192.168.1.110	TCP	66	7000 → 62910 [SYN, ACK]
		Seq=0 Ack=1 Win=64240 Len=0	MSS=1460 SACK_PERM WS=128			
570	31.351799	192.168.1.110	servidor	TCP	54	62910 → 7000 [ACK]
		Seq=1 Ack=1 Win=2102272 Len=0				
571	31.352463	192.168.1.110	servidor	TCP	65	62910 → 7000 [PSH, ACK]
		Seq=1 Ack=1 Win=2102272 Len=11				
572	31.352469	servidor	192.168.1.110	TCP	54	7000 → 62910 [ACK]
		Seq=1 Ack=12 Win=64256 Len=0				
573	31.355524	servidor	192.168.1.110	TCP	4150	7000 → 62910 [PSH, ACK]
		Seq=1 Ack=12 Win=64256 Len=4096	[TCP PDU reassembled in 575]			
574	31.355543	servidor	192.168.1.110	TCP	7354	7000 → 62910 [PSH, ACK]
		Seq=4097 Ack=12 Win=64256 Len=7300	[TCP PDU reassembled in 575]			
575	31.355556	servidor	192.168.1.110	Gryphon	2974	
576	31.355954	192.168.1.110	servidor	TCP	54	62910 → 7000 [ACK]
		Seq=12 Ack=14317 Win=2087936 Len=0				
577	31.355963	servidor	192.168.1.110	Gryphon	14654	- Invalid --
		Invalid -				
578	31.355967	servidor	192.168.1.110	Gryphon	14654	- Invalid -
579	31.355954	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2092032 Len=0				
580	31.355954	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2096128 Len=0				
581	31.355954	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2100224 Len=0				
582	31.355954	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2102272 Len=0				
583	31.356253	192.168.1.110	servidor	TCP	54	62910 → 7000 [ACK]
		Seq=12 Ack=43517 Win=2073088 Len=0				
584	31.356259	servidor	192.168.1.110	Gryphon	29254	- Invalid -
585	31.356263	servidor	192.168.1.110	TCP	2974	7000 → 62910 [PSH, ACK]
		Seq=72717 Ack=12 Win=64256 Len=2920	[TCP PDU reassembled in 586]			
586	31.356265	servidor	192.168.1.110	Gryphon	26334	- Invalid --
		Invalid -				
587	31.356253	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2077184 Len=0				
588	31.356253	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2081280 Len=0				
589	31.356253	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2085376 Len=0				
590	31.356253	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2089472 Len=0				
591	31.356253	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2093568 Len=0				
592	31.356253	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2097664 Len=0				
593	31.356274	192.168.1.110	servidor	TCP	54	[TCP Window Update]
		62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2101760 Len=0				
594	31.356563	192.168.1.110	servidor	TCP	54	62910 → 7000 [ACK]

```

Seq=12 Ack=101917 Win=2043392 Len=0
595 31.356572     servidor      192.168.1.110  Gryphon 35094 - Invalid --
Invalid -
596 31.356563     192.168.1.110  servidor      TCP 54 [TCP Window Update]
62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2048000 Len=0
597 31.356563     192.168.1.110  servidor      TCP 54 [TCP Window Update]
62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2052096 Len=0
598 31.356563     192.168.1.110  servidor      TCP 54 [TCP Window Update]
62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2056192 Len=0
599 31.356563     192.168.1.110  servidor      TCP 54 [TCP Window Update]
62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2060288 Len=0
.....
.....

```

Reconstruimos el diálogo en el primer paquete que muestra este filtro → Analyze → Follow → TCP Stream



donde:

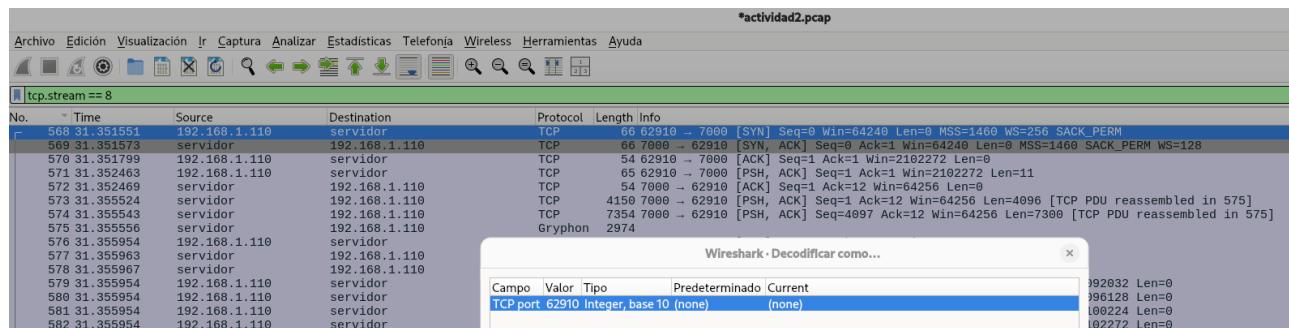
- El cliente envía la cadena /lena.gif (en rojo).
- El servidor devuelve un contenido que empieza por GIF87a, que es la firma ("magic bytes") de un fichero GIF.
- La aplicación está sirviendo un recurso llamado lena.gif, y el cliente lo solicita indicando la ruta. Esto es muy compatible con HTTP (GET de un objeto), pero también podría ser un protocolo propio minimalista, en el que el cliente manda la ruta y el servidor devuelve bytes "a pelo".

Podemos deducir un patrón: el cliente pide un recurso por "ruta/selector" y el servidor devuelve el fichero en bruto.

Vamos a tirar por hacer un “Decode As... COMO HTTP” de un paquete que ofrezca el puerto 7000 a ver que nos dice: De los dos primeros paquetes de este stream 8:

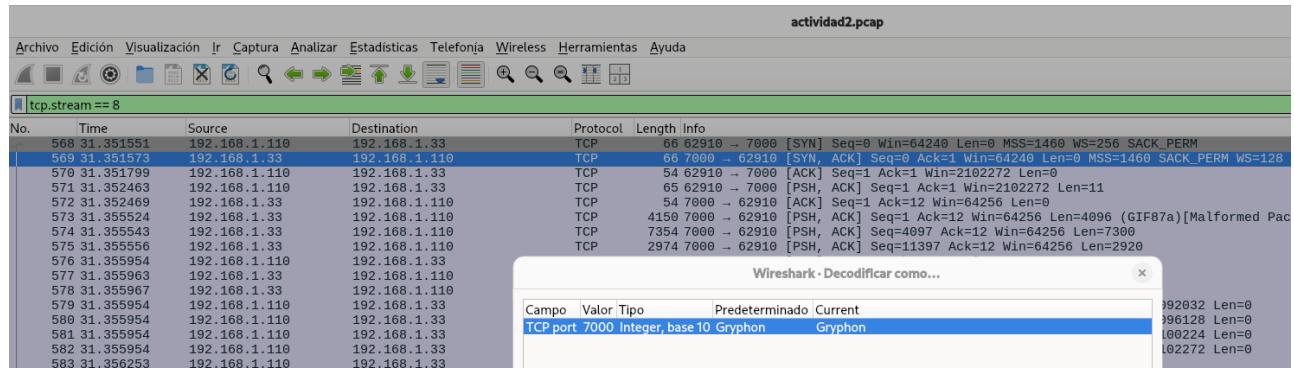
- Paquete 568: 62910 → 7000 [SYN]

Es el SYN del cliente, donde el puerto “destino” es 7000 pero el paquete se origina desde el puerto 62910.



- Paquete 569: 7000 → 62910 [SYN, ACK]

Es el SYN/ACK del servidor, y aquí el paquete sale desde el puerto 7000. Al abrir Decode As desde este paquete, es mucho más probable que Wireshark te muestre TCP port = 7000, que es lo que necesitas para decirle: “trata el puerto 7000 como HTTP”.



donde:

- “Protocol”, Wireshark etiqueta el paquete 575 como Gryphon.** Sin embargo, si investigamos este protocolo Gryphon, es un protocolo de red industrial utilizado principalmente en la automoción (para diagnósticos de vehículos y buses CAN). Es un protocolo binario.

Aunque Wireshark intenta interpretarlo como Gryphon debido al puerto 7000, lo que vimos en la captura donde veíamos palabras legibles en texto plano que decían "Welcome to the... server!", "hello.txt", "RFCs", "lena.gif" indica un protocolo de texto. Luego el protocolo que buscamos no es ese Gryphon.

Como parece que es un protocolo de texto, vamos a intentar hacer un decode del paquete 569, obligando a usar HTTP:

tcp.stream == 8

No.	Time	Source	Destination	Protocol	Length	Info
568	31.351551	192.168.1.110	192.168.1.33	TCP	66	62910 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
569	31.351573	192.168.1.33	192.168.1.110	TCP	66	7000 → 62910 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
570	31.351799	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
571	31.352463	192.168.1.110	192.168.1.33	TCP	65	62910 → 7000 [PSH, ACK] Seq=1 Ack=1 Win=2102272 Len=11
572	31.352469	192.168.1.33	192.168.1.110	TCP	54	7000 → 62910 [ACK] Seq=1 Ack=12 Win=64256 Len=0
573	31.355524	192.168.1.33	192.168.1.110	TCP	4150	7000 → 62910 [PSH, ACK] Seq=1 Ack=12 Win=64256 Len=4096 (GIF87a)[Malformed Packet]
574	31.355543	192.168.1.33	192.168.1.110	TCP	7354	7000 → 62910 [PSH, ACK] Seq=4097 Ack=12 Win=64256 Len=7300
575	31.355556	192.168.1.33	192.168.1.110	TCP	2974	7000 → 62910 [PSH, ACK] Seq=11397 Ack=12 Win=64256 Len=2920
576	31.355954	192.168.1.110	192.168.1.33			
577	31.355963	192.168.1.33	192.168.1.110			
578	31.355967	192.168.1.33	192.168.1.110			
579	31.355954	192.168.1.110	192.168.1.33			
580	31.355954	192.168.1.110	192.168.1.33			
581	31.355954	192.168.1.110	192.168.1.33			
582	31.355954	192.168.1.110	192.168.1.33			

Pero no cambia nada:

actividad2.pcap							
Archivo	Edición	Visualización	Ir	Captura	Analizar	Estadísticas	Telefonía
				<img alt			

No usa el protocolo http. Veamos que responde google:

protocolo que puede tener esa respuesta: **Welcome to the 192.168.1.33:7000 server! fake (NULL) 0 i fake (NULL) 0 OHello /hello.txt 192.168.1.33 7000 +1RFCs /dir/ 192.168.1.33 7000 + gLena /lena.gif 192.168.1.33 7000 +1Quux Mega Server / 142.4.200.132 7000**

[Mostrar resultados originales](#)

◆ Visión general creada por IA

Esa respuesta es típica de un servidor FTP (File Transfer Protocol) que lista directorios y archivos, usando comandos como LIST o NLST, respondiendo con códigos numéricos (como '0' o '1' para indicar tipo de archivo) y nombres/rutas, mostrando un listado de recursos disponibles en un servidor, posiblemente un servidor de desarrollo o de pruebas, identificándose como "fake" y listando archivos/directorios con sus rutas y metadatos.

Análisis de la respuesta:

- **Welcome to the 192.168.1.33:7000 server! fake (NULL) 0 i fake (NULL) 0 OHello /hello.txt 192.168.1.33 7000 +1RFCs /dir/ 192.168.1.33 7000 + gLena /lena.gif 192.168.1.33 7000 +1Quux Mega Server / 142.4.200.132 7000**: Mensaje de bienvenida del servidor FTP, que indica la IP y puerto (192.168.1.33:7000), su nombre ("fake"), y un posible indicador de estado o tipo.

Vamos a probar con FTP. Hacemos un decode de ese paquete 569 con el protocolo FTP:

donde:

- FTP (control) usa comandos tipo: USER, PASS, SYST, PWD, TYPE, RETR, PASV, etc.
- Normalmente en puerto 21, y además abre canales de datos separados.
- En el stream:
 - El “request” es solo /lena.gif
 - La “response” empieza por GIF87a (cabecera binaria de un GIF)
- **Se demuestra que Dios también se equivoca: Tampoco es FTP.**
- **Lo que sí sabemos es que: Es un protocolo de petición/respuesta donde el cliente envía un “selector/ruta” y el servidor devuelve el recurso binario.**

Preguntamos a google por protocolos similares a http:

dime protocolos similares a http

Modo IA Todo Vídeos Imágenes Vídeos cortos Shopping Noticias Más Herr

Visión general creada por IA

Protocolos similares a HTTP incluyen [HTTPS](#) (HTTP seguro), [WebSockets](#) para comunicación bidireccional, [FTP](#) para transferencia de archivos, y otros como [SMTP](#) (correo), [IMAP/POP3](#) (recuperación de correo), y [Gopher](#), que operan en la capa de aplicación para diferentes propósitos, pero todos facilitan la comunicación en redes, basándose a menudo en TCP/IP.

Protocolos web y de aplicación

- **HTTPS (HTTP Secure):** Es HTTP con una capa de cifrado (TLS/SSL) para seguridad, esencial para transacciones sensibles.
- **WebSockets:** Permite una comunicación full-duplex persistente sobre una única conexión TCP, ideal para aplicaciones en tiempo real, evolucionando desde HTTP.
- **FTP (File Transfer Protocol):** Diseñado específicamente para transferir archivos entre computadoras, distinto de HTTP que es para hipertexto.
- **Gopher:** Un sistema de información distribuida más antiguo, con un protocolo ligero muy similar a HTTP, permitiendo la navegación en menús.

donde:

- **HTTPS no puede ser.** Los datos son legibles, están en texto claro. La característica principal de HTTPS es que cifra la información para que nadie pueda leerla. Si fuera HTTPS, en Wireshark veríamos caracteres aleatorios y sin sentido [Application Data](#). En cambio, en la captura del paquete 547 vemos frases perfectamente legibles como [Welcome to the server!](#) o nombres de archivos como [hello.txt](#).
- **Websocket no puede ser** ya que falta el handshake de bienvenida. Todo WebSocket debe empezar pidiendo permiso a través de una página web (HTTP) con una frase especial llamada [Upgrade: websocket](#). En la captura del paquete 547, el cliente no pide permiso ni usa cabeceras web.
- **FTP ya hemos visto que tampoco.**

Probamos con Gopher: Un sistema de información distribuida más antiguo, con un protocolo ligero muy

similar a HTTP, permitiendo la navegación en menús. Volvemos hacer en ese paquete un "Decode As..."

COMO Gopher :

actividad02.pcap							
No.	Time	Source	Destination	Protocol	Length	Info	
568	31.351551	192.168.1.110	192.168.1.33	TCP	66	62910 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM	
569	31.351573	192.168.1.33	192.168.1.110	TCP	66	7000 → 62910 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128	
570	31.351799	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0	
571	31.352463	192.168.1.110	192.168.1.33	Gopher	65	Request: /lena.gif	
572	31.352469	192.168.1.33	192.168.1.110	TCP	54	7000 → 62910 [ACK] Seq=1 Ack=12 Win=64256 Len=0	
573	31.355524	192.168.1.33	192.168.1.110	Gopher	4150	Response	
574	31.355543	192.168.1.33	192.168.1.110	Gopher	7354	Response	
575	31.355556	192.168.1.33	192.168.1.110	Gopher	2974	Response	
576	31.355954	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2087936 Len=0	
577	31.355963	192.168.1.33	192.168.1.110	Gopher	14654	Response	
578	31.355967	192.168.1.33	192.168.1.110	Gopher	14654	Response	
579	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2092032 Len=0	
580	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2096128 Len=0	
581	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2100224 Len=0	
582	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2102272 Len=0	
583	31.356253	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2073088 Len=0	
584	31.356259	192.168.1.33	192.168.1.110	Gopher	29254	Response	
585	31.356263	192.168.1.33	192.168.1.110	Gopher	2974	Response	
586	31.356265	192.168.1.33	192.168.1.110	Gopher	26334	Response	
587	31.356253	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2077184 Len=0	
588	31.356253	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2081280 Len=0	
589	31.356253	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2085376 Len=0	
590	31.356253	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2089472 Len=0	
591	31.356253	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2093568 Len=0	
592	31.356253	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2097664 Len=0	
593	31.356274	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2101760 Len=0	
594	31.356563	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2043392 Len=0	
595	31.356572	192.168.1.33	192.168.1.110	Gopher	35094	Response	
596	31.356563	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2048000 Len=0	
597	31.356563	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2052096 Len=0	
598	31.356563	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=101917 Win=2056192 Len=0	

donde:

- Wireshark disecciona el tráfico como Gopher, y el comportamiento observado (cliente envía un selector /lena.gif y servidor devuelve bytes que empiezan por GIF87a) encaja exactamente con Gopher.

Volvemos a aplicar el filtro: `tcp.port == 7000`:

actividad02.pcap							
No.	Time	Source	Destination	Protocol	Length	Info	
550	23.659351	192.168.1.110	192.168.1.33	Gopher	55	Request: [Directory list]	
551	23.659418	192.168.1.33	192.168.1.110	TCP	54	7000 → 62897 [ACK] Seq=1 Ack=3 Win=64256 Len=0	
552	23.662103	192.168.1.33	192.168.1.110	Gopher	111	Response	
553	23.662161	192.168.1.33	192.168.1.110	Gopher	220	Response	
554	23.662373	192.168.1.110	192.168.1.33	TCP	54	62897 → 7000 [ACK] Seq=3 Ack=225 Win=2102016 Len=0	
555	23.662554	192.168.1.110	192.168.1.33	TCP	54	62897 → 7000 [FIN, ACK] Seq=3 Ack=225 Win=2102016 Len=0	
556	23.662565	192.168.1.33	192.168.1.110	TCP	54	7000 → 62897 [ACK] Seq=225 Ack=4 Win=64256 Len=0	
557	25.852265	192.168.1.110	192.168.1.33	TCP	66	62900 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM	
558	25.852296	192.168.1.33	192.168.1.110	TCP	66	62900 → 62900 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128	
559	25.852649	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0	
560	25.853468	192.168.1.110	192.168.1.33	Gopher	66	Request: /hello.txt	
561	25.853479	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=13 Win=64256 Len=0	
562	25.858682	192.168.1.33	192.168.1.110	Gopher	67	Response	
563	25.858754	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [FIN, ACK] Seq=14 Ack=13 Win=64256 Len=0	
564	25.859028	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=13 Ack=15 Win=2102272 Len=0	
565	25.859976	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [FIN, ACK] Seq=13 Ack=15 Win=2102272 Len=0	
566	25.859993	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=0 Win=64256 Len=0	
568	31.351551	192.168.1.110	192.168.1.33	TCP	66	62910 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM	
569	31.351573	192.168.1.33	192.168.1.110	TCP	66	7000 → 62910 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128	
570	31.351799	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0	
571	31.352463	192.168.1.110	192.168.1.33	Gopher	65	Request: /lena.gif	
572	31.352469	192.168.1.33	192.168.1.110	TCP	54	7000 → 62910 [ACK] Seq=1 Ack=12 Win=64256 Len=0	
573	31.355524	192.168.1.33	192.168.1.110	Gopher	4150	Response	
574	31.355543	192.168.1.33	192.168.1.110	Gopher	7354	Response	
575	31.355556	192.168.1.33	192.168.1.110	Gopher	2974	Response	
576	31.355954	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2087936 Len=0	
577	31.355963	192.168.1.33	192.168.1.110	Gopher	14654	Response	
578	31.355967	192.168.1.33	192.168.1.110	Gopher	14654	Response	
579	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2092032 Len=0	
580	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2096128 Len=0	
581	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2100224 Len=0	
582	31.355954	192.168.1.110	192.168.1.33	TCP	54	[TCP Window Update] 62910 → 7000 [ACK] Seq=12 Ack=14317 Win=2102272 Len=0	
583	31.356253	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=12 Ack=43517 Win=2073088 Len=0	
584	31.356259	192.168.1.33	192.168.1.110	Gopher	29254	Response	
585	31.356263	192.168.1.33	192.168.1.110	Gopher	2974	Response	
586	31.356265	192.168.1.33	192.168.1.110	Gopher	26334	Response	

donde:

- Con el filtro `tcp.port == 7000` se observan varias conexiones donde Wireshark ya decodifica Gopher.
- Gopher Request: [Directory list] (listado de directorio/menú).
- Gopher Request: /hello.txt (petición de fichero de texto).
- Gopher Request: /lena.gif (petición de imagen GIF).
- Respuestas "Gopher Response" asociadas.

- **Conclusión: la aplicación usa el protocolo Gopher sobre TCP, pero en un puerto no estándar, el 7000 en vez de 70.**

2. Información intercambiada entre el cliente y el servidor

Vamos a realizar una captura de pantalla en la que se vea la información intercambiada entre el cliente y el servidor:

- Aplicamos filtro: `tcp.port == 7000`.
- Seleccionaremos dos paquetes
 - “Gopher Request: /hello.txt”
 - “Gopher Request: /lena.gif”.
- Analyze → Follow → TCP Stream → Analizamos la información que se intercambia.

Listar directorio - Paquete 550

actividad2.pcap						
No.	Time	Source	Destination	Protocol	Length	Info
543	2021-01-21 00:15:30,365000Z	216.58.211.228	192.168.1.33	TLSv1.2	105	Application Data
544	2021-01-21 00:15:30,365190Z	192.168.1.33	216.58.211.228	TCP	66	43128 → 443 [ACK] Seq=5709 Ack=403395 Win=4548 Len=0 TSval=2
545	2021-01-21 00:15:30,365203Z	192.168.1.33	216.58.211.228	TLSv1.2	105	Application Data
546	2021-01-21 00:15:30,370029Z	216.58.211.228	192.168.1.33	TCP	66	443 → 43128 [ACK] Seq=403395 Ack=5748 Win=367 Len=0 TSval=2
547	2021-01-21 00:15:31,277631Z	192.168.1.110	192.168.1.33	TCP	66	62897 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SAC
548	2021-01-21 00:15:31,277655Z	192.168.1.33	192.168.1.110	TCP	66	7000 → 62897 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
549	2021-01-21 00:15:31,277851Z	192.168.1.110	192.168.1.33	TCP	54	62897 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
550	2021-01-21 00:15:31,278629Z	192.168.1.110	192.168.1.33	Gopher	56	Response

```

> Frame 550: Packet, 56 bytes on wire (448 bits), 56 bytes captured (448 bits)
> Ethernet II, Src: Dell_b4:43:7e (e4:b9:7a:b4:43:7e), Dst: Microsoft_38:01:04
> Internet Protocol Version 4, Src: 192.168.1.110, Dst: 192.168.1.33
> Transmission Control Protocol, Src Port: 62897, Dst Port: 7000, Seq: 1, Ack: 1
> Gopher response: [Unknown]
  Unknown Gopher transaction data: \r\n

```

donde:

- Vemos una petición de tipo Gopher.
- En el panel de detalles del paquete, vemos que el cliente (192.168.1.110) envía una petición vacía: sólo un salto de línea `\r\n`. En el protocolo Gopher, esto se interpreta como la orden para listar el directorio raíz.

La respuesta del servidor está en los paquetes 552 y 553:

actividad2.pcap						
No.	Time	Source	Destination	Protocol	Length	Info
549	2021-01-21 00:15:31,277851Z	192.168.1.110	192.168.1.33	TCP	54	62897 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
550	2021-01-21 00:15:31,278629Z	192.168.1.110	192.168.1.33	Gopher	56	Response
551	2021-01-21 00:15:31,278696Z	192.168.1.33	192.168.1.110	TCP	54	7000 → 62897 [ACK] Seq=1 Ack=3 Win=64256 Len=0
552	2021-01-21 00:15:31,281381Z	192.168.1.33	192.168.1.110	Gopher	111	Request: iWelcome to the 192.168.1.33:7000 server!\tfake\t(N
553	2021-01-21 00:15:31,281439Z	192.168.1.33	192.168.1.110	Gopher	220	Request: i\tfake\t(NULL)\t0
554	2021-01-21 00:15:31,281651Z	192.168.1.110	192.168.1.33	TCP	54	62897 → 7000 [ACK] Seq=3 Ack=225 Win=2102016 Len=0
555	2021-01-21 00:15:31,281832Z	192.168.1.110	192.168.1.33	TCP	54	7000 → 62897 [FIN, ACK] Seq=3 Ack=225 Win=2102016 Len=0
556	2021-01-21 00:15:31,281843Z	192.168.1.33	192.168.1.110	TCP	54	7000 → 62897 [ACK] Seq=225 Ack=4 Win=64256 Len=0

```

> Frame 552: Packet, 111 bytes on wire (888 bits), 111 bytes captured (888 bits)
> Ethernet II, Src: Microsoft_38:01:04 (00:15:5d:38:01:04), Dst: Dell_b4:43:7e
> Internet Protocol Version 4, Src: 192.168.1.33, Dst: 192.168.1.110
> Transmission Control Protocol, Src Port: 7000, Dst Port: 62897, Seq: 1, Ack: 1
> Gopher request: iWelcome to the 192.168.1.33:7000 server!\tfake\t(NULL)\t0
  Gopher client request: iWelcome to the 192.168.1.33:7000 server!\tfake\t(N

```

Fichero hello.txt - Paquete 560

tcp.stream eq 7

No.	Time	Source	Destination	Protocol	Length	Info
557	25.852265	192.168.1.110	192.168.1.33	TCP	66	62900 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
558	25.852296	192.168.1.33	192.168.1.110	TCP	66	7000 → 62900 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 WS=256
559	25.852649	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0 MSS=1460 WS=256
560	25.853468	192.168.1.110	192.168.1.33	Gopher	66	Request: /hello.txt
561	25.853479	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0 MSS=1460 WS=256
562	25.858682	192.168.1.33	192.168.1.110	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0 MSS=1460 WS=256
563	25.858754	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0 MSS=1460 WS=256
564	25.859028	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0 MSS=1460 WS=256
565	25.859976	192.168.1.110	192.168.1.33	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0 MSS=1460 WS=256
566	25.859993	192.168.1.33	192.168.1.110	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0 MSS=1460 WS=256

Wireshark · Seguir secuencia TCP (tcp.stream eq 7) · actividad2.pcap

/hello.txt
Hello world!

donde:

- En la parte del cliente: la petición con el nombre del fichero y el lugar donde está: `/hello.txt`.
- En la parte del servidor: la respuesta del servidor que envía el contenido de ese fichero: `Hello world!`.

Fichero lena.gif - Paquete 571

tcp.stream eq 8

No.	Time	Source	Destination	Protocol	Length	Info
568	31.351551	192.168.1.110	192.168.1.33	TCP	66	62910 → 7000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
569	31.351573	192.168.1.33	192.168.1.110	TCP	66	7000 → 62910 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
570	31.351799	192.168.1.110	192.168.1.33	TCP	54	62910 → 7000 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
571	31.352463	192.168.1.110	192.168.1.33	Gopher	65	Request: /lena.gif
572	31.352469	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
573	31.355524	192.168.1.33	192.168.1.110	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
574	31.355543	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
575	31.355556	192.168.1.33	192.168.1.110	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
576	31.355954	192.168.1.110	192.168.1.33	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
577	31.355963	192.168.1.33	192.168.1.110	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
578	31.355967	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
579	31.355954	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
580	31.355954	192.168.1.110	192.168.1.33	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
581	31.355954	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
582	31.355954	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
583	31.356253	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
584	31.356259	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
585	31.356263	192.168.1.33	192.168.1.110	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
586	31.356265	192.168.1.33	192.168.1.110	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
587	31.356253	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0
588	31.356253	192.168.1.110	192.168.1.33	TCP	54	7000 → 62900 [ACK] Seq=1 Ack=12 Win=64240 Len=0
589	31.356253	192.168.1.110	192.168.1.33	TCP	54	62900 → 7000 [ACK] Seq=1 Ack=12 Win=64240 Len=0

Wireshark · Seguir secuencia TCP (tcp.stream eq 8) · actividad2.pcap

/lena.gif
GIF87a.....a].LP.8D.]Y.]a.8H.YY.DL....Ua.0@.,@.H.
<D.....\$<.....YP.....P.4a.HP.4.....Y.@a.4.....qa.4....D].De..m...}.PU..
q..y.8a.La.uu.yi.ye.DUq,a.mai,Y.@P.e].LP.ae.]].HP.8D.4D.]Y.U].DL.UY.@H.@P
.YY....<D.....@. <m L....YPY LH.8H.8m.8a.@D.4..y..m]<.y..m.....ydu..jU.4.e.....U...je.)
).ya.ym.0U.qq.ma.m].i].iu[e].a].DL.Ye.Yi[@....<H.8H..}@(q(H)\$@.....P.8y.<nD.YPY.Di@L.4e.q.4.Py
Y.D.u.LeU.@@....Y.0...}y...L].yi.qm.a.um.ii....@H.i'8Pa(U.Y).DL....<D.4DL@...L.0m<....UP].@]
.0@.4...Y.<LUU.8U.8..m..y..m@U.i..i..y.}qi.y.PU.}yy8e.}....yy.mm.ee.ym.um.PL.i].Um.BD.aY.i
e.Y.}q.PU.jm...q8i.}.ja.um.mm.y].uy.qi.U..u].HP..y.qq.im.ai.my.PP.em.eq.8L.LL.Y].....Y...
@.W.<....ctn.;Y.d....9].X.j.J..j..x....V.....E..~.A.R.W..B.*....8p..".P.BU.".y.....W
s'0....d....z.0....f.P...).{....Kw..3t..1C....F.X/...j...|y=a0.S..o.[{...};I].3..F....{#7f,w/.....K7p..k..0FY..u
....y ..

donde:

- En la parte del cliente: la petición `/lena.gif`.

- En la parte del servidor: la respuesta:

```
GIF87a.....a].LP.8D.]Y.]a.8H.YY.DL....Ua.0@.,@.H.  
<D.....$<.....YP.....P.4a.HP.4.....Y.@a.4.....qa.4....D].De..m...}.PU..  
q..y.8a.La.uu.yi.ye.DUq,a.mai,Y.@P.e].LP.ae.]].HP.8D.4D.]Y.U].DL.UY.@H.@P  
.YY....<D.....@. <m L....YPY LH.8H.8m.8a.@D.4..y..m].. Que es el fichero en crudo.
```

- La respuesta del servidor empieza por GIF87a, que es la firma ("magic bytes") de un fichero GIF. Eso indica que lo que viene a continuación son bytes binarios del propio fichero.

Veamos la foto en crudo: Estando en este paque 571, en la ventana: Analyze → Follow → TCP Stream → Cambiamos “Mostrar como”: Raw.

donde:

- Vemos la petición del cliente y del servidor en crudo.

Cambiamos unas opciones para poder guardar la foto:

0 client pkt(s), 13 server pkt(s), 0 turn(s).

192.168.1.33:7000 → 192.168.1.110:62910 (227 | ▾) Mostrar como Raw ▾ No delta times ▾ Secuencia 8 ▾

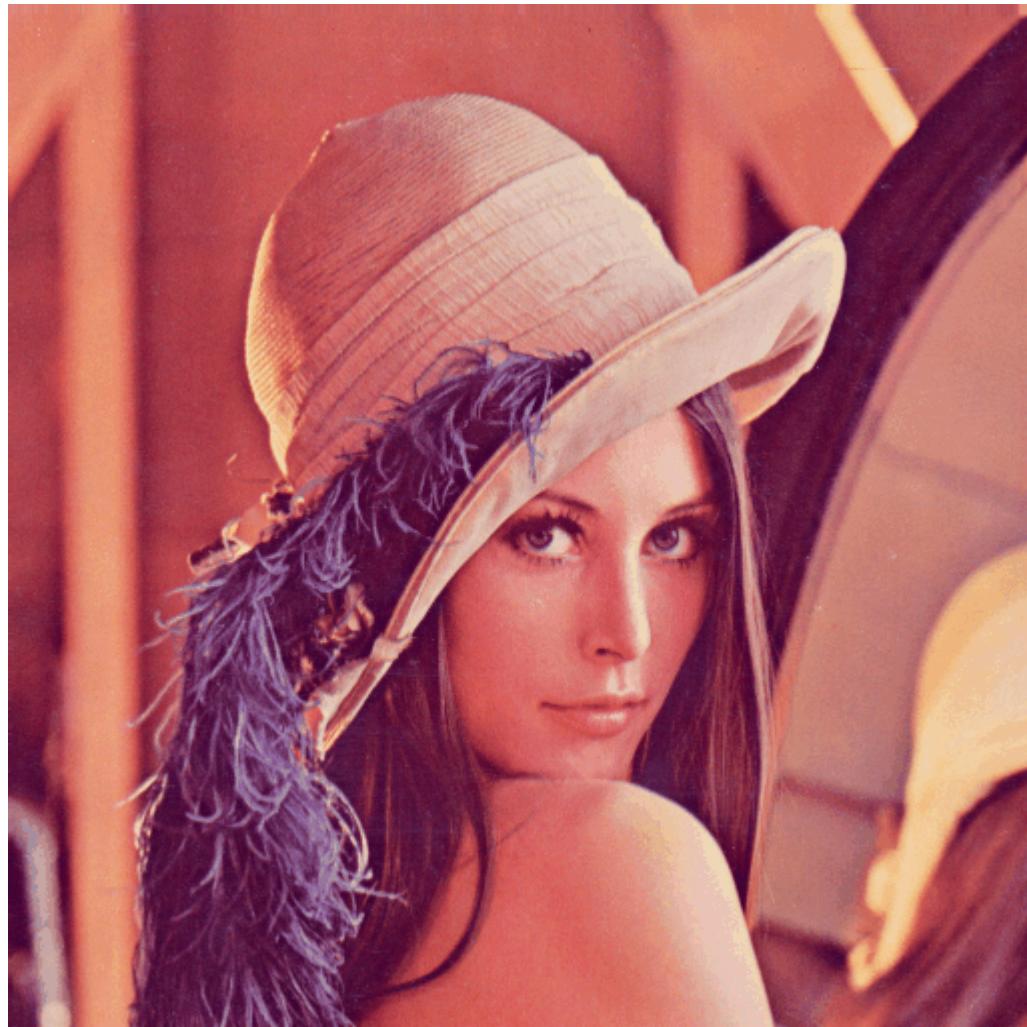
Buscar: Mayúsculas y minúsculas [Buscar siguiente](#)

 Ayuda [Filtrar secuencia](#) [Imprimir](#) [Guardar como...](#) [Atrás](#)  Cerrar

donde:

- “Show data as / Mostrar datos como” debe estar en Raw.
 - Tipo “Stream/Conversation” o “Direction: 192.168.1.33:7000 --> 192.168.1.110
 - Pulsamos el botón guardar y damos un nombre al fichero.

Abrimos el fichero descargado:



La investigación del fichero pcap con una captura de tráfico concluye satisfactoriamente con la identificación del protocolo Gopher y la extracción exitosa de los activos hello.txt y lena.gif. También se ha podido reconstruir del archivo binario a partir de los datos en formato Raw.