

Práctica Bloque III

Alumno: Navarro Jimena, David

Titulación: Grado de Ingeniería Informática

Ejercicio 1. ¿Cuál es el puerto utilizado por el servidor? ¿Es el normal de HTTP (80)? ¿Por qué?

- Tramas analizadas: 719

Utiliza el puerto 443, uno distinto al normal de HTTP porque usa HTTPS. El puerto 443 es un puerto también para navegación web usando el protocolo HTTPS y TLS por debajo

The image shows a Wireshark packet capture of an HTTP GET request. The packet list at the top shows a GET request to /api/people/32/ on port 443. The packet details pane shows the Transmission Control Protocol (TCP) segment with Source Port 443 and Destination Port 53451. The packet bytes pane shows the raw data of the request, including the HTTP GET method and the URL.

No.	Time	Source	Destination	Protocol	Length	Info
719	14.717362	52.58.110.120	192.168.18.15	HTTP/TLS	835	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
720	14.719733	192.168.18.15	52.58.110.120	HTTP	201	GET /api/people/32/ HTTP/1.1
721	14.782794	52.58.110.120	192.168.18.15	HTTP/TLS	858	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
722	14.786698	192.168.18.15	52.58.110.120	HTTP	202	GET /api/planets/28/ HTTP/1.1
723	14.844417	52.58.110.120	192.168.18.15	HTTP/TLS	928	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
724	14.847379	192.168.18.15	52.58.110.120	HTTP	201	GET /api/people/10/ HTTP/1.1
725	14.922942	52.58.110.120	192.168.18.15	HTTP/TLS	1227	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
726	14.927234	192.168.18.15	52.58.110.120	HTTP	202	GET /api/planets/20/ HTTP/1.1
727	14.981626	52.58.110.120	192.168.18.15	HTTP/TLS	803	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
889	19.372633	192.168.18.15	52.58.110.120	HTTP	205	GET /api/starships HTTP/1.1
909	19.602854	52.58.110.120	192.168.18.15	HTTP/TLS	732	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)
911	19.606698	192.168.18.15	52.58.110.120	HTTP	204	GET /api/starships/33/ HTTP/1.1
922	19.732638	52.58.110.120	192.168.18.15	HTTP/TLS	405	HTTP/1.1 404 NOT FOUND, JavaScript Object Notation (application/json)
937	19.830208	192.168.18.15	52.58.110.120	HTTP	203	GET /api/starships/3/ HTTP/1.1
947	19.898457	52.58.110.120	192.168.18.15	HTTP/TLS	1003	HTTP/1.1 200 OK, JavaScript Object Notation (application/json)

Total Length: 821
Identification: 0x904a (36938)
> 010. = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 231
Protocol: TCP (6)
Header Checksum: 0x8ae6 [validation disabled]
[Header checksum status: Unverified]
Source Address: 52.58.110.120
Destination Address: 192.168.18.15
Transmission Control Protocol, Src Port: 443, Dst Port: 53451, Seq: 17276, Ack: 2005, Len: 781
Source Port: 443
Destination Port: 53451
[Stream index: 14]
[Conversation completeness: Complete, WITH_DATA (47)]
[TCP Segment Len: 781]
Sequence Number: 17276 (relative sequence number)

0020 12 0f 01 bb d0 cb 45 d9 bd ba 50 90 fd 60 50 18E...P...P...
0030 01 2f a2 39 00 00 17 03 03 03 08 57 54 b4 1c b0 .../...NT...
0040 36 e6 ca 37 80 dc 5c 8e 2f f3 de f5 3e 04 22 51 6.../.../...Q...
0050 48 de 77 b6 68 fa fd 9d e7 13 23 92 d3 17 0f ea H...h...#...
0060 0b 24 47 55 a8 b8 6c 30 85 ef aa b1 f8 e2 9e be \$GU...10...
0070 94 36 12 13 5a 4a 59 47 5d 13 ac 39 a6 7f 5e 63 6...ZJYG...9...c...
0080 51 ee c9 40 44 64 21 2a 73 7c a2 be 27 7b 13 59 Q...@d!* s|...{...Y...
0090 3c 5b 01 b8 0e f4 02 a0 11 d0 90 14 82 af ba a1 <[...
00a0 e6 3f 92 d3 72 f4 6d 47 4d 34 b7 4e d2 00 ad 95 ?...mG H4-W...
00b0 3a 26 1d 76 7e 60 7c 2a fd 91 6f 13 16 34 74 a6 :&w...|*...o...4t...
00c0 b5 32 de 5e bb 50 fe 5d 05 59 5e 8d 76 85 7b 54 2...^P...Y...v...{T...
00d0 8f fc f5 aa b0 bf 5c 11 db b7 e8 6e dc 4b 14 f0V...n...K...
00e0 2a 13 bf f9 18 39 c8 e8 33 f2 6b 70 dc 65 e3 86 *...9...3...kp...
00f0 a7 f2 63 83 82 87 a6 63 cb 03 e2 7f 15 56 5a 83 ...c...c...VZ...
0100 2b 44 9f e3 e0 bf 2c 5f fa 5c c3 7b 1a a0 75 af +D...,...\...{...u...
0110 90 bd 02 78 e1 04 da c7 f4 d6 02 47 40 d9 08 c0 ...x...@...
0120 6b 77 c2 a6 c7 bc b9 97 38 d6 53 5b 2c 20 0d 3a kw...8-S[,...:
0130 42 84 9d 37 4e 59 47 96 31 b5 6f fd a0 f7 67 d5 B...7NYG...1...g...

Frame (835 bytes) Decrypted TLS (752 bytes) De-chunked entity body (413 bytes)

Ejercicio 2. Observe el número de conexiones realizadas. ¿Cuántas hace? ¿Usa una conexión permanente (en la misma conexión hace varias peticiones) o no permanente (solo realiza una por conexión)? En caso de ser permanente, ¿qué cabecera de la petición indica que queremos que sea permanente?

- Tramas analizadas: 719

Realiza una sola conexión. Usa una conexión permanente al hacer varias request por conexión. Para que sea permanente tenemos que añadir en la cabecera el argumento keep-alive. Así que mientras hagamos request con frecuencia, no deberíamos tener que volver a crear la conexión.

The screenshot shows a Wireshark capture of an HTTP GET request and its response. The request is highlighted in red, and the response is highlighted in blue. The 'Connection: keep-alive' header is circled in red in the request. The response status is 200 OK.

Request (Red):

```
GET /api/people HTTP/1.1
Accept: application/json
User-Agent: David-2023
Host: swapi.dev
Connection: keep-alive
```

Response (Blue):

```
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Tue, 30 May 2023 14:44:20 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN
ETag: "b493126da505af6fec015ec116fec193"
Allow: GET, HEAD, OPTIONS
Strict-Transport-Security: max-age=15768000
```

Ejercicio 3. Describa el significado de las cabeceras de una petición y una respuesta (sin incluir las X-*).

- Tramas analizadas: 719

```
GET /api/people HTTP/1.1
Accept: application/json
User-Agent: David-2023
Host: swapi.dev
Connection: keep-alive

HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Tue, 30 May 2023 14:44:20 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN
ETag: "b493126da505af6fec015ec116fec193"
Allow: GET, HEAD, OPTIONS
Strict-Transport-Security: max-age=15768000
```

En rojo tenemos la cabecera de la petición donde vemos el método HTTP, en este GET, y a que llamamos. La versión de HTTP, que tipo de datos aceptamos con Accept, en este caso json, el usuario que hace la llamada, el host y por último, que queremos mantener la conexión viva.

En azul tenemos la respuesta. Donde podemos ver primero la versión de HTTP y estado de la request. Es este 200 así que todo bien. Después vemos el servidor, la fecha, el tipo de contenido que tiene que ser igual al pedido por la request. El tipo de codificación para la transmisión y el argumento de la conexión. Vary sirve para hacer coincidir los encabezados de solicitudes futuras para ver si se puede usar la cache. Tras esto tenemos X-Frame-Options es una opción para saber si el navegador debería renderizar esos datos, en este caso solo se mostraría si todos los frames tienen el mismo origen que la página. Ahora tenemos ETag que es un identificador para una versión del recurso y que la cache sea más eficiente. Por último tenemos Allow que nos dice que métodos HTTP están permitidos y Strict-Transport-Security donde podemos poner opciones para el transporte

Ejercicio 4. Filtre por el protocolo rtsp y use la opción **Follow TCP Stream** de Wireshark para observar el diálogo completo que han mantenido el cliente de correo y el servidor. Explique brevemente (una línea) el significado de cada comando enviado por el cliente (si algún comando se repite solo debe explicarlo una vez).

- Tramas analizadas:8,11,23,37,1617,6727

```
OPTIONS rtsp://rtsp.stream:554/movie RTSP/1.0
CSeq: 2
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
```

Con options podemos indicar la URI del archivo multimedia que queremos

```
DESCRIBE rtsp://rtsp.stream:8554/bunny.mkv RTSP/1.0
CSeq: 4
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Accept: application/sdp
```

Con este método podemos conseguir una descripción del archivo

```
SETUP rtsp://23.88.67.97:8554/bunny.mkv/track1 RTSP/1.0
CSeq: 5
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Transport: RTP/AVP;unicast;client_port=51500-51501
```

Con este comando podemos indicar el tipo de mecanismo de transporte para el archivo indicado

```
PLAY rtsp://23.88.67.97:8554/bunny.mkv/ RTSP/1.0
CSeq: 7
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Session: 1352FCCC
Range: npt=0.000-
```

Con este indicamos al servidor que comience a enviar los datos

```
PAUSE rtsp://23.88.67.97:8554/bunny.mkv/ RTSP/1.0
CSeq: 8
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Session: 1352FCCC
```

Con este pausamos la transmisión

```
TEARDOWN rtsp://23.88.67.97:8554/bunny.mkv/ RTSP/1.0
CSeq: 12
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Session: 1352FCCC
```

Y por último, con teardown hacemos que pare la transmisión

Ejercicio 5. ¿Por qué se hacen dos comandos SETUP? ¿Cómo sabía que debía hacer dos comandos de ese estilo?

- Tramas analizadas: 23, 26, 20

Realiza dos comandos SETUP para indicar los parámetros de transporte de cada una de las tracks del video, además de indicar en el segundo la sesión que está usando el cliente. Sabe que tiene que hacer eso ya que en DESCRIBE se muestra que hay 2 tracks.

```
v=0
o=- 1654711921171645 1 IN IP4 23.88.67.97
s=Matroska video+audio+(optional)subtitles, streamed by RTSP.Stream
i=bunny.mkv
t=0 0
a=tool:LIVE555 Streaming Media v2021.11.01
a=type:broadcast
a=control:*
a=range:npt=0-634.642
a=x-qt-text-nam:Matroska video+audio+(optional)subtitles, streamed by RTSP.Stream
a=x-qt-text-inf:bunny.mkv
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
b=AS:500
a=rtpmap:96 H264/90000
a=fmtp:96 packetization-mode=1;profile-level-id=64001F;sprop-parameter-sets=Z2QA6zZgUH7DmoCAGKAAADAIABX5AHjBjN,a015yyLA
a=control:track1
m=audio 0 RTP/AVP 97
c=IN IP4 0.0.0.0
b=AS:96
a=rtpmap:97 MPEG4-GENERIC/48000
a=fmtp:97 streamtype=5;profile-level-id=1;mode=AAC-hbr;sizelength=13;indexlength=3;indexdeltalength=3;config=118856E500
a=control:track2
```

Ejercicio 6. ¿Qué comandos ha provocado adelantar la reproducción del vídeo? ¿Cómo indica por donde debe seguir la reproducción tras el cambio?

- Tramas analizadas: 2945, 2927

Tiene primero que pausar el video y después usar PLAY indicando en el argumento Range el momento del video que queremos.

```
PAUSE rtsp://23.88.67.97:8554/bunny.mkv/ RTSP/1.0
CSeq: 10
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Session: 1352FCCC

RTSP/1.0 200 OK
CSeq: 10
Date: Wed, Jun 08 2022 18:12:32 GMT
Session: 1352FCCC

PLAY rtsp://23.88.67.97:8554/bunny.mkv/ RTSP/1.0
CSeq: 11
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Session: 1352FCCC
Range: npt=342.199-
```

Ejercicio 7. Si observa los comandos y las respuestas son muy similares a las que usa HTTP. Indique dos cabeceras que use RTSP que también se usen en HTTP e indique (y explique) dos cabeceras de RTSP que no se usen en HTTP.

- Tramas analizadas:

En ambos protocolos se usan User-Agent y Accept. Pero en RTSP tenemos Transport que tenemos que usar para indicar el protocolo y el puerto, cosa que no necesitamos en HTTP. Y también tenemos CSeq para indicar el par de respuesta y request para poder saber cuáles van juntas, así aunque lleguen desordenadas se puede saber que respuestas pertenecen a que request.

Ejercicio 8. Ahora filtre por el protocolo rtp que se utiliza para transmitir el recurso multimedia tal cual. ¿Cómo se decidieron los puertos a utilizar en estas comunicaciones RTP? ¿Se confirman de alguna forma cada uno de los envíos RTP?

- Tramas analizadas:43

Se deciden al usar el comando SETUP. No, no se confirman, funciona como una conexión UDP, sería muy lento si se tuvieran que confirmar.

```
SETUP rtsp://23.88.67.97:8554/bunny.mkv/track1 RTSP/1.0
CSeq: 5
User-Agent: LibVLC/3.0.14 (LIVE555 Streaming Media v2016.11.28)
Transport: RTP/AVP;unicast client_port=51500-51501
```