

Infraestructura de Comunicaciones

Laboratorio 2 - Parte 2

Juan Esteban MÉNDEZ

201531707

Michel SUCCAR

201532368

Septiembre 15, 2019

Contents

1	Pruebas de Conectividad y Análisis DNS	2
2	Captura de paquetes servicio Streaming	3
3	Análisis de Protocolo FTP	5
4	Análisis de los protocolos de correo electrónico: SMTP y POP3	8
5	Análisis del protocolo HTTP	12
6	Análisis del protocolo HTTPS sobre youtube.com	15
7	Preguntas	18

1 Pruebas de Conectividad y Análisis DNS

Para iniciar las pruebas de conectividad con el servidor DNS, fue necesario abrir el *Command Prompt* de la máquina Windows que hizo las veces de cliente:

1. Una vez adentro del Command Prompt se procedió a realizar las pruebas de conectividad al enrutador local de usuarios y al otro usuario de la red usando la dirección IP respectiva de cada máquina:
2. Luego, se realizó una prueba de conectividad entre nuestra máquina y el servidor DNS y se tomaron las respectivas capturas en *WireShark*:

```
C:\Users\Equipo D>ping 192.168.1.14

Haciendo ping a 192.168.1.14 con 32 bytes de datos:
Respuesta desde 192.168.1.14: bytes=32 tiempo=1ms TTL=62

Estadísticas de ping para 192.168.1.14:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
        (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 1ms, Máximo = 1ms, Media = 1ms
```

3. Igualmente se realizó una prueba de conectividad al servidor Web utilizando su dirección URL (`ping web.grupo4.ultrared.com`) y se guardaron las capturas en *WireShark*:

```
C:\Users\Equipo D>ping web.grupo4.ultrared.com

Haciendo ping a web.grupo4.ultrared.com [192.168.1.24] con 32 bytes de datos:
Respuesta desde 192.168.1.24: bytes=32 tiempo<1ms TTL=62
Respuesta desde 192.168.1.24: bytes=32 tiempo=1ms TTL=62
Respuesta desde 192.168.1.24: bytes=32 tiempo<1ms TTL=62
Respuesta desde 192.168.1.24: bytes=32 tiempo=1ms TTL=62

Estadísticas de ping para 192.168.1.24:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
        (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 0ms, Máximo = 1ms, Media = 0ms
```

4. Se borró la cache de resolución DNS con el comando `ipconfig /flushdns`, y se verificó su funcionamiento con el comando `ipconfig /displaydns`:

```
C:\Users\Equipo D>ipconfig /flushdns

Configuración IP de Windows

Se vació correctamente la caché de resolución de DNS.

C:\Users\Equipo D>ipconfig /displaydns

Configuración IP de Windows

No se pudo mostrar la caché de resolución de DNS.
```

5. Prueba de conectividad al servidor Web utilizando la dirección IP:

```
C:\Users\Equipo D>ping 192.168.1.24

Haciendo ping a 192.168.1.24 con 32 bytes de datos:
Respuesta desde 192.168.1.24: bytes=32 tiempo=1ms TTL=62

Estadísticas de ping para 192.168.1.24:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
        (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 1ms, Máximo = 1ms, Media = 1ms
```

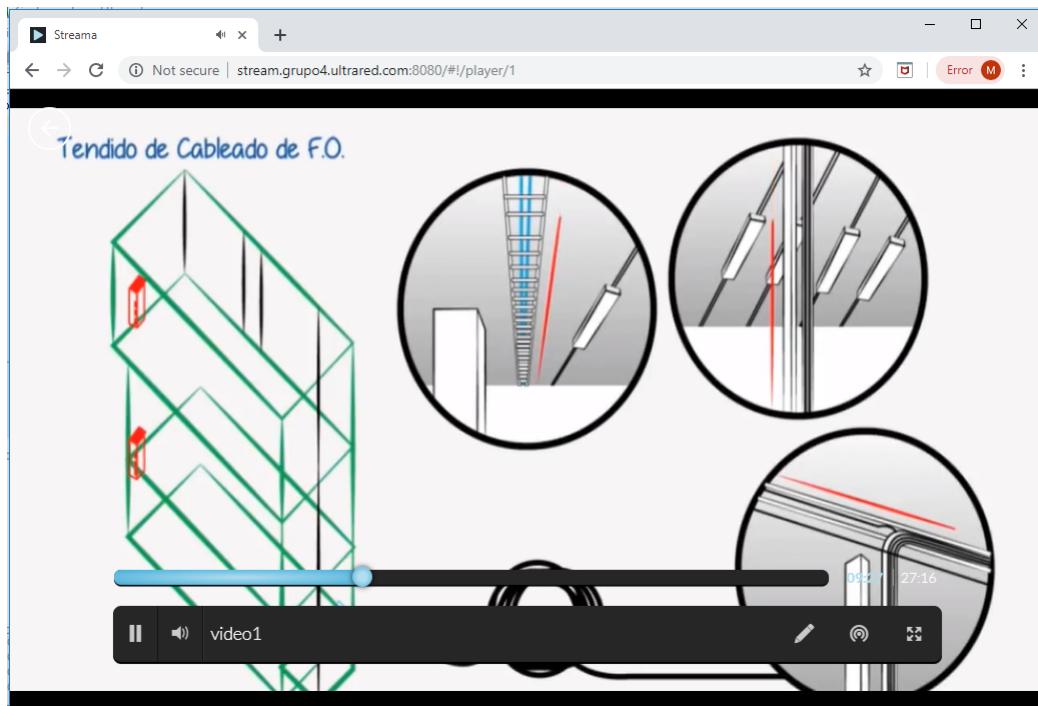
2 Captura de paquetes servicio Streaming

A continuación se presenta el procedimiento realizado para capturar los paquetes en *WireShark* para el tráfico entre la máquina con el servicio de streaming y su respectivo cliente:

1. Se ejecutó la emisión del video en la máquina virtual que realiza las veces de servidor.

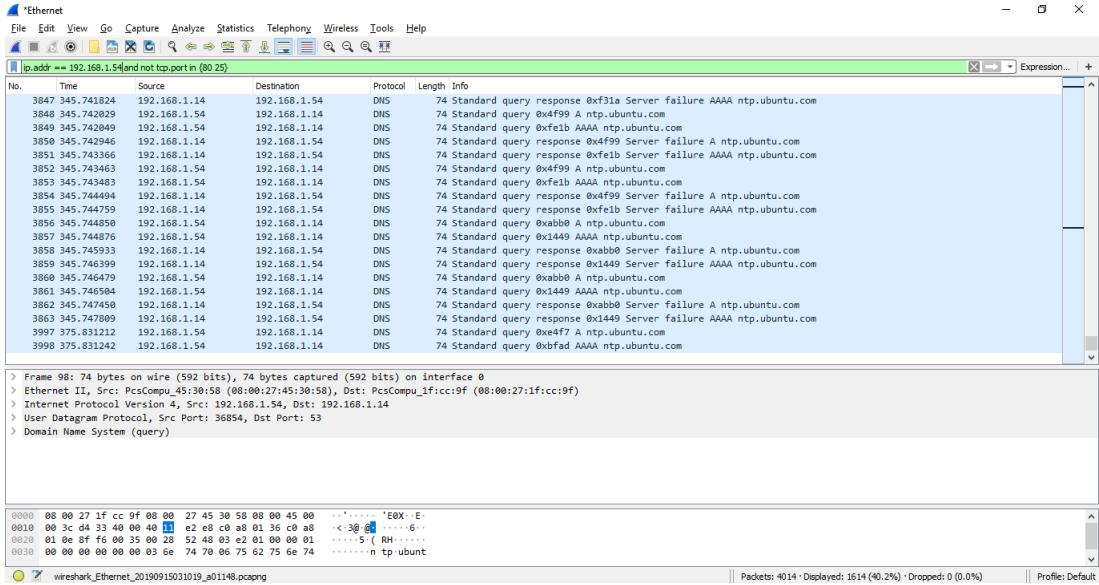
Por medio de la consola de la maquina ubuntu desktop, se ejecuto el ejecutable de **streama 1.7.0** por medio de java. Cuando el ejecutable se declaro en funcionamiento se utilizo el navegador Google Chrome para acceder a la dirección `localhost:8080`, donde se accede a streama y se reproduce el video subido en la actividad 1 del laboratorio 2.

2. Iniciamos una nueva captura en *WireShark* y se reprodujo el contenido multimedia en la máquina virtual cliente:



Como se ve en la imagen, el video logra continuar reproduciendo en donde el servidor va, logrando el propósito de streaming.

3. Visualizamos durante algunos segundos el video, y luego analizamos la captura de paquetes en *WireShark*:



Como se puede ver en la imagen de arriba, hay un constante intercambios de paquetes entre el servidor de streaming y el cliente solicitante.

3 Análisis de Protocolo FTP

Para iniciar el análisis del protocolo FTP, primero limpiamos la cache de resolución de DNS con el comando `ipconfig /flushdns` y reiniciamos el modo de captura de *WireShark*. Posteriormente, realizamos los siguientes pasos:

1. Para comprobar la conexión con el servidor FTP, le hicimos ping mediante su dirección IP:

```
C:\Users\Equipo D>ping 192.168.1.44

Haciendo ping a 192.168.1.44 con 32 bytes de datos:
Respuesta desde 192.168.1.44: bytes=32 tiempo=1ms TTL=62

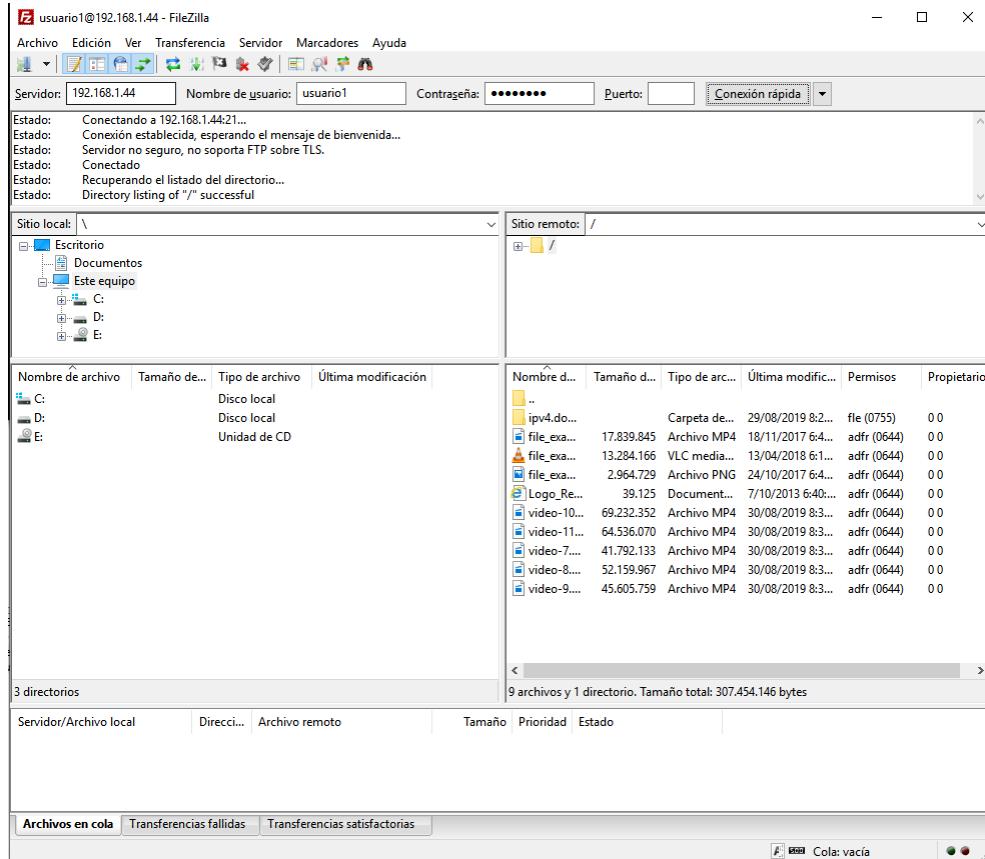
Estadísticas de ping para 192.168.1.44:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
                (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 1ms, Máximo = 1ms, Media = 1ms
```

2. Accedimos al programa cliente de FTP (*FileZilla*) con la máquina cliente con los siguientes credenciales:

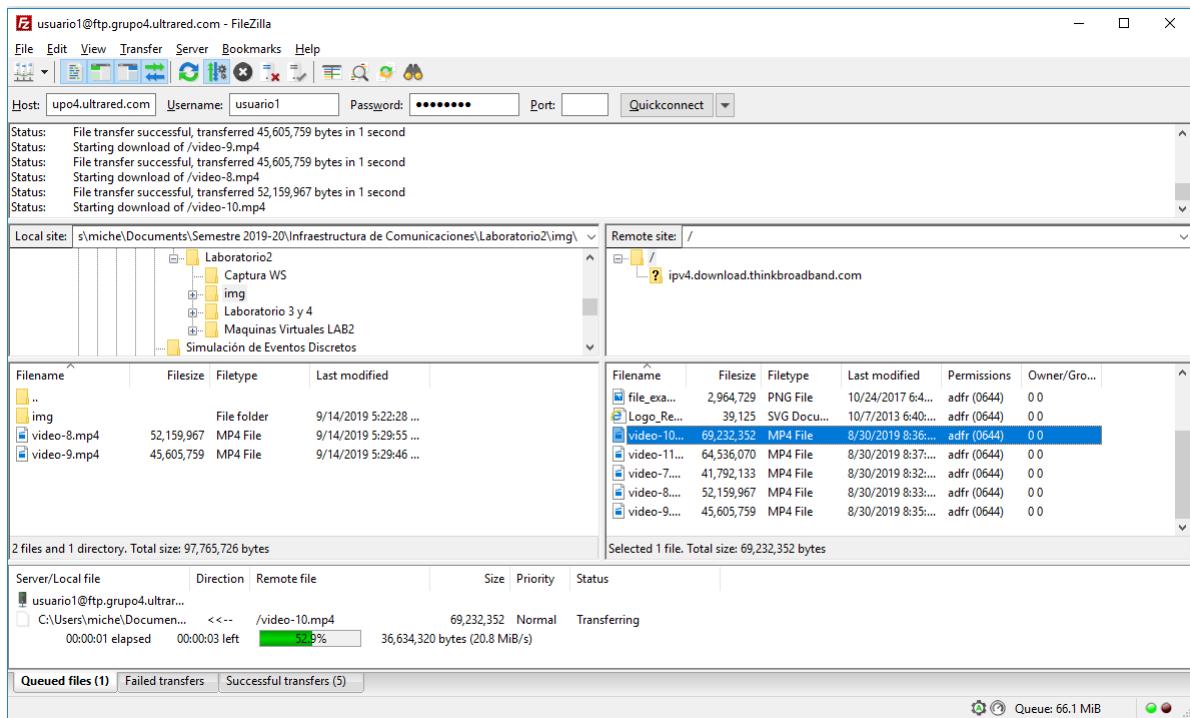
- Servidor: `ftp.grupo4.ultrared.com`
- Nombre de usuario: `usuario1`
- Contraseña: `usuario1`
- Puerto: 21

Utilizamos la dirección IP 192.168.1.44 para conectarnos al servidor FTP con el `usuario1` al puerto 21.

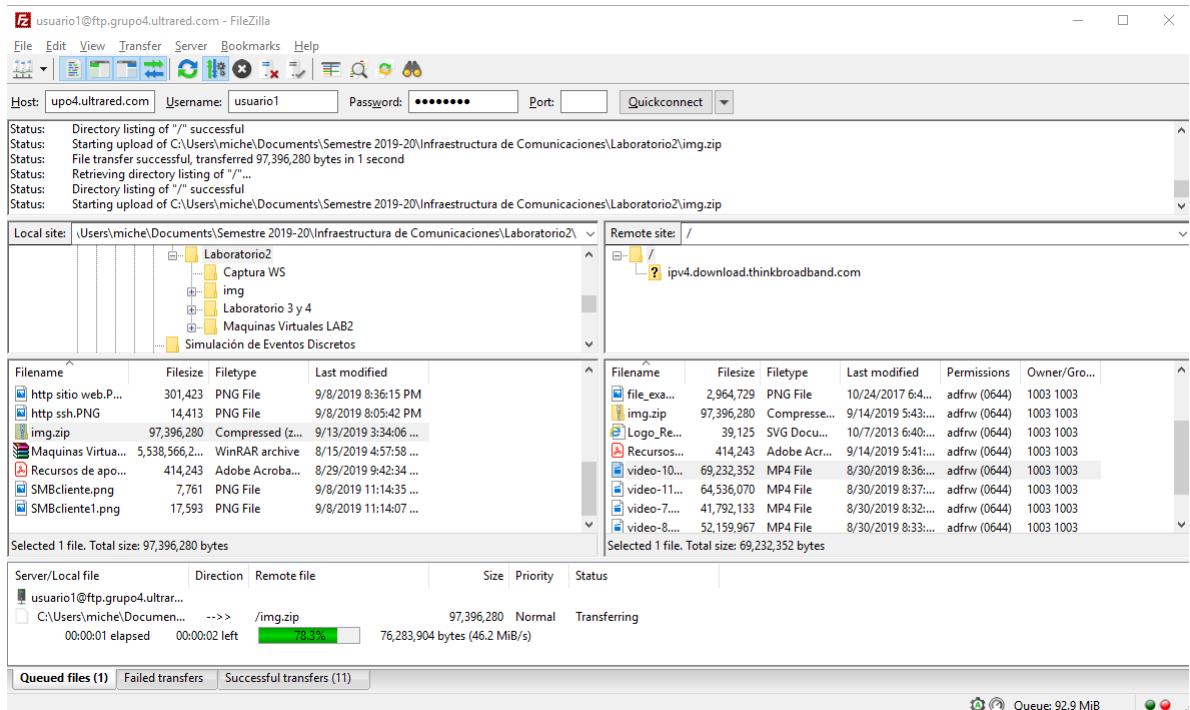
3. Al ingresar los credenciales del `usuario1`, *FileZilla* nos mostró el contenido del directorio compartido del `usuario1`:



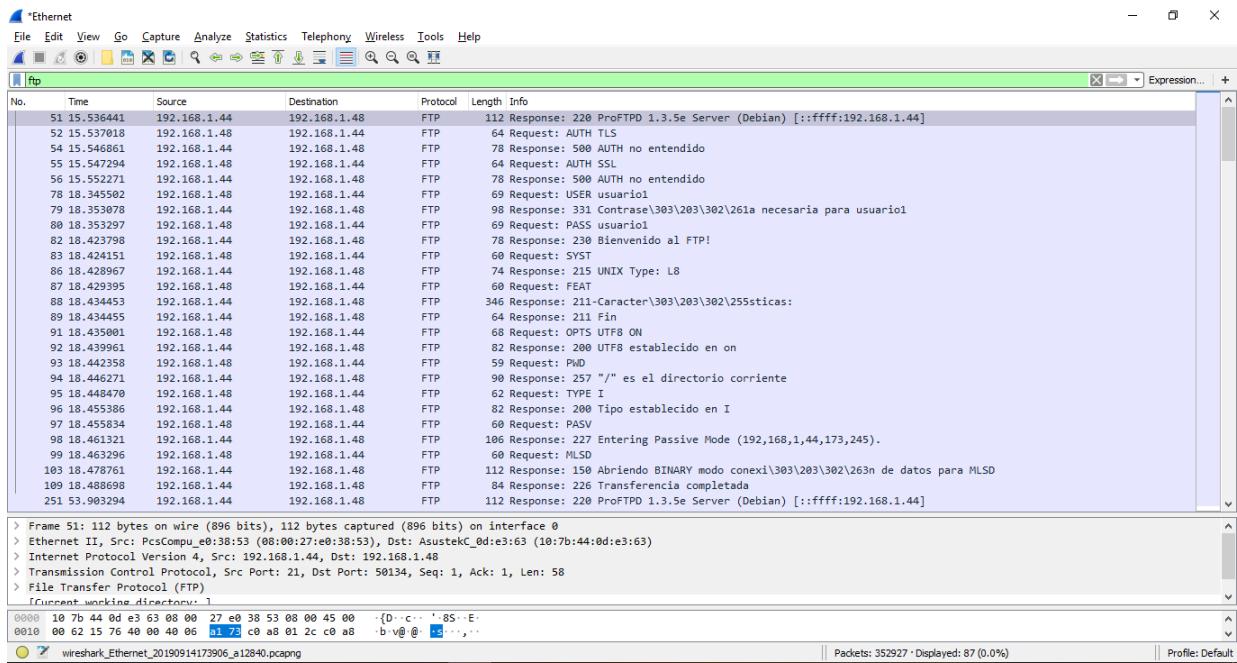
4. Escogimos uno de los archivos en el directorio compartido y lo descargamos a la máquina cliente:



5. Luego cargamos un archivo en la máquina cliente y lo subimos al directorio compartido del **usuario1** en el servidor FTP:



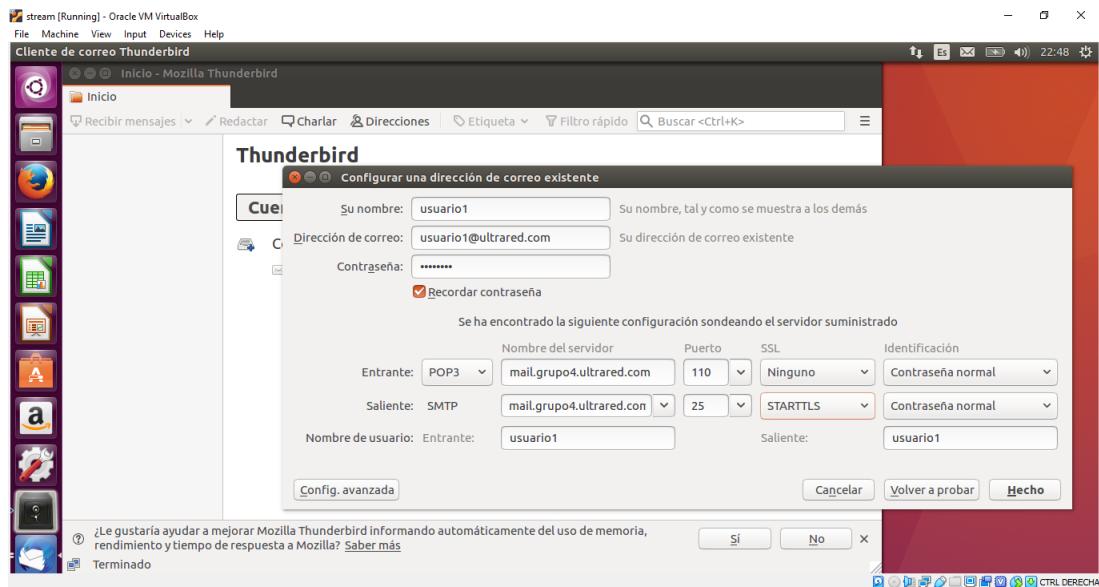
6. Finalmente, detuvimos la captura en *WireShark* y realizamos un análisis del protocolo FTP mediante esta herramienta:



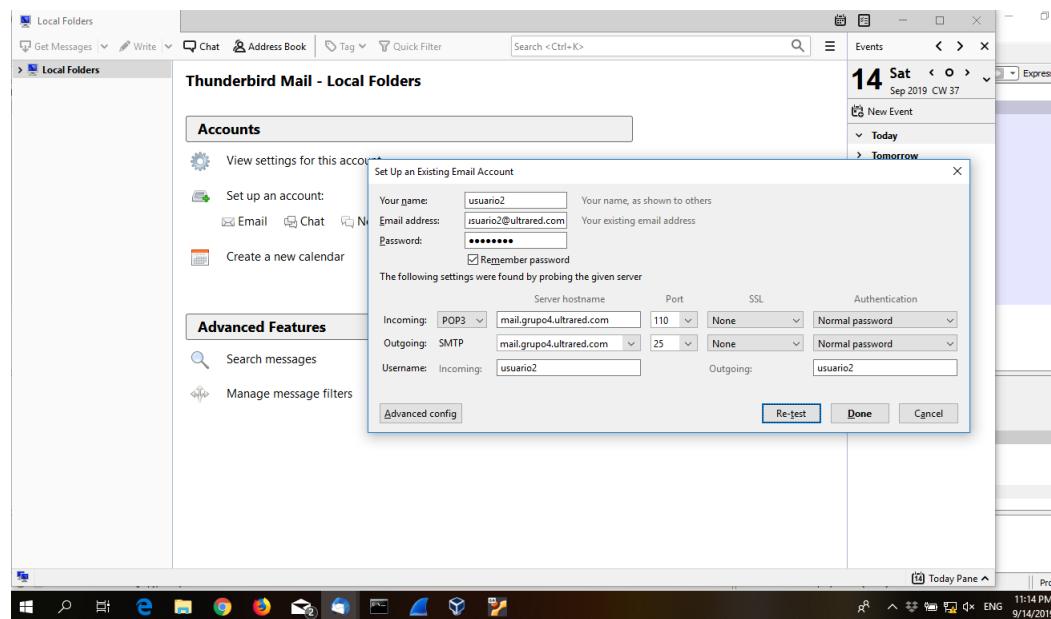
4 Análisis de los protocolos de correo electrónico: SMTP y POP3

Para iniciar el análisis los protocolos SMTP y POP3, primero limpiamos la cache de resolución de DNS con el comando `ipconfig /flushdns` y reiniciamos el modo de captura de *WireShark*. Posteriormente, realizamos los siguientes pasos:

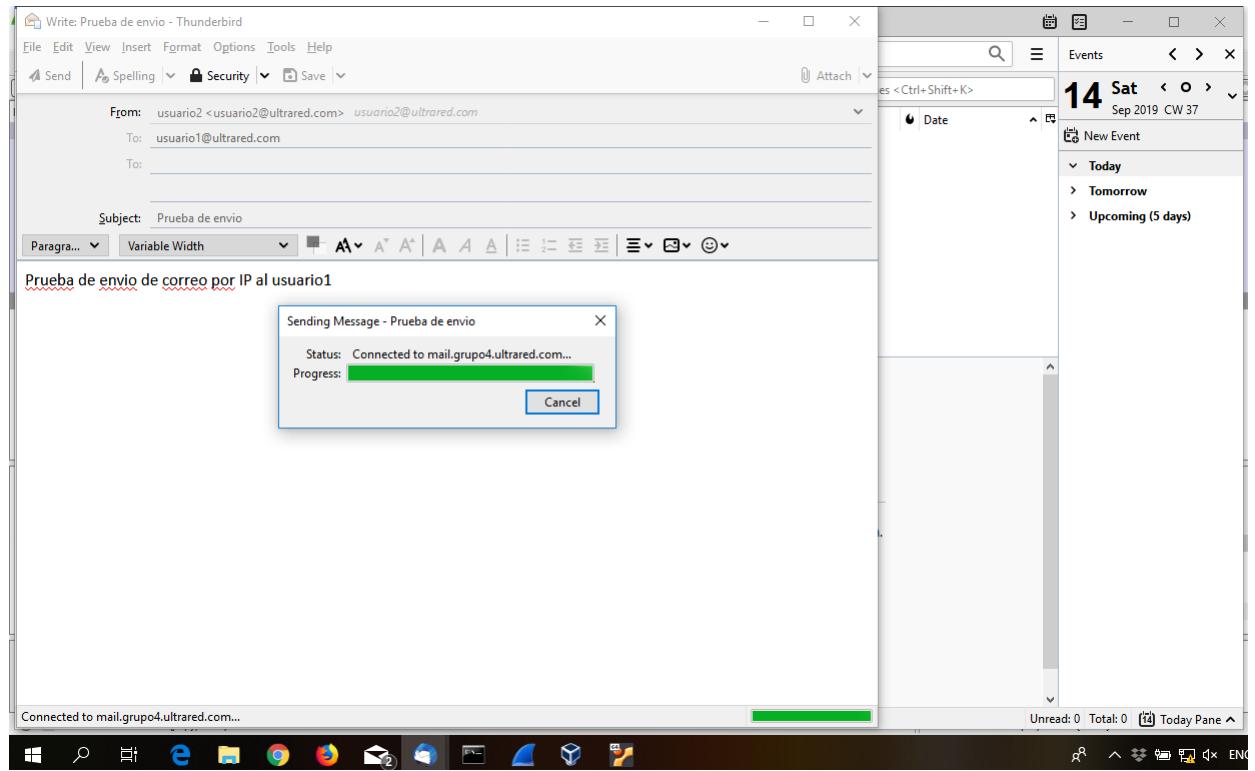
1. Abrimos la aplicación Mozilla *Thunderbird* en cada una de las máquinas cliente y reiniciamos el modo de captura de *WireShark*.
2. Accedemos con el `usuario1` y el `usuario2`, en la máquina cliente *Ubuntu* y en la máquina cliente *Windows* respectivamente:
 - Login `usuario1` en máquina *Ubuntu*:



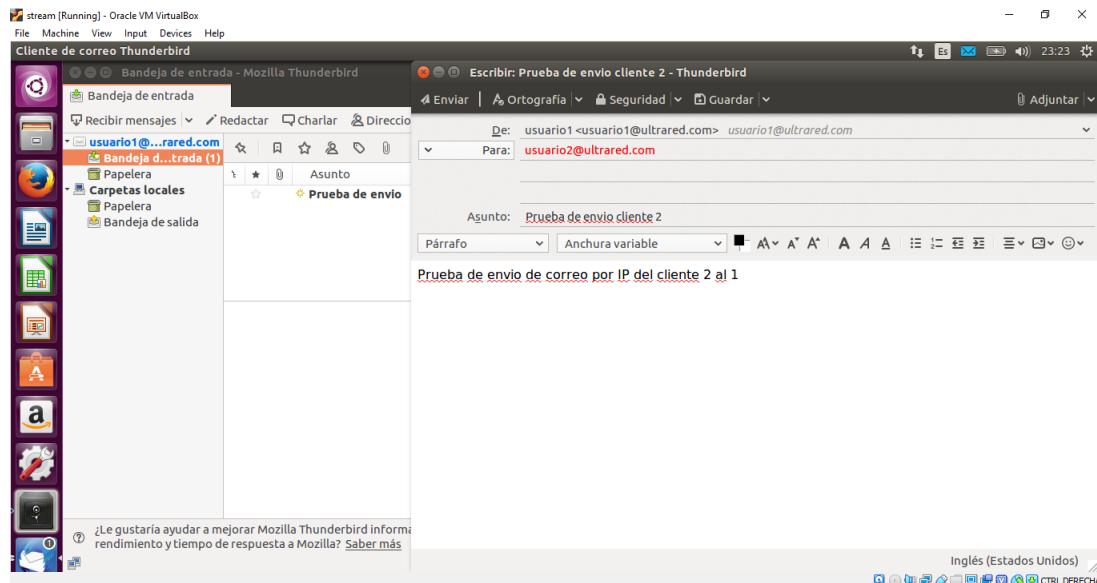
- Login `usuario2` en máquina *Windows*:



3. Guardamos los paquetes capturados en *WireShark* y reiniciamos el modo de captura. Enviamos un correo desde el **usuario1** al **usuario2**:

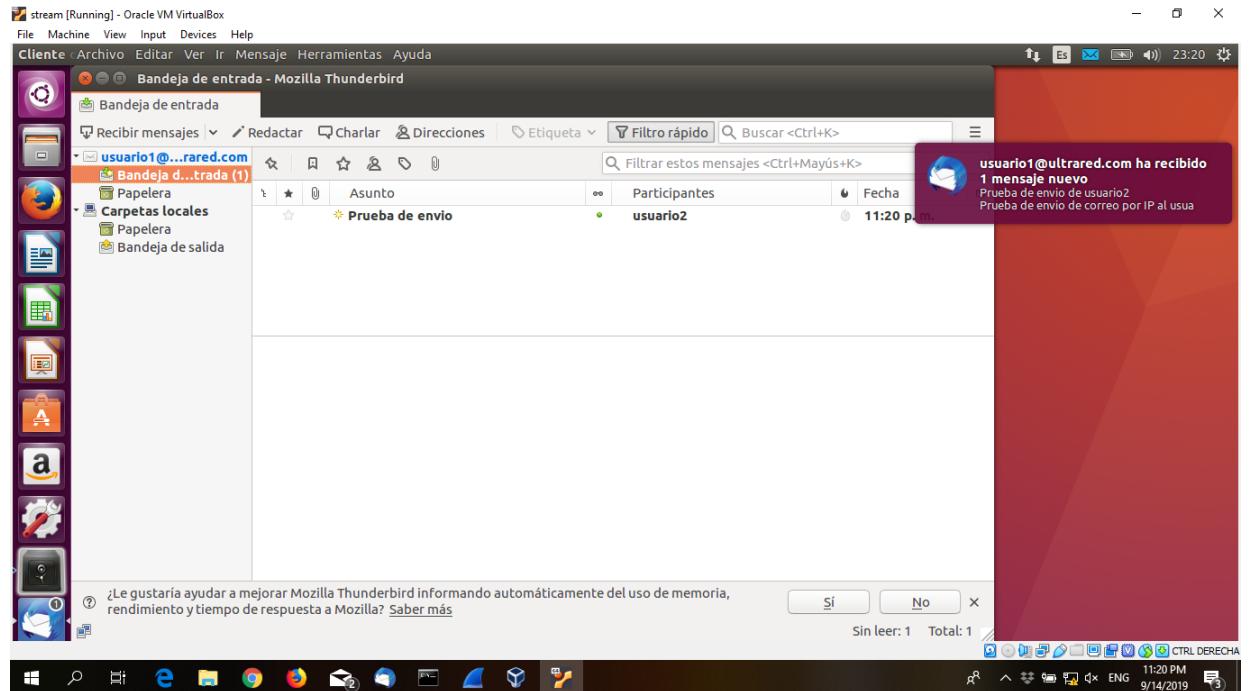


4. Guardamos los paquetes capturados en *WireShark* y reiniciamos el modo de captura. Ahora, desde la cuenta del **usuario2** enviamos un correo al **usuario1**:

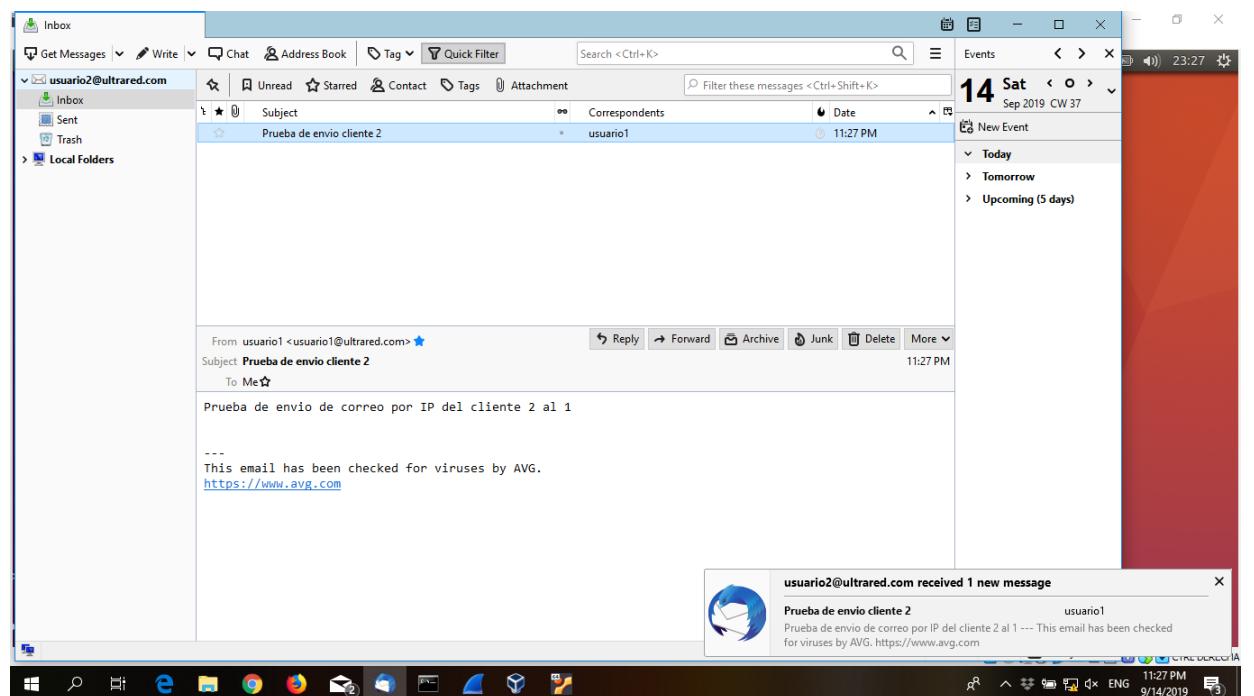


5. Comprobamos que los correos si hayan sido enviados correctamente, chequeando la **bandeja de entrada** de cada usuario:

- Bandeja de entrada del usuario1

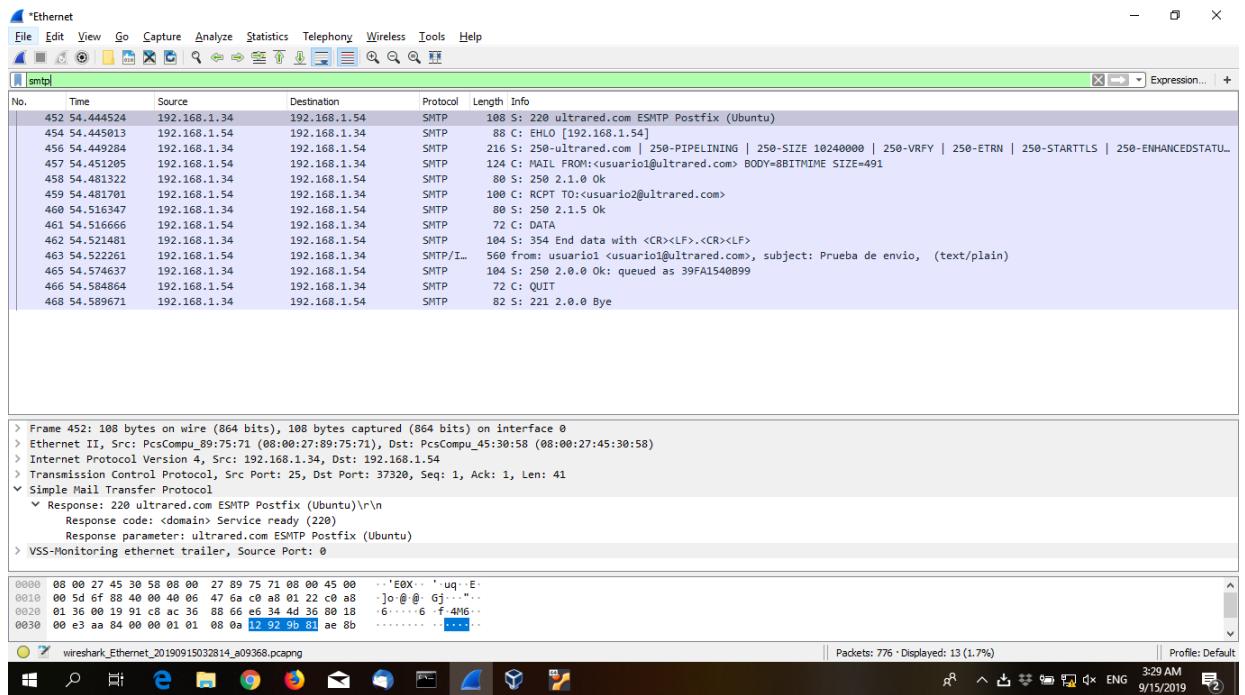


- Bandeja de entrada del usuario2

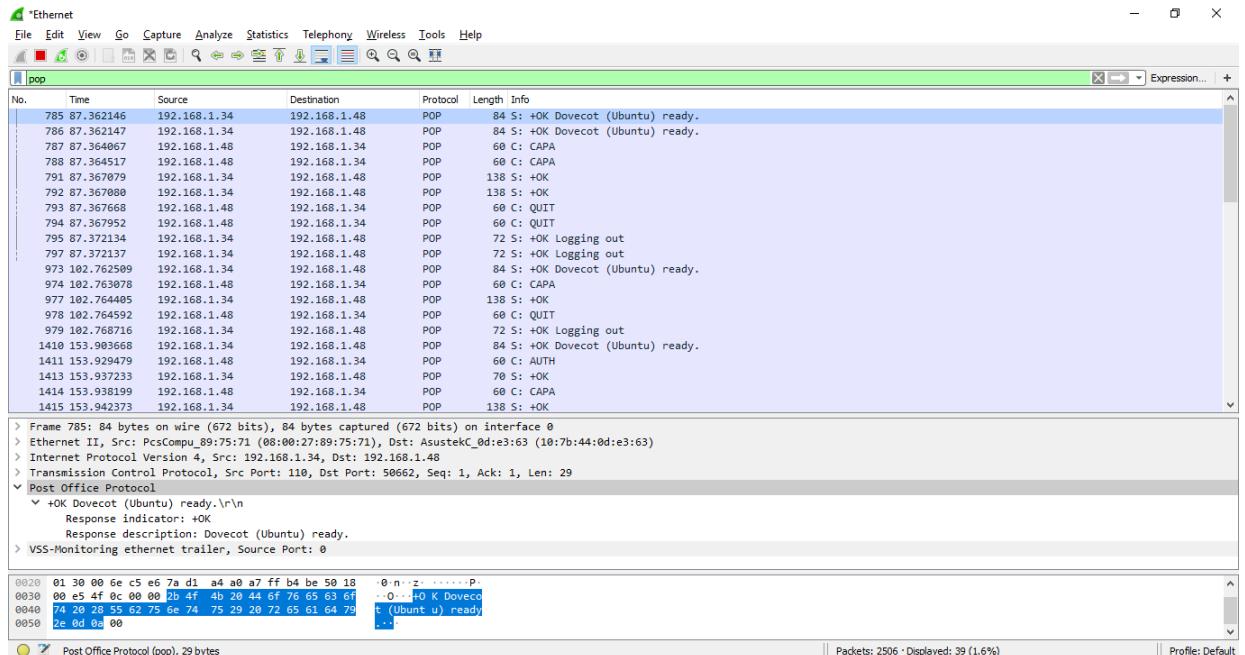


6. Guardamos los paquetes capturados en *WireShark*, y los analizamos:

- Observamos el protocolo SMTP



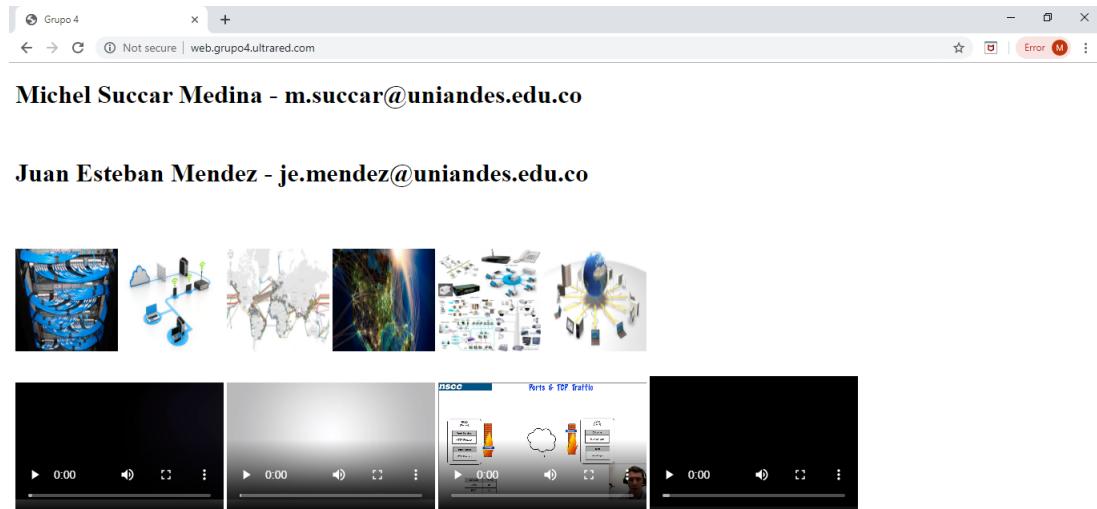
- Observamos el protocolo POP3



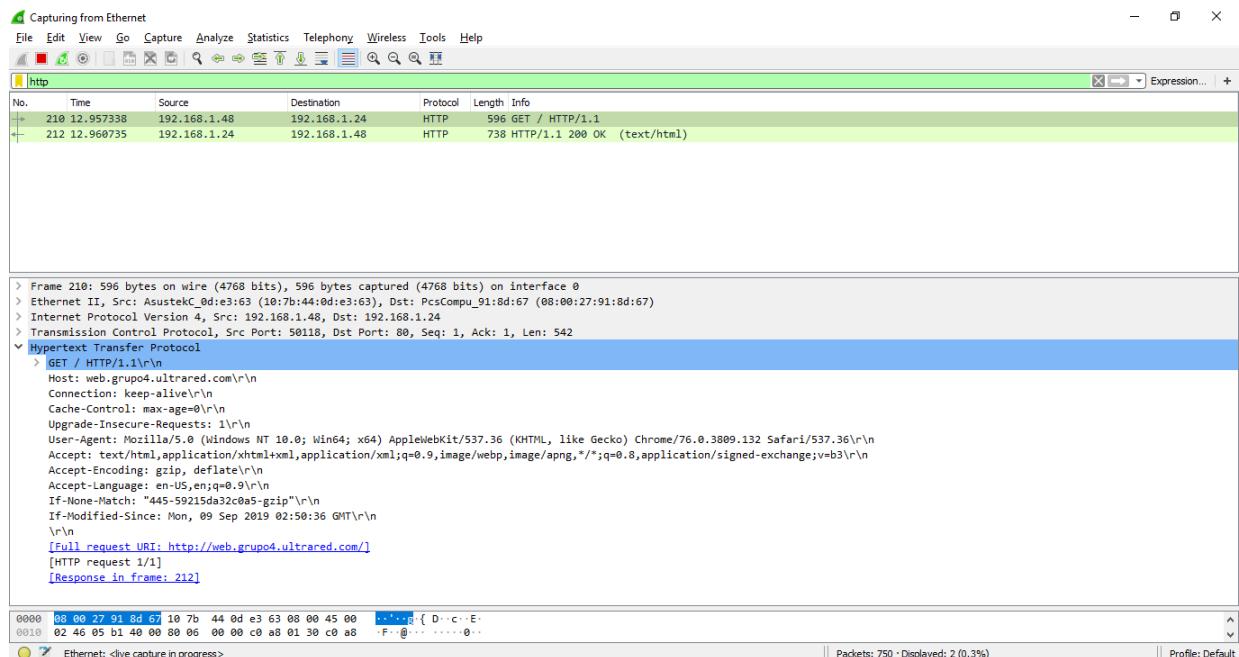
5 Análisis del protocolo HTTP

Para iniciar el análisis del protocolo HTTP, primero limpiamos la cache de resolución de DNS con el comando `ipconfig /flushdns` y reiniciamos el modo de captura de *WireShark*. Posteriormente, realizamos los siguientes pasos:

1. Abrimos *Google Chrome* desde la máquina cliente y nos conectamos al servicio HTTP prestado en red, ingresando a la URL `http://web.grupo4.ultrared.com`:



2. Visualizamos los diferentes contenidos de la página y realizamos la captura en *WireShark*.



The screenshot shows the Wireshark interface capturing traffic from an Ethernet interface. The packet list pane shows two HTTP requests from 192.168.1.48 to 192.168.1.24. The details pane displays the request for frame 210, which is an HTTP 1.1 GET request for the URL http://web.grupo4.ultrared.com/. The bytes pane shows the raw HTTP response message, including headers like Date, Server, Last-Modified, ETag, Accept-Ranges, Vary, Content-Encoding, and Content-Length, followed by the file data (HTML content). The bottom status bar indicates the capture is live.

Capturing from Ethernet

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
+ 210	12.957338	192.168.1.48	192.168.1.24	HTTP	596	GET / HTTP/1.1
+ 212	12.960735	192.168.1.24	192.168.1.48	HTTP	738	HTTP/1.1 200 OK (text/html)

```
\t</head>\n\t<body>\n\t<h1>\n\t\tMichel Succar Medina - m.succar@uniandes.edu.co\n\t</h1>\n\t<br>\n\t<h1>\n\t\tJuan Esteban Mendez - je.mendez@uniandes.edu.co\n\t</h1>\n\t<br>\n\t<br>\n\t<br>\n\t\n\t\n\t\n\t\n\t\n\t\n\t<br>\n\t<br>\n\t<video width="260" height="180" controls autoplay>\n\t<source src=".//vid/video-1.mp4" type="video/mp4">\n\t</video>\n\t<video width="260" height="180" controls autoplay>\n\t<source src=".//vid/video-2.mp4" type="video/mp4">\n\t</video>\n\t<video width="260" height="180" controls autoplay>\n\t<source src=".//vid/video-3.mp4" type="video/mp4">\n\t</video>\n
```

Frame (738 bytes) Uncompressed entity body (1093 bytes)

Ethernet: live capture in progress>

Packets: 2651 · Displayed: 2 (0.1%)

Profile: Default

Capturing from Ethernet

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

No.	Time	Source	Destination	Protocol	Length	Info
210	12.957338	192.168.1.48	192.168.1.24	HTTP	596	GET / HTTP/1.1
212	12.960735	192.168.1.48	192.168.1.24	HTTP	738	HTTP/1.1 200 OK (text/html)

```

\t<br>\n
\t<h1>\n
\tJuan Esteban Mendez - je.mendez@uniandes.edu.co\n
\t</h1>\n
\t<br>\n
\t<br>\n
\t<br>\n
\t\n
\t\n
\t\n
\t\n
\t\n
\t\n
\t<br>\n
\t<br>\n
\t<video width="260" height="180" controls autoplay>\n
\t<source src=".vid/video-1.mp4" type="video/mp4">\n
\t</video>\n
\t<video width="260" height="180" controls autoplay>\n
\t<source src=".vid/video-2.mp4" type="video/mp4">\n
\t</video>\n
\t<video width="260" height="180" controls autoplay>\n
\t<source src=".vid/video-3.mp4" type="video/mp4">\n
\t</video>\n
\t<video width="260" height="180" controls autoplay>\n
\t<source src=".vid/video-4.mp4" type="video/mp4">\n
\t</video>\n
\t</body>\n
</html>\n

```

Frame (738 bytes) Uncompressed entity body (1093 bytes)

Ethernet: <live capture in progress>

||| Packets: 2888 · Displayed: 2 (0.1%)

6 Análisis del protocolo HTTPS sobre youtube.com

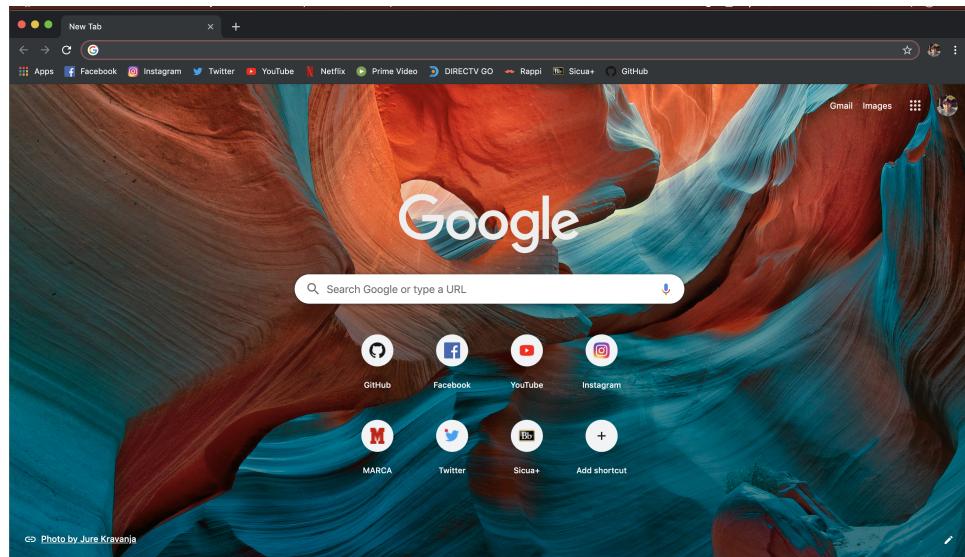
Para analizar el protocolo HTTPS se utilizó como cliente un *Macintosh* con navegador *Google Chrome*. A continuación los pasos realizados:

1. En primera instancia, se borró la caché de resolución de DNS:

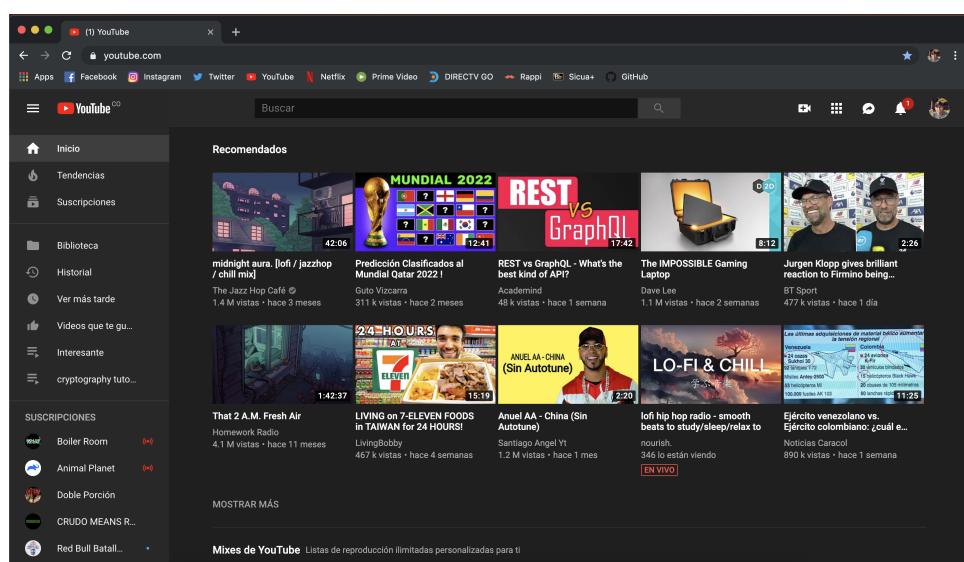


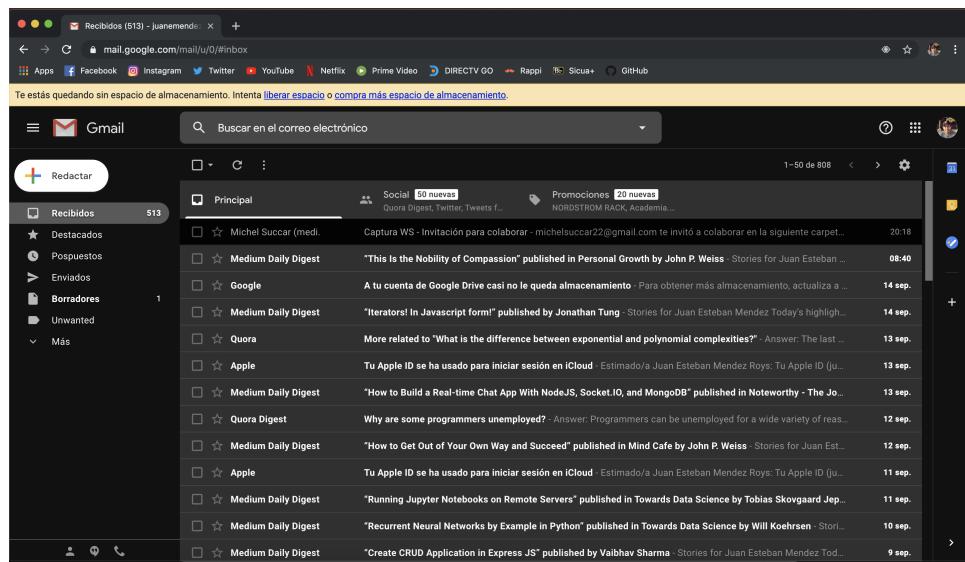
```
Last login: Sun Sep 15 19:41:36 on ttys000
Hi Juan! Welcome back to your terminal :) It's always nice having you back.
(base) juanestebanmendez@Juans-MacBook-Pro-3:~$ sudo killall -HUP mDNSResponder
>Password:
(base) juanestebanmendez@Juans-MacBook-Pro-3:~$
```

2. Se abrió el navegador con una única pestaña para poder tomar la captura de red más limpia posible:

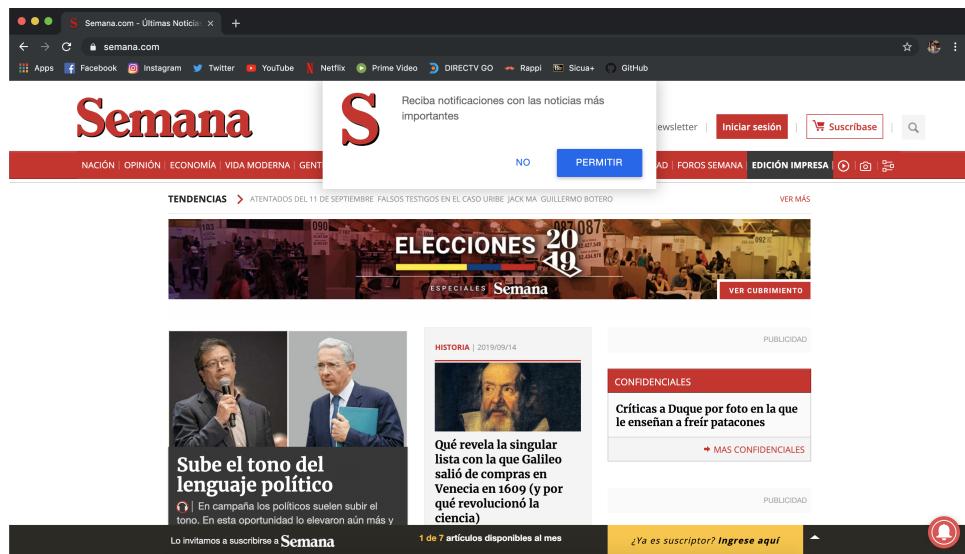


3. Reiniciamos el modo de captura de *WireShark* e ingresamos a www.youtube.com. Se navegaron por diferentes canales y se revisó la bandeja de entrada de *Gmail*:





4. Reiniciamos el modo de captura de *WireShark* y navegamos a páginas web de noticias como www.eltiempo.com y www.semana.com:



5. Observamos los paquetes capturados en *WireShark*:

WireShark screenshot showing network traffic analysis:

No.	Time	Source	Destination	Protocol	Length	Info
101	4.928462	172.20.10.9	172.217.28.110	UDP	70	58715 → 443 Len=28
102	4.928540	172.20.10.9	172.217.28.110	UDP	70	58715 → 443 Len=28
103	4.928596	172.217.28.110	172.20.10.9	UDP	917	443 → 58715 Len=875
104	4.928615	172.20.10.9	172.217.28.110	UDP	70	58715 → 443 Len=28
105	4.928691	172.20.10.9	172.217.28.110	UDP	70	58715 → 443 Len=28
106	4.928762	172.20.10.9	172.217.28.110	UDP	70	58715 → 443 Len=28
107	4.947734	172.20.10.9	172.20.10.1	DNS	74	Standard query 0xaf51 A www.google.com
108	4.955771	50.97.172.202	172.20.10.9	TCP	66	443 → 60717 [ACK] Seq=1 Ack=1 Win=139 Len=0 TSval=1282348526 TSecr=1392244353 TSval=1282348526 TSecr=1392244353
109	4.955825	172.20.10.9	50.97.172.202	TCP	54	[TCP ACKed unseen segment] 60717 → 443 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=1282348526 TSecr=1392244353
110	4.956584	172.20.10.9	172.217.30.202	UDP	430	58858 → 443 Len=388
111	5.091385	172.20.10.1	172.20.10.9	DNS	90	Standard query response 0xaf51 A www.google.com A 172.217.28.100
112	5.091390	172.217.30.202	172.20.10.9	UDP	73	443 → 58858 Len=31
113	5.095144	172.20.10.9	172.217.28.100	UDP	1392	56029 → 443 Len=1350
114	5.096181	172.20.10.9	172.217.28.100	UDP	1392	56029 → 443 Len=1350

Frame details for frame 104:

- Frame 104: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
- Ethernet II, Src: Apple_79:63:6f (8c:85:90:79:63:6f), Dst: f6:dc:cd:1b:27:64 (f6:dc:cd:1b:27:64)
- Internet Protocol Version 4, Src: 172.20.10.9, Dst: 172.217.28.110
- User Datagram Protocol, Src Port: 58715, Dst Port: 443

Selected bytes and hex dump for frame 104:

```

0000  f6 dc cd 1b 27 64 8c 85 90 79 63 6f 08 00 45 00  . . . 'd' . . . yco . E .
0010  00 38 58 1d 00 00 40 11 a3 33 ac 14 0a 09 ac d9  . 8x . @ . 3 . . .
0020  1c 6e e5 5b 01 bb 00 24 42 9c 40 a6 31 0f 30 66  . n [ . . $ B @ 1 0f
0030  d4 a1 4a 0a 9a 9e 52 ed 96 34 3a 11 ba e8 ce fa  . . j . R . 4 : . . .
0040  61 45 79 bf 73 0c aEy . s .

```

7 Preguntas

1. ¿Es posible garantizar el envío confiable en un Streaming de video y audio? Si no es así, ¿Qué problemas generaría hacerlo de esta forma? ¿Es posible agregar mecanismos de seguridad como el cifrado al streaming de voz y video?

Sí es posible garantizar envío confiable en un streaming de video y en audio, mediante el uso de mecanismos como la encriptación. Se puede utilizar TLS (*Transport Layer Secure*), el cual proporciona una comunicación segura a través de HTTP. Todas las medidas de seguridad tienen que ser implementadas como *patches* sobre la capa de aplicación, debido a que los protocolos de esta capa como lo es HTTP no fueron creados con la seguridad en mente.

Si no fuera posible agregar una capa de seguridad al audio y a un streaming de video, existirían problemas de seguridad como lo son: *man in the middle*, *sniffing* y *spoofing*. Existen mecanismos de cifrado, que utilizan *wave files*, los cuales son archivos que contienen información de audio y video y aplican algoritmos sobre ellos. Entre los algoritmos utilizados se encuentran: DES, Blowfish, RC4, RC2, RC6 y AES.

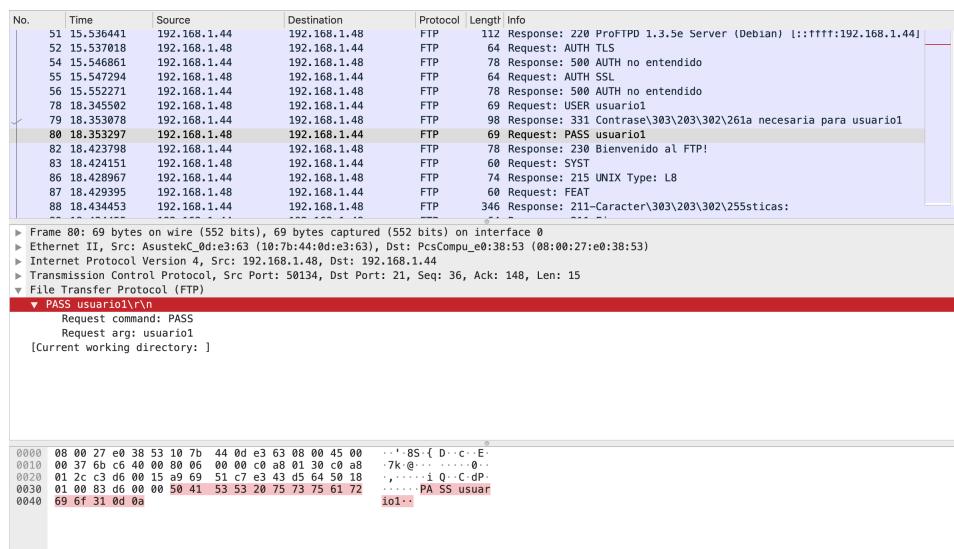
2. Explique qué ocurre si desde un PC cliente se intenta hacer ping a la URL de uno de los servidores, pero dicho cliente tiene configurada la IP de forma estática, pero no le fue definida la dirección IP de servidor DNS.

Lo que ocurriría es que el PC cliente al momento de hacer ping a la URL de alguno de los servidores, no sabría como mapear la URL a alguna dirección IP. Saldría algo así como *host unreachable* en el *command prompt*. Por ello siempre es necesario que el cliente conozca la dirección IP del servidor DNS, para que así cada una de las peticiones que haga por medio de un nombre dominio, este se encargue de mapearlas a su dirección IP correspondiente.

3. Es posible ver durante la autenticación en la captura de tráfico de wireshark, el nombre de usuario y contraseña de un usuario en el servidor FTP, ¿A qué se debe? Encuentre evidencia que sustenta el enunciado.

Es posible ver el nombre de usuario y la contraseña durante la autenticación en el servidor FTP. Esto es debido a que el protocolo FTP por si solo no provee ningún tipo de capa de seguridad. Es por ello que es necesario aplicar un *patch* de seguridad con mecanismos como SSL y TSL, para que el intercambio de archivos sea seguro.

A continuación se muestra la evidencia en los paquetes analizados en *WireShark*, de que la contraseña del usuario es enviada en texto plano mediante el protocolo FTP a través de la red:



4. Según la captura de tráfico. ¿Qué protocolos de red que han sido desarrollados en las prácticas de laboratorio participan en el intercambio de datos generados al ingresar al sitio web Youtube? Consultelos y explíquelos brevemente.

Según la captura de tráfico, se pudieron observar protocolos de red como: UDP y TCP. Igualmente se pueden ver capas de seguridad como TLS. En el siguiente *screenshot* podemos observar los protocolos escaneados en *WireShark*:

No.	Time	Source	Destination	Protocol	Length	Info
189..	44.787835	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=550568 Win=142080 Len=0 TSval=11115
189..	44.787835	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=551673 Win=140992 Len=0 TSval=11115
189..	44.788101	216.58.222.229	172.20.10.9	TLSv1...	1247	Application Data
189..	44.788105	216.58.222.229	172.20.10.9	TLSv1...	1321	Application Data
189..	44.788107	216.58.222.229	172.20.10.9	TLSv1...	1454	Application Data
189..	44.788157	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=552854 Win=142144 Len=0 TSval=11115
189..	44.788170	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=554109 Win=140928 Len=0 TSval=11115
189..	44.788177	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=555497 Win=139520 Len=0 TSval=11115
189..	44.788245	172.217.204.189	172.20.10.9	UDP	62	443 → 49559 Len=20
189..	44.788296	172.20.10.9	216.58.222.229	TCP	66	[TCP Window Update] 60821 → 443 [ACK] Seq=45155 Ack=555497 Win=1433
189..	44.789097	216.58.222.229	172.20.10.9	TLSv1...	998	Application Data
189..	44.789103	216.58.222.229	172.20.10.9	TLSv1...	1454	Application Data
189..	44.789176	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=556429 Win=142400 Len=0 TSval=11115
190..	44.789176	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=557817 Win=141056 Len=0 TSval=11115
190..	44.792381	216.58.222.229	172.20.10.9	TLSv1...	1454	Application Data
190..	44.792385	216.58.222.229	172.20.10.9	TLSv1...	1454	Application Data
190..	44.792386	216.58.222.229	172.20.10.9	TLSv1...	1063	Application Data
190..	44.792450	172.20.10.9	216.58.222.229	TCP	66	60821 → 443 [ACK] Seq=45155 Ack=559205 Win=141952 Len=0 TSval=1111909222

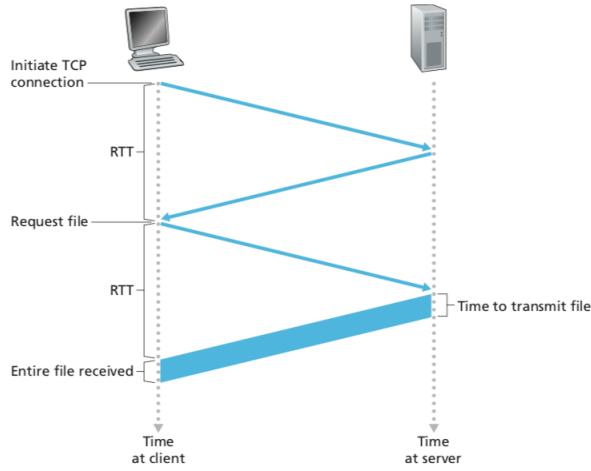
- **TCP:** El modelo de servicio TCP es orientado a conexión y a un intercambio de datos confiable. Es utilizado en aplicaciones que tienen no pueden soportar ninguna perdida de datos, como por ejemplo: transferencia de archivos, *e-mail*, *web documents*, o mensajería en *smartphones*.
- **UDP:** UDP es un protocolo de transporte ligero, que provee servicios minimalistas. UDP es *connectionless*, es decir que no hay ningún tipo de protocolo de *handshake* entre los dos procesos para que se empiecen a comunicar. UDP provee una transferencia de datos no confiable. UDP no asegura que el mensaje que se quiera enviar llegará a su destino final. De igual manera, los mensajes pueden llegar en un orden diferente a el que se enviaron. Es frecuentemente utilizado en servicios de *audio* y de *streaming*.

5. ¿Identifica tráfico HTTP y HTTPS generado al ingresar al sitio web? ¿El tráfico HTTP es significativo frente al tráfico HTTPS? ¿Identifica qué componentes o información de su navegación en el portal web generó este tráfico HTTP?

Sí se pudo identificar tráfico HTTP y HTTPS al ingresar al sitio web. Muchas páginas oficiales como las visitadas, es decir www.eltiempo.com, www.youtube.com y www.semana.com cuentan con implementación del protocolo HTTPS. Esto es debido a que todas estas entidades cuentan con un *digital certificate* el cual nos cerciora de que la información que viaja entre el *path* de mi computador hasta el servidor está completamente encriptada. Cada vez que se ingresaba a un nuevo *link*, se generaba tráfico HTTP.

6. Para el protocolo HTTPS: Explique gráficamente y con sus palabras el proceso de handshake. Encuentra e inspeccionar los detalles del intercambio de certificados, incluyendo la expansión del bloque de protocolo de enlace dentro de la TLS Record. Al igual que los mensajes "Hello", el contenido del mensaje de certificado es visible, ¿A qué se debe esto?

El proceso de *handshake* para el protocolo HTTPS empezaría con una conexión por TCP. El cliente le pregunta al servidor por su *digital certificate*. El servidor envía su *digital certificate* por el canal. El cliente abre el certificado y se apropiá de la llave pública. El cliente verifica con la CA (*Certificate Authority*) la veracidad del certificado y por ende de la llave pública. Una vez haya comprobado que si es la misma que tiene la CA, el cliente procede a hacer su respectiva petición HTTP.



El contenido del certificado es visible debido a que esa información es pública. Esta hecha para que cualquier cliente o persona pueda mirarla y comprobar la veracidad de la entidad que provee el servicio web.

No.	Time	Source	Destination	Protocol	Length	Info
22	1.958379	172.20.10.9	172.217.30.202	TLSv1...	152	Application Data
23	1.958532	172.20.10.9	172.217.30.202	TLSv1...	105	Application Data
24	1.958532	172.20.10.9	172.217.30.202	TLSv1...	195	Application Data
48	2.596484	172.20.10.9	172.217.30.202	TLSv1...	101	Application Data
619	10.265327	172.20.10.9	17.248.184.116	TLSv1...	97	Encrypted Alert
1381	16.457201	200.14.44.115	172.20.10.9	TLSv1...	97	Encrypted Alert
1435	16.943837	172.20.10.9	40.97.125.130	TLSv1...	85	Encrypted Alert
6771	27.677018	147.75.78.123	172.20.10.9	TLSv1...	112	[TCP Previous segment not captured] , Application Data
6774	27.677978	147.75.78.123	172.20.10.9	TLSv1...	97	Encrypted Alert
6856	29.170834	172.20.10.9	17.248.184.116	TLSv1...	97	Encrypted Alert
155..	36.136726	172.20.10.9	172.217.30.206	TLSv1...	583	Client Hello
155..	36.298377	172.217.30.206	172.20.10.9	TLSv1...	1454	Server Hello, Change Cipher Spec
155..	36.299719	172.217.30.206	172.20.10.9	TLSv1...	1070	Application Data
155..	36.301013	172.20.10.9	172.217.30.206	TLSv1...	130	Change Cipher Spec, Application Data
155..	36.416298	172.217.30.206	172.20.10.9	TLSv1...	630	Application Data, Application Data
155..	36.707027	172.20.10.9	172.217.30.202	TLSv1...	152	Application Data
155..	36.707091	172.20.10.9	172.217.30.202	TLSv1...	105	Application Data
155..	36.707091	172.20.10.9	172.217.30.202	TLSv1...	226	Application Data
155..	36.798336	172.217.30.202	172.20.10.9	TLSv1...	105	Application Data
171..	37.330373	172.20.10.9	172.217.30.202	TLSv1...	101	Application Data
172..	37.701190	172.20.10.9	172.217.28.101	TLSv1...	583	Client Hello
172..	37.937098	172.20.10.9	216.58.222.229	TLSv1...	643	Client Hello
172..	38.010224	172.217.28.101	172.20.10.9	TLSv1...	1454	Server Hello, Change Cipher Spec

Frame 15520: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface 0
 ▶ Ethernet II, Src: Apple_79:63:6f (8c:85:90:79:63:6f), Dst: f6:dc:cd:1b:27:64 (f6:dc:cd:1b:27:64)
 ▶ Internet Protocol Version 4, Src: 172.20.10.9, Dst: 172.217.30.206
 ▶ Transmission Control Protocol, Src Port: 60819, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
 ▶ Transport Layer Security
 ▼ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
 Content Type: Handshake (22)
 Version: TLS 1.0 (0x0301)
 Length: 512
 ▶ Handshake Protocol: Client Hello

Podemos expandir el mensaje *client hello* y observar los detalles. Podemos ver como el servidor intercambia su certificado:

```

▼ Transport Layer Security
  ▼ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
  ▼ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 508
    Version: TLS 1.2 (0x0303)
    Random: 5366cfb3974ec14b2fa2374d6561d44ab47bf2562802f2c1...
    Session ID Length: 32
    Session ID: b66cbaf794cde9cae0b249636b96f697fbb487c96cc2f62...
    Cipher Suites Length: 34
    ▶ Cipher Suites (17 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 401
      ▷ Extension: Reserved (GREASE) (len=0)
      ▷ Extension: server_name (len=24)
      ▷ Extension: extended_master_secret (len=0)
      ▷ Extension: renegotiation_info (len=1)
      ▷ Extension: supported_groups (len=10)
      ▷ Extension: ec_point_formats (len=2)
      ▷ Extension: session_ticket (len=0)
      ▷ Extension: application_layer_protocol_negotiation (len=14)
      ▷ Extension: status_request (len=5)
      ▷ Extension: signature_algorithms (len=20)
      ▷ Extension: signed_certificate_timestamp (len=0)
      ▷ Extension: key_share (len=43)
      ▷ Extension: psk_key_exchange_modes (len=2)
      ▷ Extension: supported_versions (len=11)
      ▷ Extension: compress_certificate (len=3)
      ▷ Extension: Reserved (GREASE) (len=1)

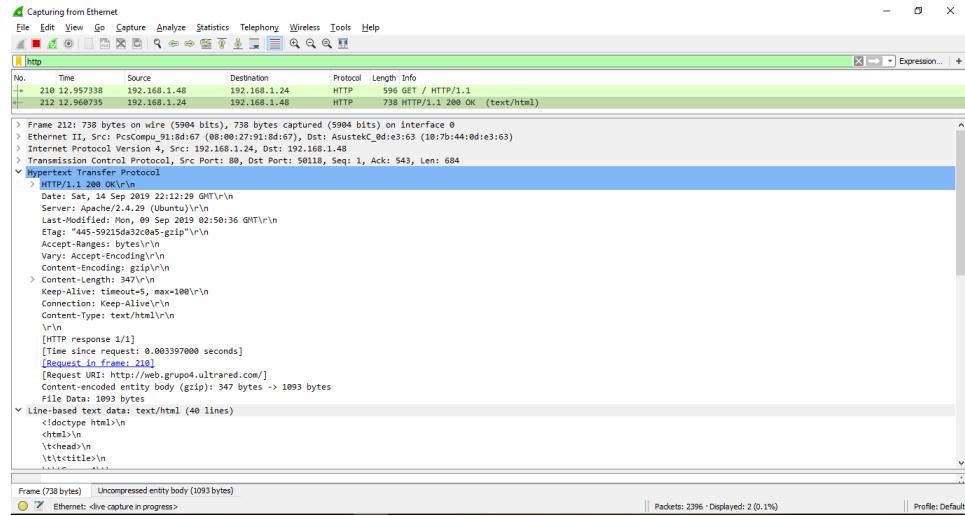
```

7. ¿Quién envía el certificado, el cliente, el servidor, o ambos? Un certificado es enviado por una de las partes para que la otra parte autentique que es quien dice ser. Sobre la base de este uso, usted debería ser capaz de reconocer con Wireshark quién envía el certificado y comprobar los mensajes en su rastro.

El *digital certificate* es enviado por el servidor al cliente. El *digital certificate* sirve como un contenedor para una llave pública (en este caso la del servidor). Para que el cliente se pueda cerciorar de que la llave pública si es la del servidor, el *digital certificate* del servidor es firmado por una CA (*Certificate Authority*). Las entidades certificadoras cuentan con un gran prestigio y son las encargadas de emitir los certificados de diferentes servidores de diferentes páginas web. La llave que es recibida es comparada con la que la CA tiene para comprobar su veracidad.

8. ¿Puede encontrarse información adicional sobre los servicios prestados en la topología usando la captura de paquetes? Por ejemplo, ¿sería posible encontrar la ubicación de los recursos de contenido? Explicar.

En las capturas de paquetes se pueden encontrar muchos diferentes tipos de información sobre los servicios prestados. Por ejemplo, cuando se hace captura de un protocolo HTTP y HTTPS se puede ver el contenido del archivo html que es enviado en los envíos de paquetes del protocolo. Esto puede dar información acerca del tipo de archivos que hay en la página web visitada, por ejemplo, que formato de vídeo se usan, o que formato tienen las imágenes de la página y si hay enlaces de redirecciónamiento en la página o no.

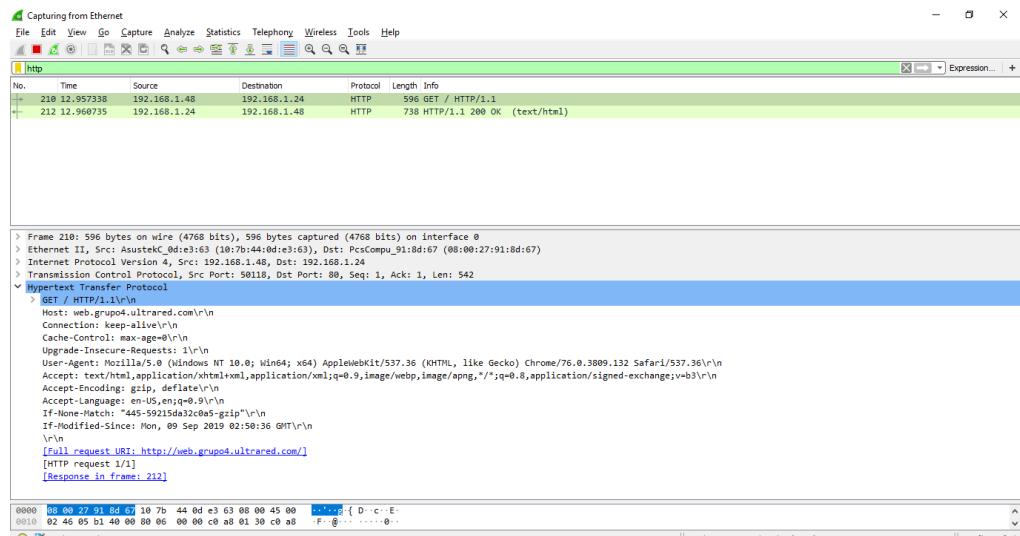


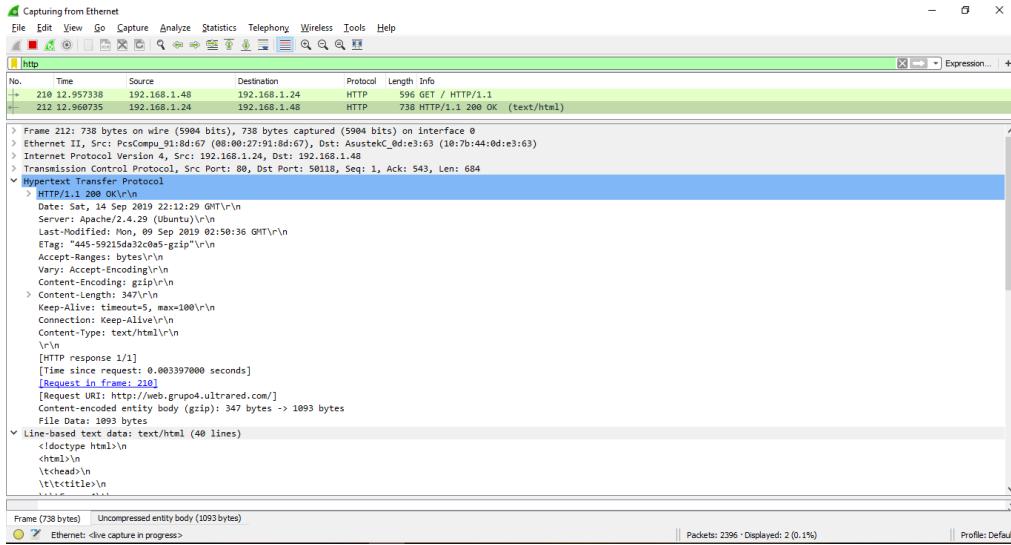
9. ¿Dentro de los paquetes capturados, hay frames cuya dirección IP no sea la dirección del cliente a la que corresponde la interfaz de red? Si es así, ¿Por qué sucede esto?

Si hay frames cuya dirección IP no sea la dirección del cliente a la que corresponde la interfaz de red. La primera que se puede ver es la IP del servidor DNS. Cada vez que se establece la conexión, el cliente envía una petición a través de este servidor. De igual manera, por ejemplo en la captura del servicio FTP, notamos la dirección IP 224.0.0.252. Expandimos el frame en *WireShark* y pudimos observar que el intercambio de paquetes con esta dirección era utilizando el protocolo LLMNR (*Link-Local Multicast Name Resolution*). Este protocolo permite que los *hosts* realicen la resolución de nombre para otros *hosts* en el mismo enlace local, es decir la misma topología.

10. Para capturas de un mismo protocolo, ¿hay diferencias significativas entre los paquetes respecto a su estructura? Explicar y mostrar capturas que respalden la respuesta.

Claramente en los paquetes que se envían dentro de un mismo protocolo tendrán estructuras diferentes, un protocolo tendrá paquetes de solicitud y paquetes de respuesta. Estos paquetes tendrán estructuras diferentes ya que los paquetes de solicitud la mayoría del tiempo no cargarán objetos, como por ejemplo las solicitudes de HTTP y HTTPS, mientras que las respuestas la mayoría de las veces cargarán con los objetos solicitados.





11. ¿Por qué Wireshark muestra la dirección MAC vigente del enrutador local, pero no la dirección MAC vigente de los servidores remotos al realizar una comunicación directa a ellos?

Wireshark muestra la dirección MAC vigente del enrutador local pues esta es una dirección que no se puede cambiar y comprueba en que red se hizo el sniffing si se sabe a que maquina pertenece la dirección. Por otro lado las direcciones MAC de los servidores remotos no serán descriptivas de la red en la que se encuentran mientras que el enrutador si.