SharkFest '19 Europe

Audio and Video with Wireshark

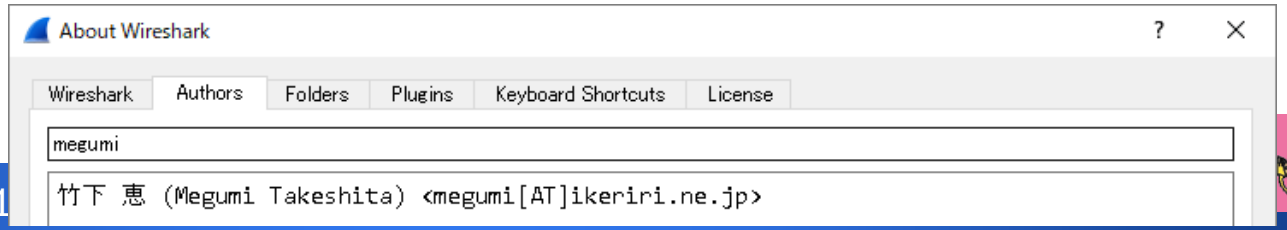
Supplemental files and Tools
http://www.ikeriri.ne.jp/sharkfest/
and official site later

Megumi Takeshita

Packet Otaku, ikeriri network service

Megumi Takeshita, ikeriri network service

- Worked SE/IS at BayNetwork, Nortel
- Founder, ikeriri network service co., ltd
- Reseller of CACE technologies in 2008
- Wrote 10+ books about Wireshark
- Instruct Wireshark to JSDF and other company
- Reseller of packet capture / wireless tools
- One of contributors of Wireshark
  Translate Wireshark into Japanese



Audio and Video with Wireshark

Now we dissect 5 typical audio and video protocols using
#1 Live / On demand streaming (RTMP FLV)
#2 HTTP Live streaming (HLS TS)
#3 VoIP (SIP/RTP)
#4 Surveillance camera (assume port / motion JPEG)
#5 Unknown drive recorder
(assume port, protocol, codec / rtpdump / ffmpeg)

Trace files/tools www.ikeriri.ne.jp/sharkfest/

Live streaming and streaming video

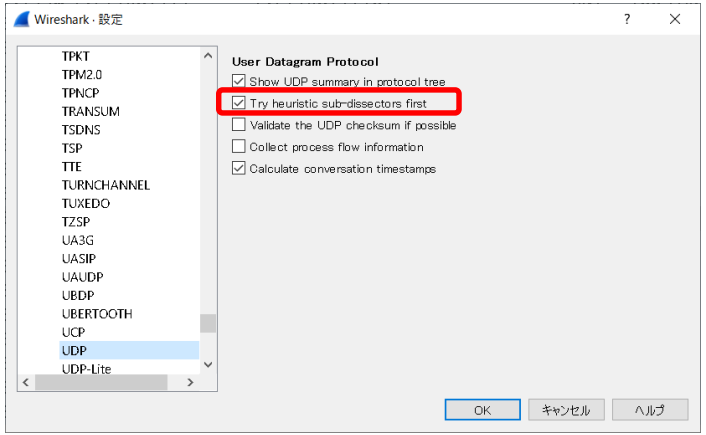
- There are many live or on-demand video services in todays Internet, for example, YouTube, Facebook Live, TikTok and many others.
- They use (or used) RTMP Real-Time Messaging Protocol for broadcasting.
- RTMP is derived from Adobe, and Flash player use them.
- We can download protocol spec https://www.adobe.com/jp/devnet/rtmp.html

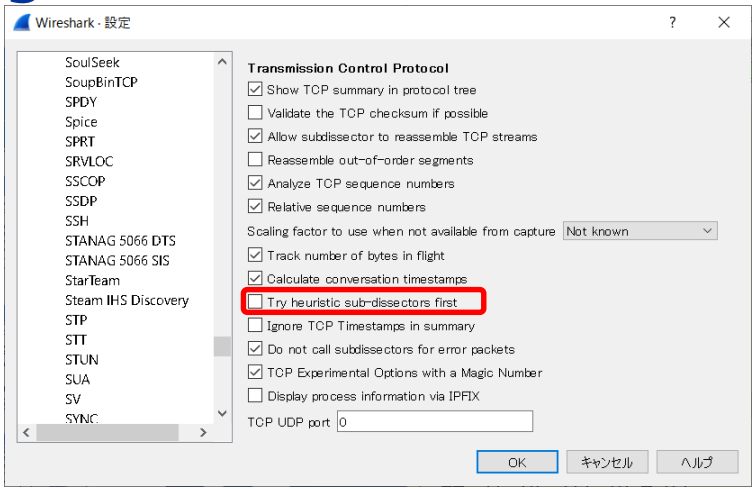
RTMP/RTMPT/RTMPS

- Plain RTMP uses TCP port 1935, but we use web browser to watch the video, so RTMPT (RTMP Tunneled) is used between web server and browser.
- RTMPS uses HTTPS to encrypt RTMP, is known as RTMP over SSL/TLS, and is UDP version of RTMP
- Some web service uses different port number.
  In those case, check “Try heuristic sub-dissectors first” in TCP/UDP preference or manually set “Decode As”

check “Try heuristic sub-dissectors first”

- “Try heuristic sub-dissectors first” means Wireshark try to assume which application protocol used in application layer, and change the dissectors from data.

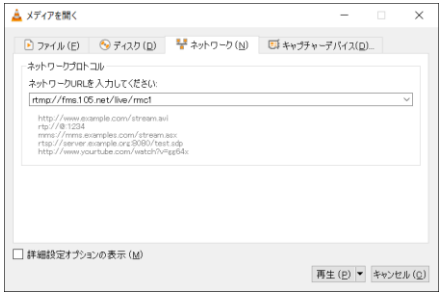




- Or set manually assign the port by “Decode As…”

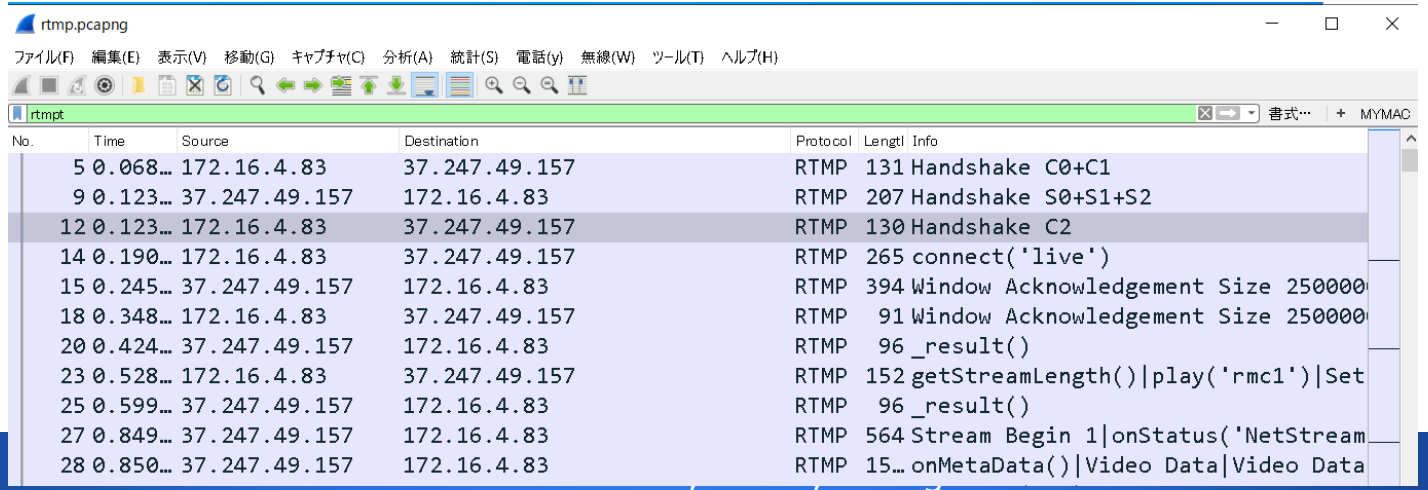
Capture plain RTMP traffic

- First, start capturing using dumpcap below
  dumpcap -i 3 -f “tcp port 1935" -w rtmp.pcapng
- Then, start streaming to open URL
  rtmp://fms.105.net/live/rmc1 by VLC player
- Stop capturing and open rtmp.pcapng
- Note: we need RTMP connection packet, so please start capturing first then play movie



Open rtmp.pcapng

- Wireshark uses “rtmpt” display filter string, because rtmp is already used for Routing Table Maintenance Protocol of Apple Talk protocol family.
- Open rtmp.pcapng and set display filter as “rtmpt”



No.	Time	Source	Destination	Protocol	Lengtl	Info
5	0.068…	172.16.4.83	37.247.49.157	RTMP	131	Handshake C0+C1
9	0.123…	37.247.49.157	172.16.4.83	RTMP	207	Handshake S0+S1+S2
12	0.123…	172.16.4.83	37.247.49.157	RTMP	130	Handshake C2
14	0.190…	172.16.4.83	37.247.49.157	RTMP	265	connect('live')
15	0.245…	37.247.49.157	172.16.4.83	RTMP	394	Window Acknowledgement Size 250000
18	0.348…	172.16.4.83	37.247.49.157	RTMP	91	Window Acknowledgement Size 250000
20	0.424…	37.247.49.157	172.16.4.83	RTMP	96	_result()
23	0.528…	172.16.4.83	37.247.49.157	RTMP	152	getStreamLength()|play('rmc1')|Set
25	0.599…	37.247.49.157	172.16.4.83	RTMP	96	_result()
27	0.849…	37.247.49.157	172.16.4.83	RTMP	564	Stream Begin 1|onStatus('NetStream
28	0.850…	37.247.49.157	172.16.4.83	RTMP	15…	onMetaData()|Video Data|Video Data

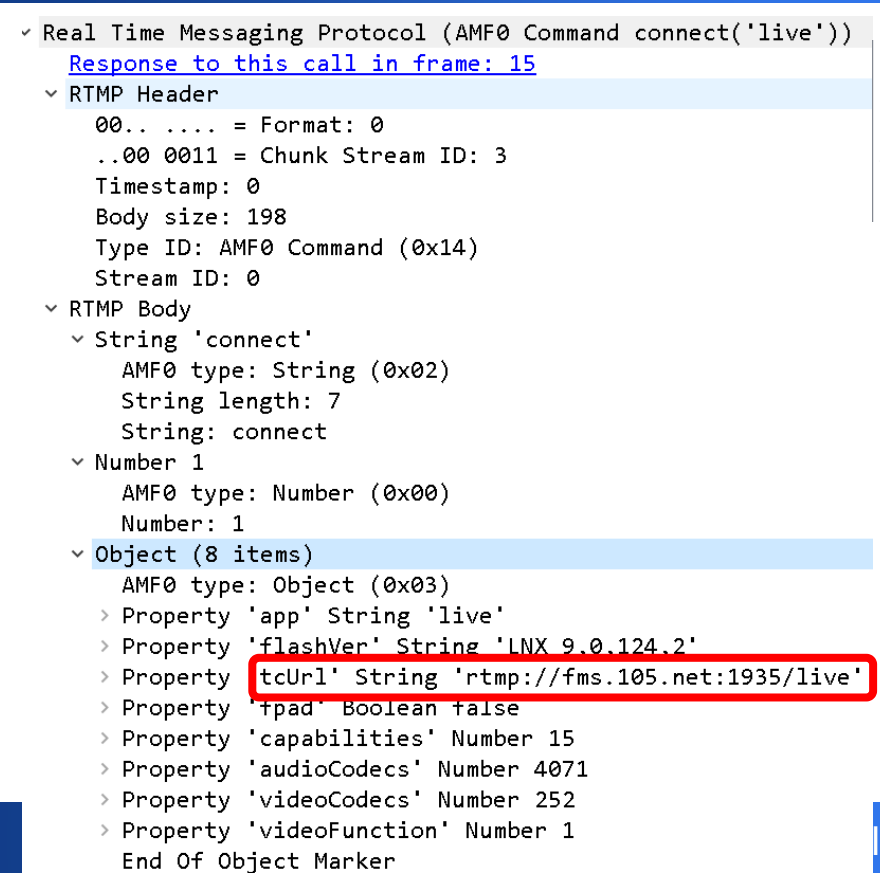
Handshake of RTMP

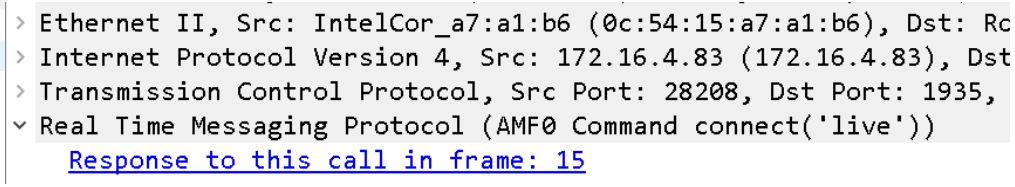
- RTMP exchanges 3 packets for handshake after the TCP connection is created.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.068426	172.16.4.83	37.247.49.157	RTMP	131	Handshake C0+C1
9	0.123162	37.247.49.157	172.16.4.83	RTMP	207	Handshake S0+S1+S2
12	0.123281	172.16.4.83	37.247.49.157	RTMP	130	Handshake C2

- Client sends C0+C1, server responds with S0+S1+S2, then Client sends C2 to exchange version and timestamp and confirmation.

Connection of RTMP





#14 packet
Client sends AMF (Action Message Format) message to connect server with tcURL rtmp://fms.105.net:1935/live, version, codecs etc.

```
Real Time Messaging Protocol (Window Acknowledgement Size 2500000)
  RTMP Header
  RTMP Body
    Window acknowledgement size: 2500000
Real Time Messaging Protocol (Set Peer Bandwidth 2500000,Dynamic)
  RTMP Header
  RTMP Body
    Window acknowledgement size: 2500000
    Limit type: Dynamic (2)
Real Time Messaging Protocol (User Control Message Stream Begin 0)
  RTMP Header
  RTMP Body
    Event type: Stream Begin (0)
Real Time Messaging Protocol (Set Chunk Size 4096)
  RTMP Header
  RTMP Body
    Chunk size: 4096
Real Time Messaging Protocol (AMF0 Command _result('NetConnection.Connect.Success'))
  Call for this response in frame: 14
  RTMP Header
  RTMP Body
    String '_result'
      AMF0 type: String (0x02)
      String length: 7
      String: _result
    Number 1
      AMF0 type: Number (0x00)
      Number: 1
    Object (3 items)
      AMF0 type: Object (0x03)
      Property 'fmsVer' String 'FMS/3,5,7,7009'
      Property 'capabilities' Number 31
      Property 'mode' Number 1
      End Of Object Marker
    Object (6 items)
      AMF0 type: Object (0x03)
      Property 'level' String 'status'
      Property 'code' String 'NetConnection.Connect.Success'
      Property 'description' String 'Connection succeeded.'
      Property 'data' ECMA array
      Property 'clientid' Number 637531730
      Property 'objectEncoding' Number 0
      End Of Object Marker
```

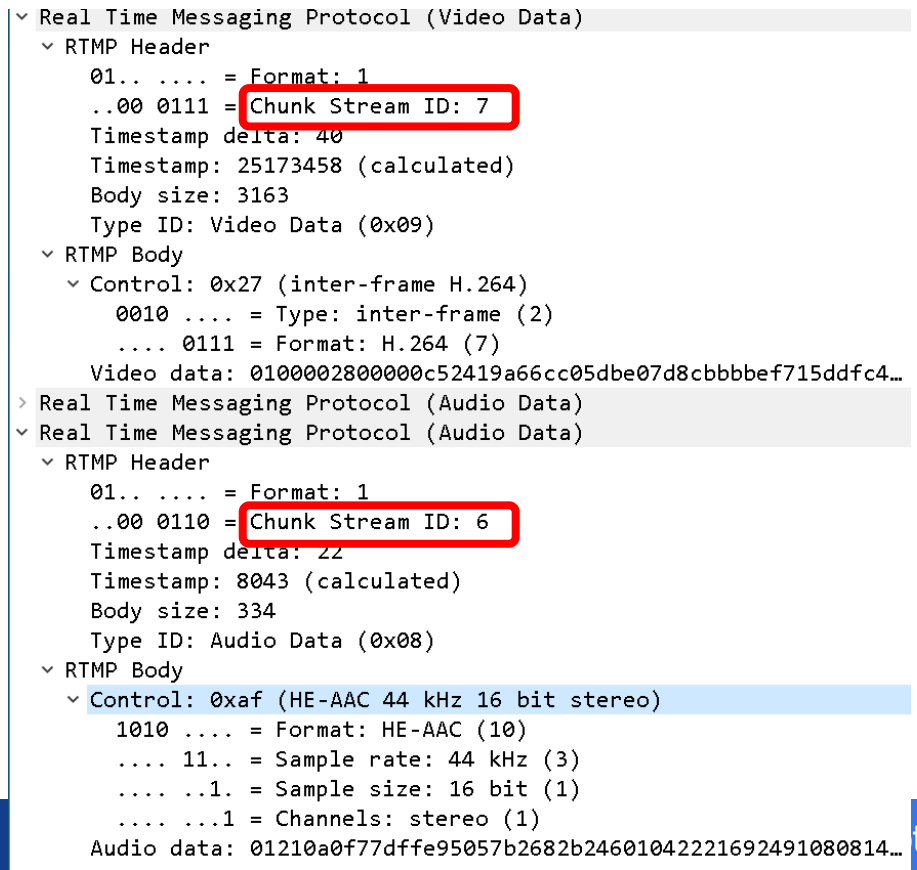
#15 Server responds with windows Ack Size, bandwidth, stream information, chunk size, and result as AMF format as “ connection succeeded”

Playing the stream

```
Window Acknowledgement Size 2500000|createStream()
_result()
getStreamLength()|play('rmc1')|Set Buffer Length 1,3000ms
_result()
Stream Begin 1|onStatus('NetStream.Play.Reset')|onStatus('NetStream.Play.Start')||Rtmp
onMetaData()|Video Data|Video Data
```

- #18 Client creates the stream
- #20 Server responds with result
- #23 Client sends play amf message to play ‘rmc1’
- #25 Server responds with result
- #27 Server starts playing the stream play('rmc1')

Streaming Data



Check #363

- Audio and Video data is divided into small size and sent by each Chunk Stream ID
- RTMP Body contains type and codec.
- Flash Player receives them and play the stream smoothly.

Save FLV using RTMDump

- RTMPDump is useful when you save video to flv
  https://rtmpdump.mplayerhq.hu/
  This time we use rtmpdump-2.3-windows.zip
- We got the stream information from packet
  From #14 connect rtmp://fms.105.net/live/
  From #23 play AMF0 Command play('rmc1')
- RTMPdump command is below
  rtmpdump –r rtmp://fms.105.net/live/rmc1
  –o ../out.flv

Playing FLV file

```
C:¥Users¥megumi¥Desktop¥rtmpdump-2.3>rtmpdump -r rtmp://fms.105.net/live/rmc1 -o ../out.flv
RTMPDump v2.3
(c) 2010 Andrej Stepanchuk, Howard Chu, The Flvstreamer Team; license: GPL
Connecting ...
INFO: Connected...
Starting download at: 0.000 kB
INFO: Metadata:
INFO:    author
INFO:    copyright
INFO:    description
INFO:    keywords
INFO:    rating
INFO:    title
INFO:    presetname              Custom
INFO:    creationdate            Sun Nov 03 08:05:30 2019
INFO:    videodevice             DML1 OutputVideo
INFO:    framerate               25.00
INFO:    width                   768.00
INFO:    height                  432.00
INFO:    videocodecid            avc1
INFO:    videodatarate           700.00
INFO:    avclevel                31.00
INFO:    avcprofile              66.00
INFO:    videokeyframe_frequency 5.00
INFO:    audiodevice             MediaLooks MultiGraph Audio
INFO:    audiosamplerate         48000.00
INFO:    audiochannels           2.00
INFO:    audioinputvolume        75.00
INFO:    audiocodecid            mp4a
INFO:    audiodatarate           128.00
1610.139 kB / 16.06 sec
```

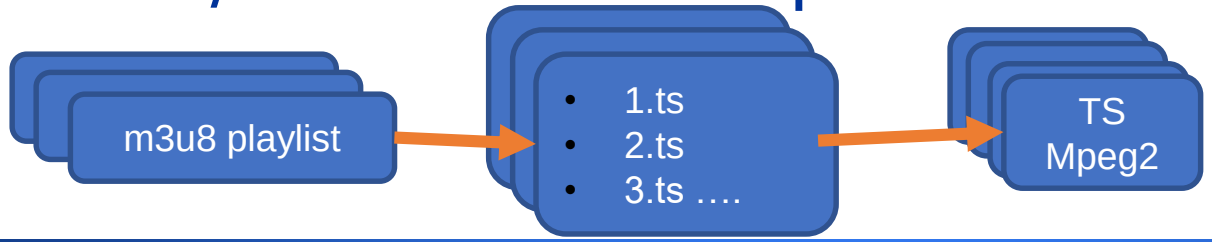
Collecting location and name of contents from Wireshark trace, you can save live or on-demand streamings to your PC

HLS (HTTP Live Streaming)

- The Flash age has been finished, off course there are still many FLV, but now we live in HTML5 generation.
- Apple may not like Adobe, so iPhone/iPad uses HLS. https://developer.apple.com/streaming/
- HLS, a.k.a. HTTP Live Streaming is an HTTP-based streaming protocol by Apple. Not only Safari and QuickTime, but Firefox, Microsoft Edge support HLS.
- HLS divides a stream into small HTTP file downloads, m3u8 extension is used for sending stream and bitrate list. And HLS also supports HTTPS too.

M3U8 playlist

- HLS is so simple mechanism to stream
- Distributor creates multiple TS files using stream segmenter (if live) or file segmenter (if on-demand)
- M3U8 playlist contains links to multiple TS file
- Server provides (multiple) m3u8 playlists
- Clients choose a M3U8 and play chunked TS file via HTTP/HTTPS with adequate cache.



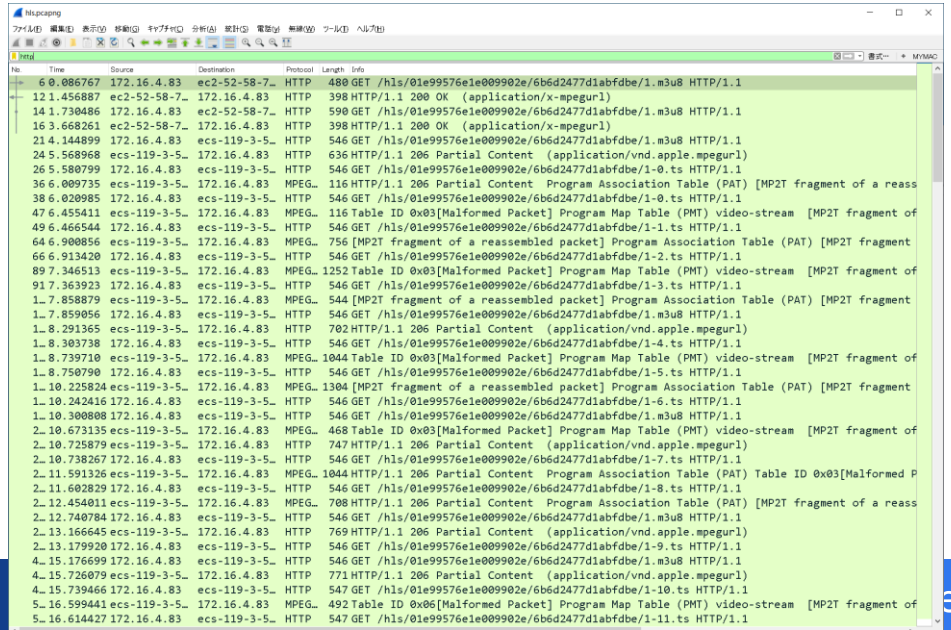
Test Network Camera Live Streaming

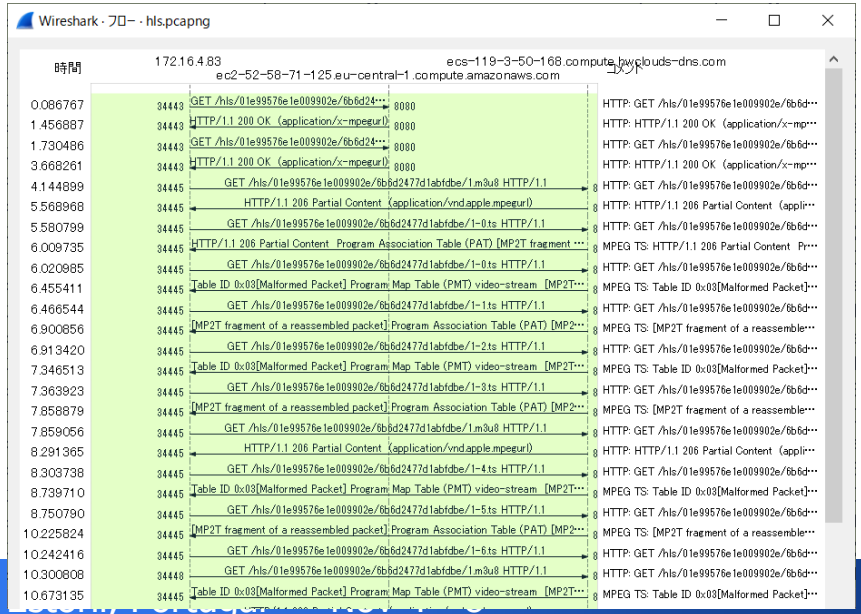
- Some Network Camera supports HLS, we try Unifore's
- dumpcap -i 3 -f "host 52.58.71.125 or port 80 or port 8080 or port 8088" –w hls.pcapng
- Open live stream with Microsoft Edge, Safari or VLC (Windows version of Chrome and Firefox may not)

http://pub-dss-hls.secu100.net:8080/hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8

hls.pcapng

- Open hls.pcapng and set display filter as http
- Statistics > Flow Graph (limit to display filter)





HLS Flows(#6/#12)

```
GET /hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8 HTTP/1.1
...
Host: pub-dss-hls.secu100.net:8080
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: openresty/1.9.3.1
Date: Mon, 04 Nov 2019 11:26:22 GMT
Content-Type: application/x-mpegURL
Connection: keep-alive
KeyCheck: 200000
content-length: 129
Access-Control-Allow-Origin: *

#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=512000
http://119.3.50.168:8088/hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8
```

Client requests m3u8 Playlist

Server responds bandwidth and alternative m3u8 Playlist (there are multiple m3u8 playlist for different bitrate settings)

HLS Flows (#14 / #16)

```
GET /hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8 HTTP/1.1
Referer: http://pub-dss-hls.secu100.net:8080/hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8
...
Host: pub-dss-hls.secu100.net:8080
Connection: Keep-Alive
Cache-Control: no-cache
HTTP/1.1 200 OK
Server: openresty/1.9.3.1
Date: Mon, 04 Nov 2019 11:26:24 GMT
Content-Type: application/x-mpegURL
Connection: keep-alive
KeyCheck: 200000
content-length: 129
Access-Control-Allow-Origin: *

#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=512000
http://119.3.50.168:8088/hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8
```

Client requests alternative m3u8 Playlist

Server responds bandwidth and m3u8 playlist including links to TS files

HLS Flows (#22 / #24)

```
GET /hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8 HTTP/1.1
Referer: http://pub-dss-hls.secu100.net:[illegible]80/hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8
...
Range: bytes=0-
Accept-Encoding: gzip, deflate
Host: 119.3.50.168:8088
Connection: Keep-Alive
Cache-Control: no-cache
```

Request m3u8 Playlist including link to TS files

```
HTTP/1.1 206 Partial Content
Server: openresty/1.9.3.1
Date: Mon, 04 Nov 2019 11:26:26 GMT
Content-Type: application/vnd.apple.mpegurl
Content-Length: 183
Last-Modified: Mon, 04 Nov 2019 11:26:26 GMT
Connection: keep-alive
ETag: "5dc00ae2-b7"
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0
Access-Control-Allow-Origin: *
Content-Range: bytes 0-182/183
```

```
#EXTM3U
#EXT-X-ALLOW-CACHE:NO
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:2
#EXTINF:1.990,
1-0.ts
#EXTINF:1.990,
1-1.ts
#EXTINF:1.990,
1-2.ts
#EXTINF:1.990,
1-3.ts
```

Streaming setting information and actual link to TS files (1-0.ts…)

HLS Flows (#26 / #36)

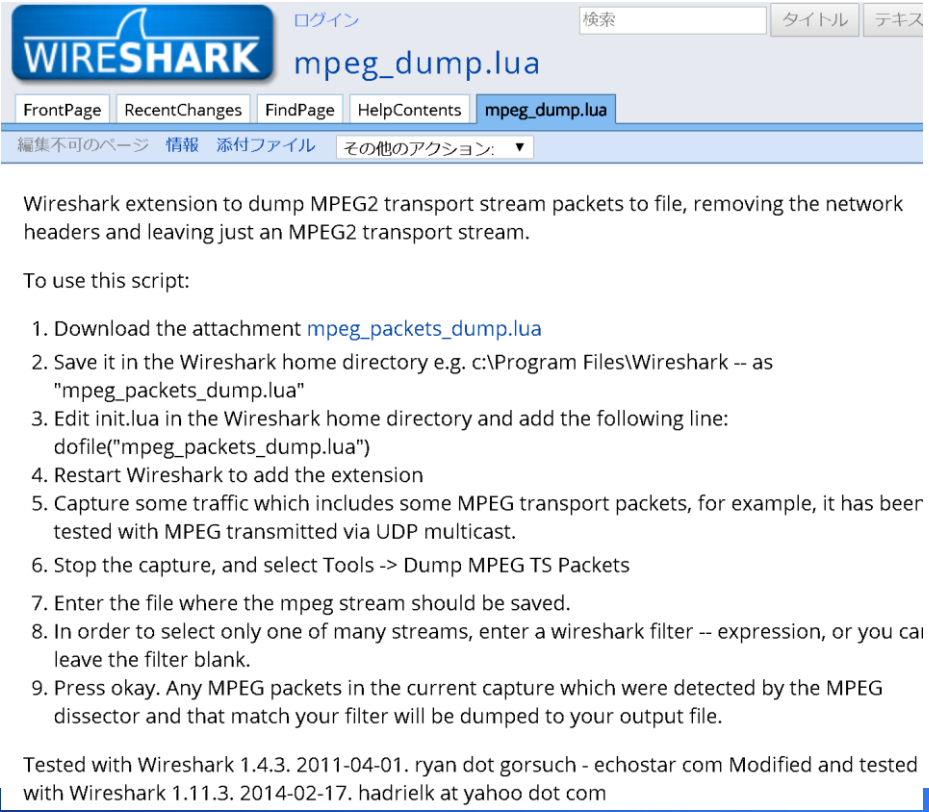
```
GET /hls/01e99576e1e009902e/6b6d2477d1abfdbe/1-0.ts HTTP/1.1
...
Host: 119.3.50.168:8088
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 206 Partial Content
Server: openresty/1.9.3.1
Date: Mon, 04 Nov 2019 11:26:27 GMT
Content-Type: video/mp2t
Content-Length: 11468
Last-Modified: Mon, 04 Nov 2019 11:26:23 GMT
Connection: keep-alive
ETag: "5dc00adf-2ccc"
Content-Range: bytes 0-11467/11468
```

Request to chunked TS file (1-0.ts)

Response to actual chunked TS file (1-0.ts)
HTTP body contains MPEG2

mpeg_dump_lua Scripts

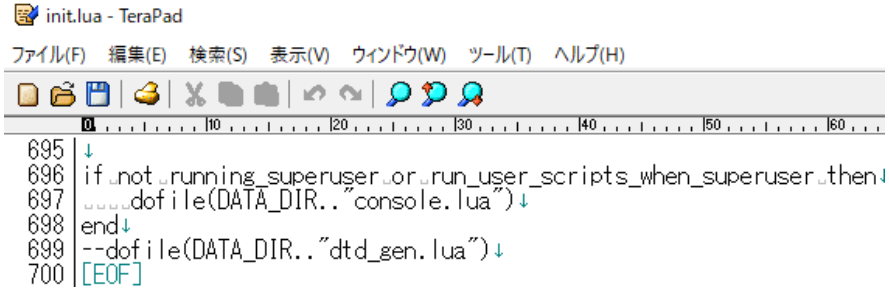


- There are many useful scripts, plugins and dissectors to extend Wireshark functions at wiki.wireshark.org/Tools
- mpeg_dump.lua by Ryan is a useful lua script to remove protocol headers and output mpeg2 stream

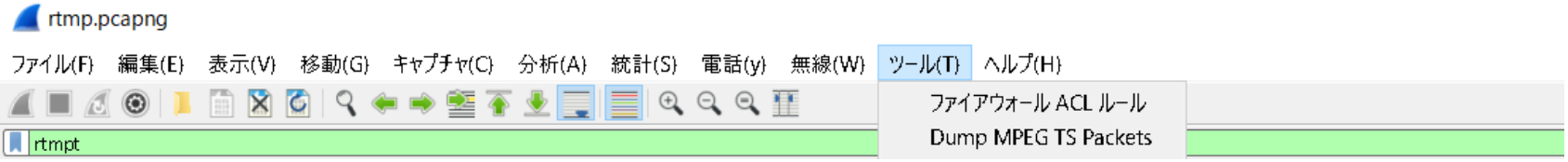
Set up mpeg_packets_dump.lua

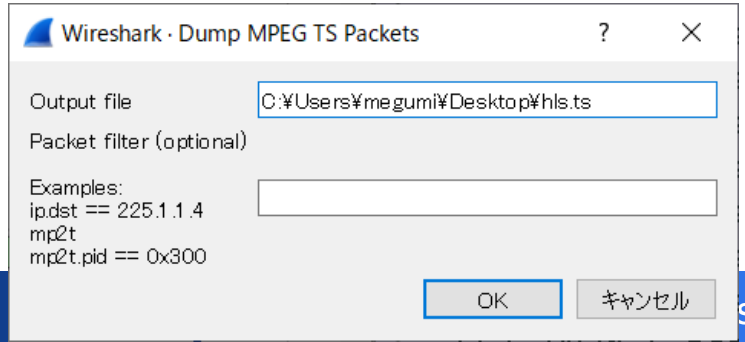
- Download mpeg_packets_dump.lua from wiki
- Save it in the Wireshark home directory C:¥Program Files¥Wireshark as "mpeg_packets_dump.lua"
- Edit init.lua in the Wireshark home directory and add the following line: dofile("mpeg_packets_dump.lua")

Export MPEG2(TS) from HLS



- Put scripts into Wireshark global folder
- Edit init.lua
- Select Tool> Dump

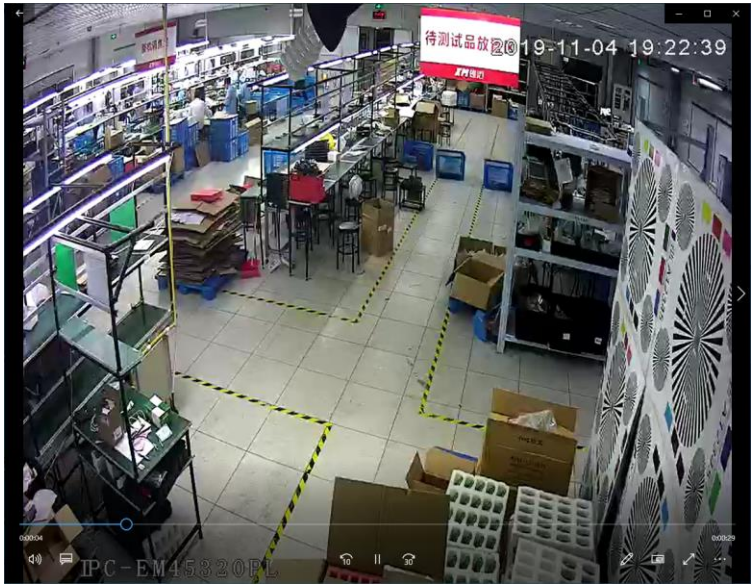




- Set Output file path and packet filter if needed then click OK to export.

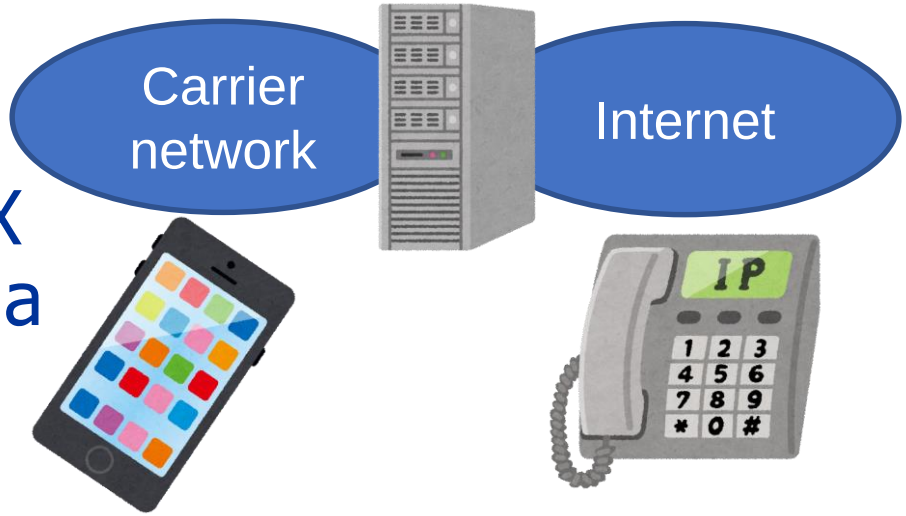
Open TS file and check MPEG2

- Open TS(MPEG2) file using Windows Media Player
- You can check streaming video qualities from the trace file



VoIP (IP Phone)

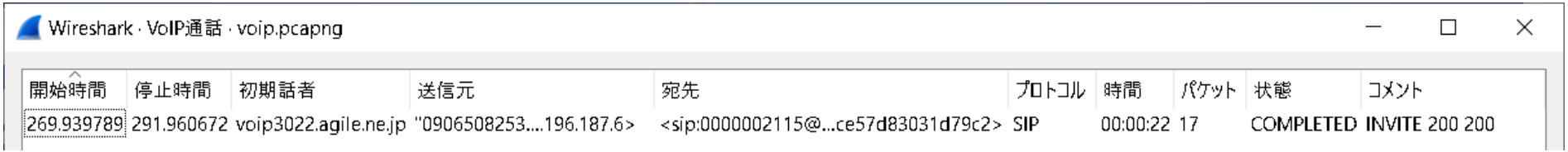
- Voice over IP is used widely in todays internet
- SIP server is a kind of PBX in VoIP network and media gateway is used for connecting former phone system with IP network



- RTP(Real-time Transport Protocol) and RTCP are used for transmitting Audio and Video data.

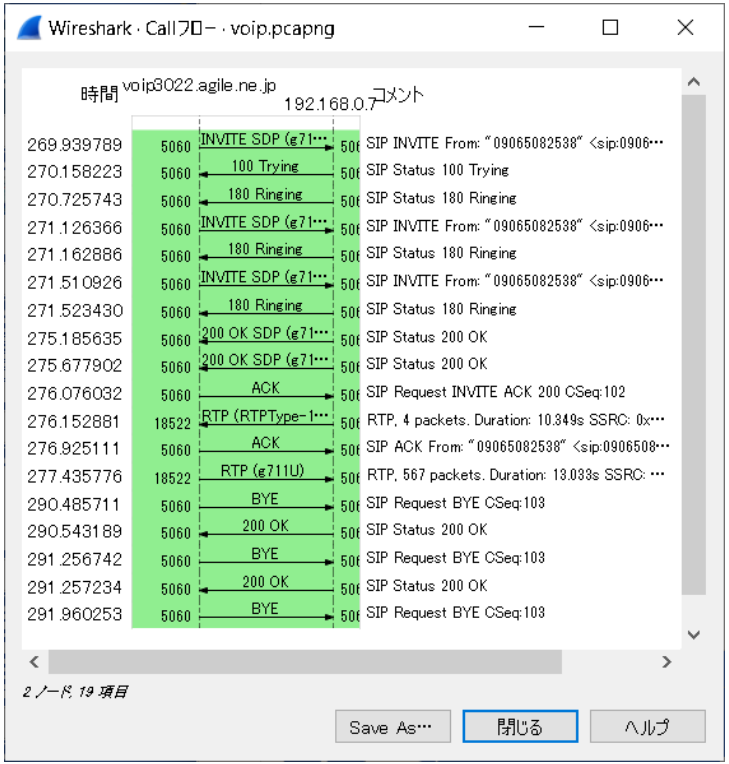
voip.pcapng

- We captured traffic at LAN (IP Phone) side, Sender 09065082538@61.196.187.6 Receiver 0000002115@192.168.0.7:506
- Open trace file voip.pcapng in Wireshark, Telephony > VoIP calls to show call



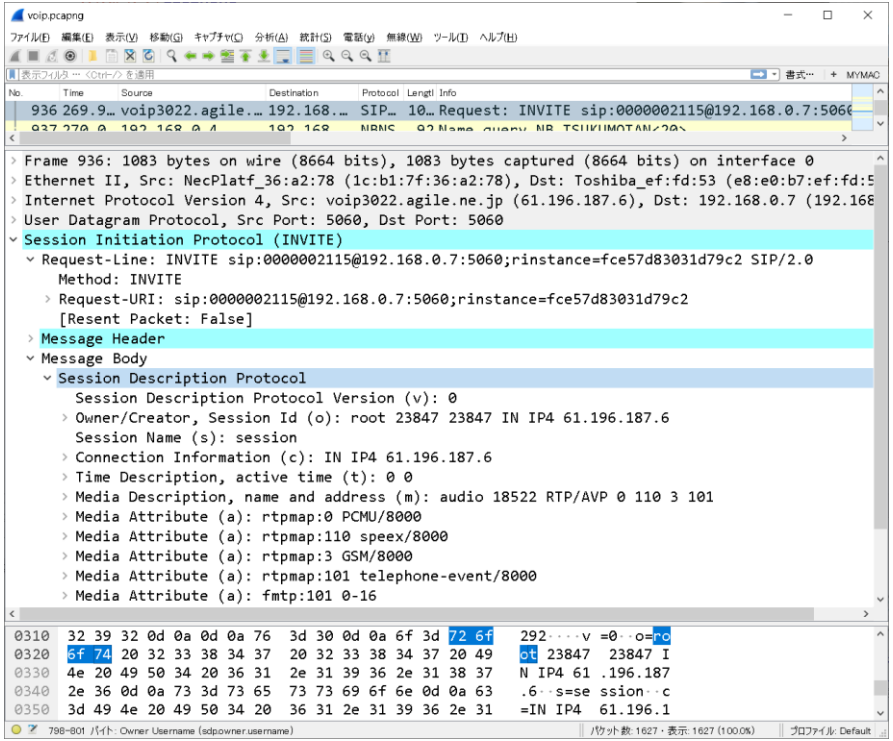
- Choose call and click Flow sequence

Call Flow



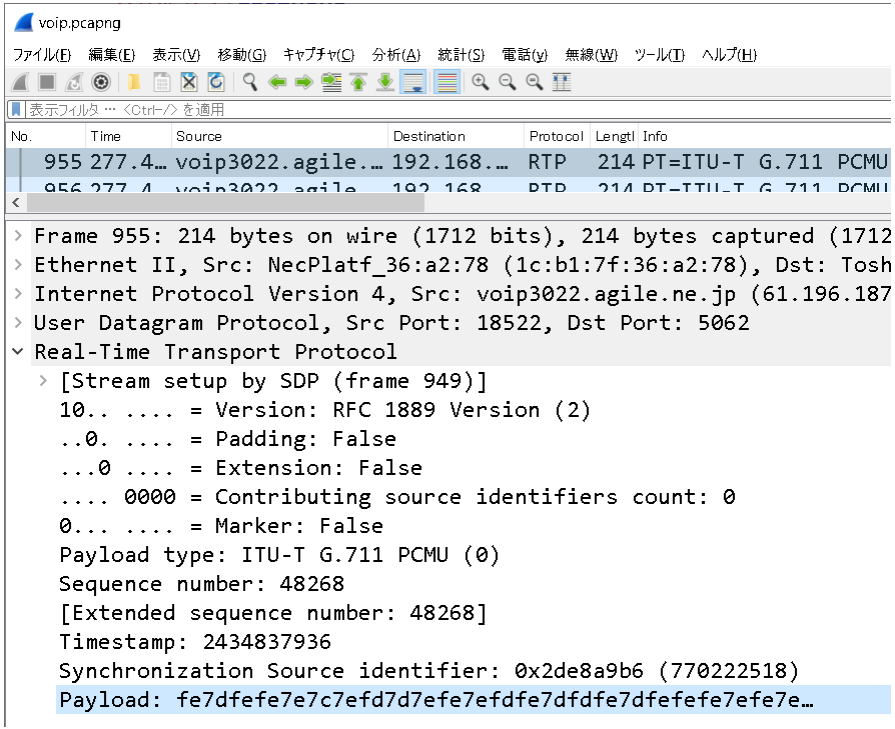
- SIP call starts with INVITE method, then receives response code 100 Trying, 180 Ringing, and 200 OK response to connect.
- Voice data is sent by RTP
- SIP call ends with BYE method, and receives 200 OK response.
- Click INVITE and RTP packets

SIP (Session Initiation Protocol)



- Look at #936 INVITE message sent by UDP
- SIP message is similar to HTTP, consists of header and body
- Each request and response have Request-Line and Response code, and many headers.

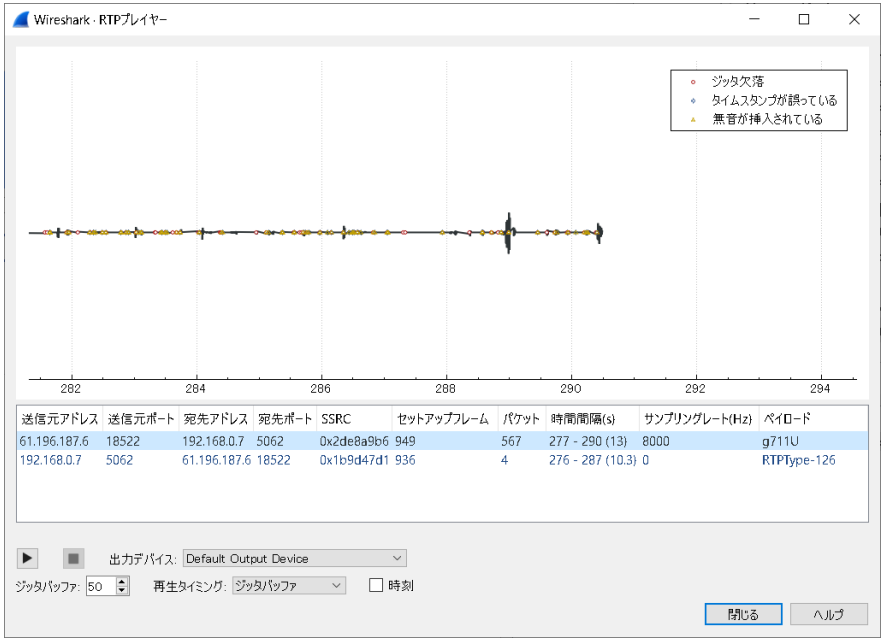
RTP (Real-Time Transport Protocol)



- Look at #955 RTP packet sent by UDP
- RTP packets contains Payload type, Sequence number, Timestamp and SSRC (Synchronization Source identifier) and Payload of actual media.

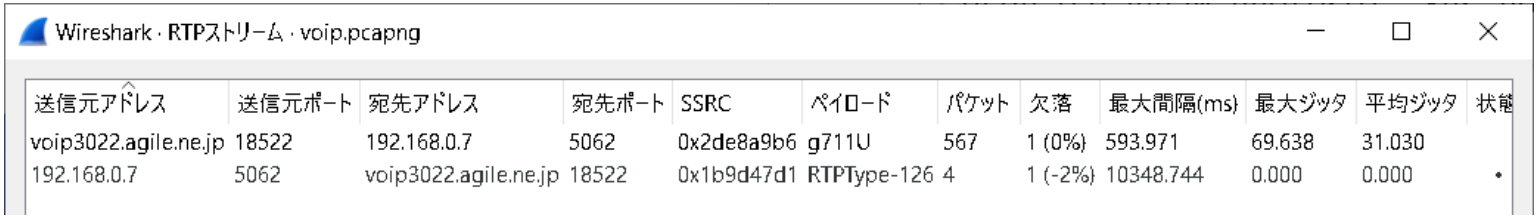
RTP Player

- Back to VoIP Calls, click “Play Stream” to open RTP Player
- We can check RTP payloads graph with the symbol of lost jitter, wrong timestamp and insert silence
- We can also listen the quality of audio (Note: this time codec is G.711u)

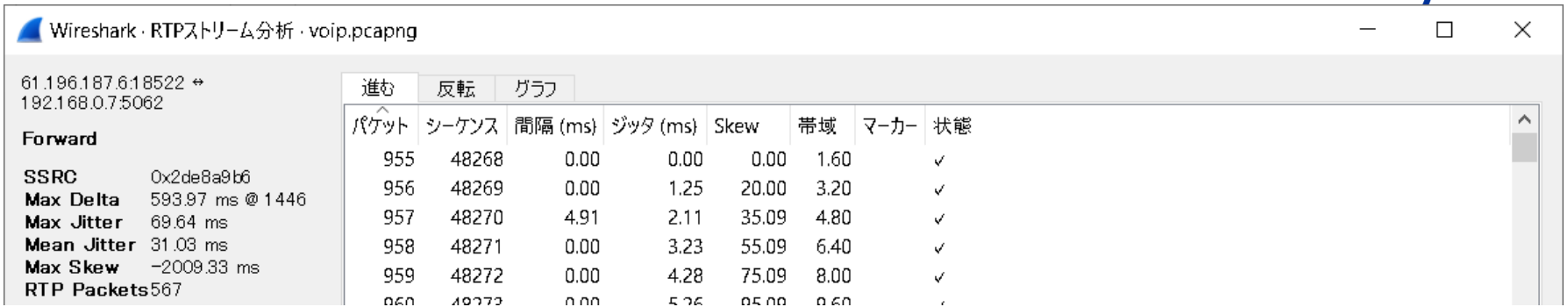


Telephony > RTP Streams

- Telephony > RTP Streams summarize RTP streams

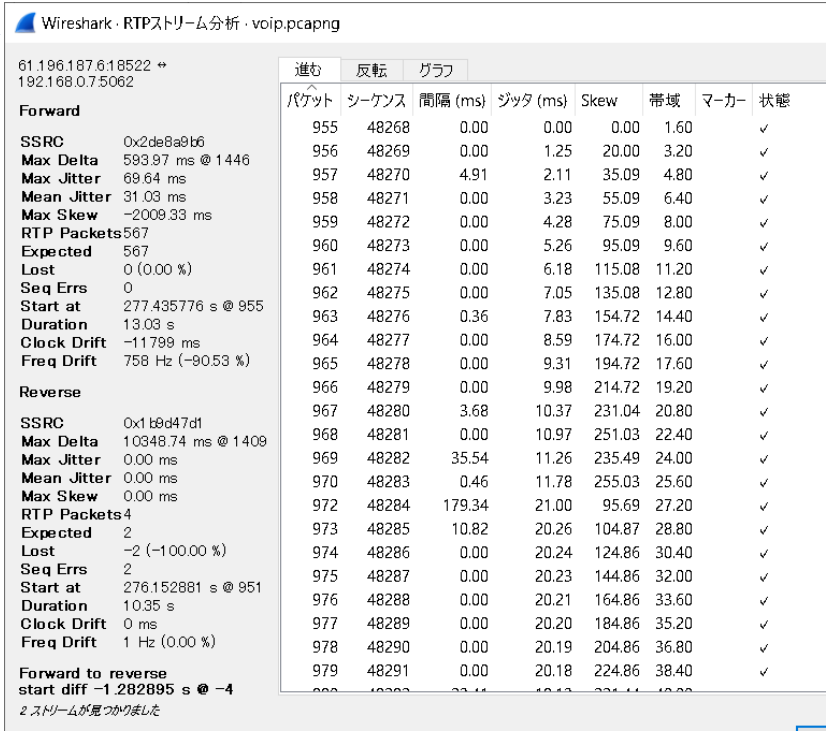


- Shift click to select both streams and click Analyze



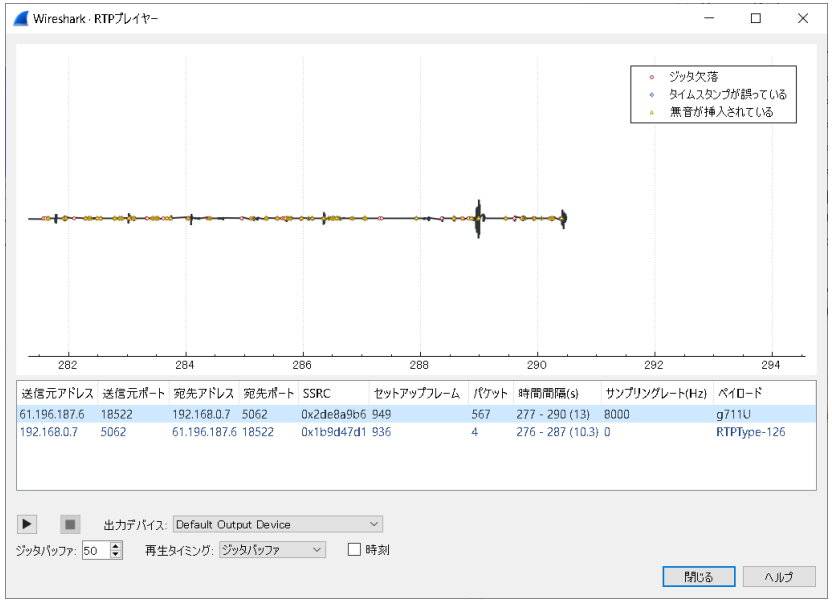
RTP Stream Analysis

- You can check both streams, SSRC, Delta, Jitter, Skew, Seq Errors and Drift info.
- Its time to save audio data, click Save as Synchronized or Asynchronized stream as au format.



Playing Saved RTP Audio.au

- VLC is a free and open source cross-platform multimedia player (https://www.videolan.org/vlc)
- Open Saved RTP Audio.au using VLC Player
- We can hear some Japanese words…



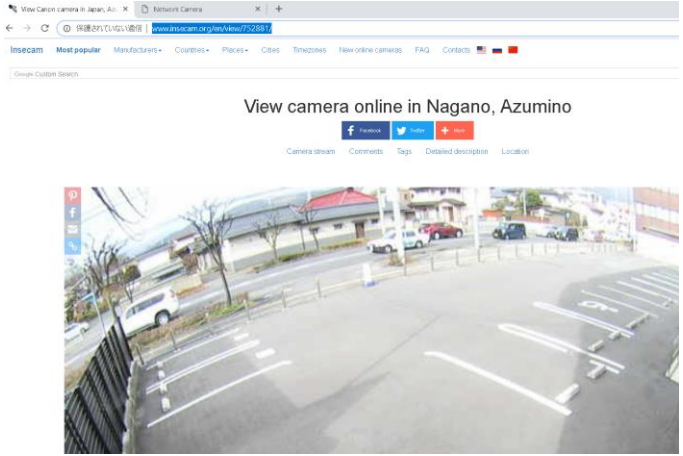
Surveillance camera

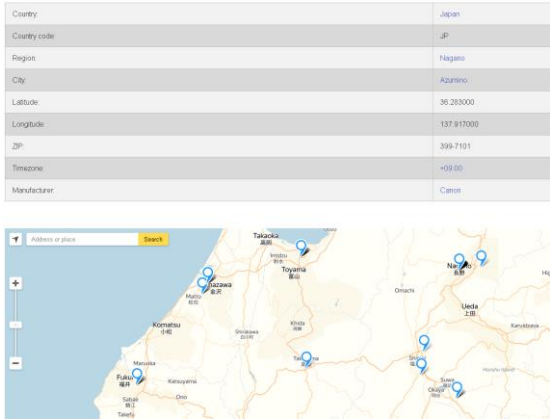
- There are tons of surveillance camera in the world, we can access them via web
- Insecam.com(http://www.insecam.com) is a famous website introducing public opened surveillance IP camera

- You can search cameras by Manufactures, Country, Places, Cities and so on.

Canon camera in Nagano, Japan

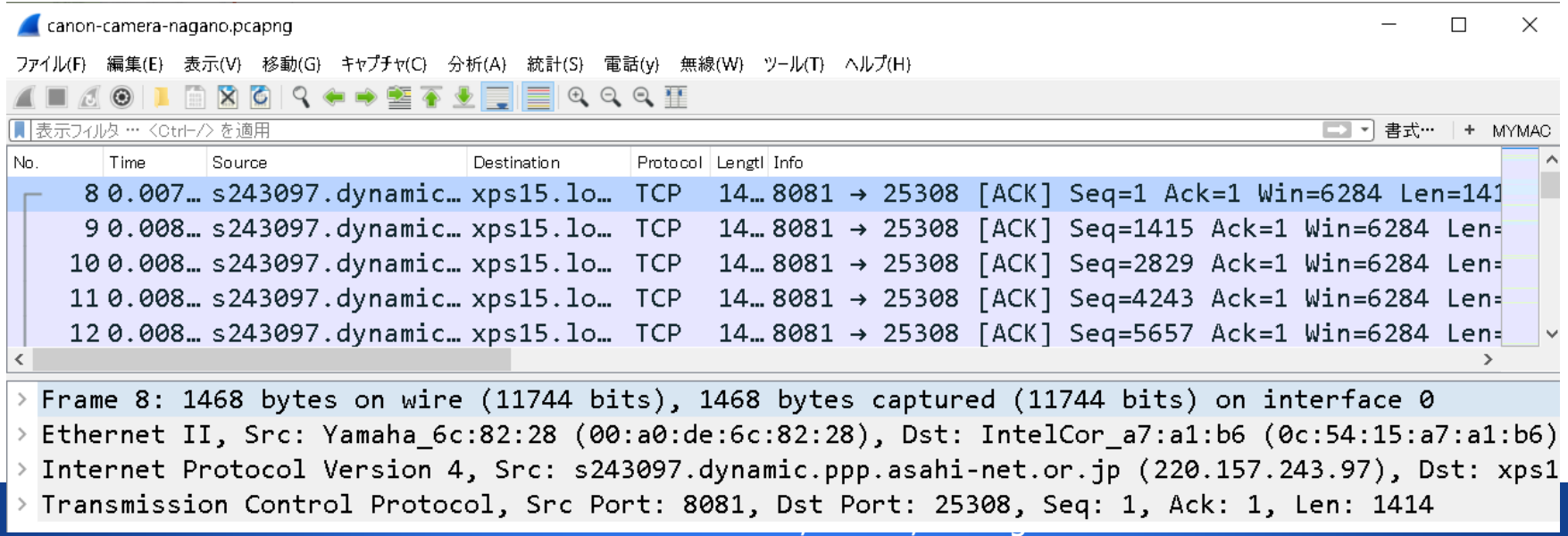
- We pick up a Canon surveillance camera at a parking lot in Nagano, Japan.
- Access the camera and capture the packet, and save as “canon-camera-nagano.pcapng”





Find the camera port number

- Open canon-camera-nagano.pcapng and select Statistics > Conversation > TCP and UDP tab
- Look for the port number that the camera use.



Decode TCP 8081 as HTTP

Wireshark · Conversations · canon-camera-nagano.pcapng

Ethernet · 12 | IPv4 · 37 | IPv6 · 3 | TCP · 36 | UDP · 43

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.100.30	25308	220.157.243.97	8081	5,084	5614 k	1,450	311 k	3,634	5303 k	0.007638	31.0797	80 k	1365 k
192.168.100.30	24889	220.157.243.97	8081	10,348	11 M	2,604	144 k	7,744	11 M	0.039839	31.0617	37 k	2871 k
192.168.100.30	25122	23.211.96.27	80	2	121	1	55	1	66	1.941729	0.0083	53 k	63 k

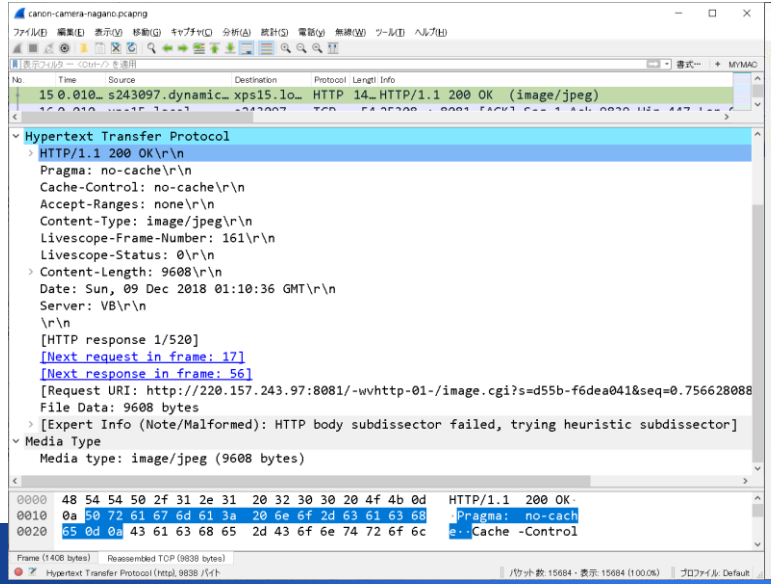
- We find TCP 8081 is the camera port, then assume the protocol. This time the camera uses HTTPso try to decode TCP 8081 as HTTP.
- Select packet and right click to choose “Decode As…” and set 8081 as HTTP

Wireshark · Decode As...

フィールド	値	種別	デフォルト	現在
TCP port	8081	整数型, 基数10	(none)	HTTP

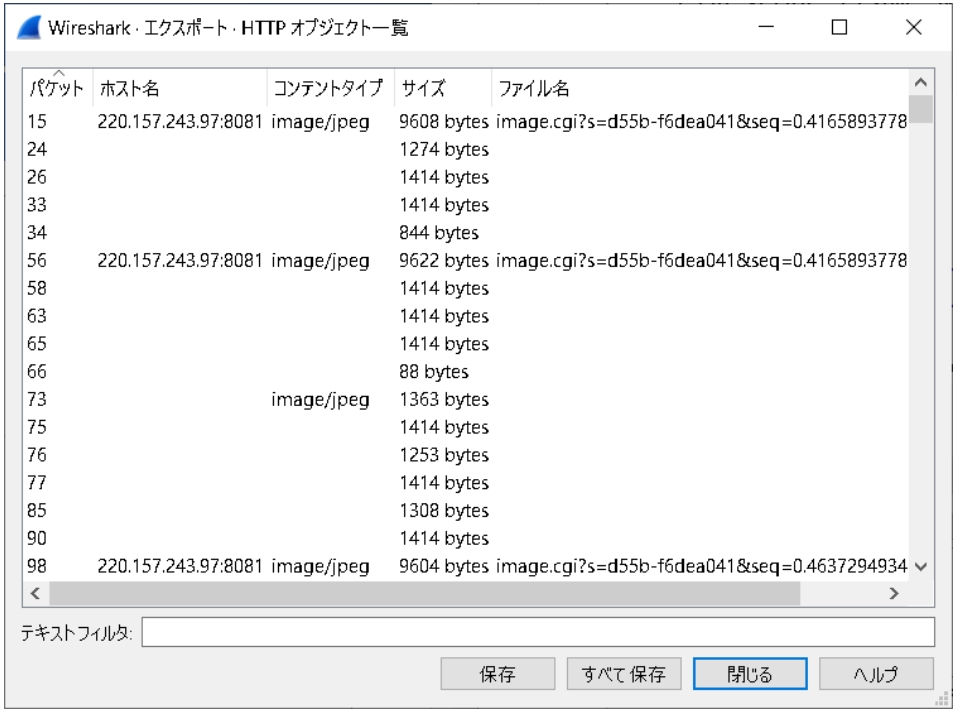
Check HTTP response and media

- Wireshark decode TCP8081 segments as HTTP, check #15 frame and open HTTP dissector and select Media Type and right click to “Show Packet



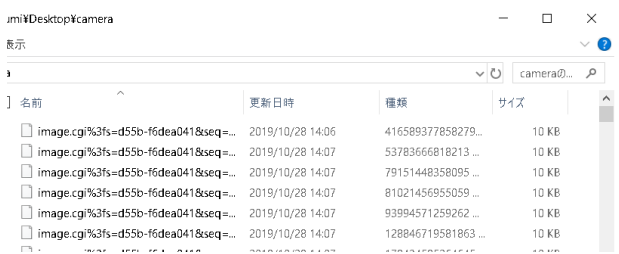
Export Objects as HTTP

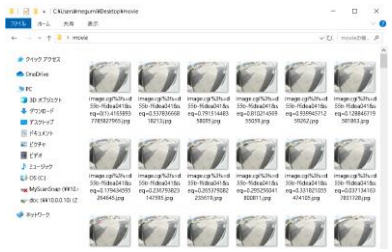
- File>Export Objects > HTTP to export files to camera folder on Desktop
- There are file lists grouped by host name, content type, size, file name
- Create a folder and “Save All” to save them



Change file extension to jpeg

- There are many image/jpeg files in the folder, but all file name is image.cgi%3fs=***
- Create new folder movie and copy these files.
- Change file extension to jpeg but there are dot in the filename, so use for command like this for %s in (*.*) do rename "%s" "%s.jpg"





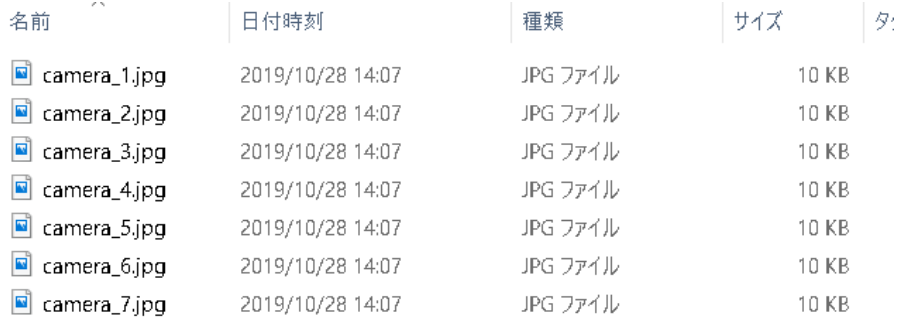
Create multiple jpeg files to movie

- There are many tools to create movie from multiple jpeg files (ex. motion jpeg)
- Adobe Premier Elements read multiple numbered JPEG files to create a video clip.
- Some open source tools like Virtual Dub, CamStudio support to create a movie
- In this case, we use ffmpeg to create movie

Create sequential files of jpeg

- Open explorer and click one of jpeg files and rename filename to create sequential files. (514 files)

名前	日付時刻	種類	サイズ
camera (1).jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera (2).jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera (3).jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera (4).jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera (5).jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera (6).jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera (7).jpg	2019/10/28 14:07	JPG ファイル	10 KB

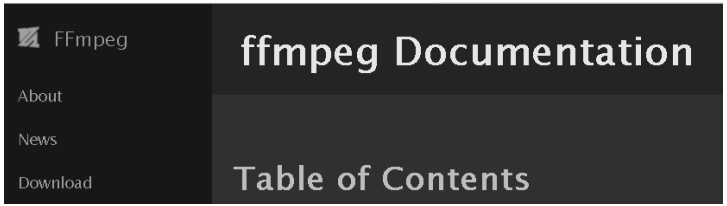


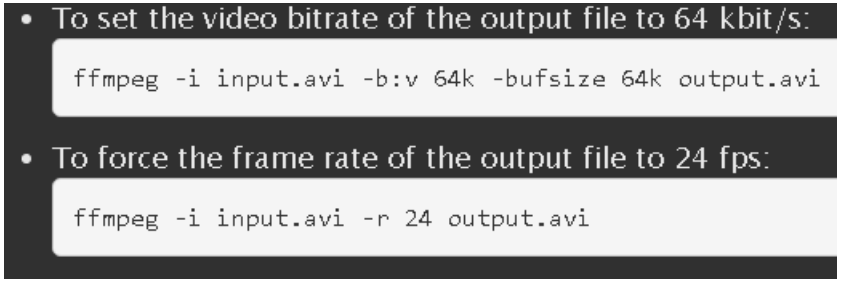
名前	日付時刻	種類	サイズ
camera_1.jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera_2.jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera_3.jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera_4.jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera_5.jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera_6.jpg	2019/10/28 14:07	JPG ファイル	10 KB
camera_7.jpg	2019/10/28 14:07	JPG ファイル	10 KB

- for /l %i in (1, 1, 514) do ren “camera (%i).jpg" “camera_%i.jpg"

ffmpeg (ffmpeg.org)

- ffmpeg is GPL tools for recording, converting and playing video and audio streams. This time we use ffmpeg-20181208-6b1c4ce-win64-static
- There are tons of options (https://ffmpeg.org/ffmpeg.html)





Create movie from sequential files

- ffmpeg -f image2 -r 30 -i camera_%d.jpg -r 30 -an -vcodec libx264 -pix_fmt yuv420p camera.mp4

-f image2 :format image2(jpeg) -r 30 30fps
-i source files –an no audio –vcodec libx264(H.264)
-pix_fmt yuv420p

```
コマンド プロンプト
C:¥Users¥megumi¥Desktop¥movie>..¥ffmpeg¥ffmpeg -f image2 -r 30 -i camera_%d.jpg -an -vcodec libx264 -pix_fmt yuv420p camera.mp4
ffmpeg version N-92639-g6b1c4ce8cf Copyright (c) 2000-2018 the FFmpeg developers
  built with gcc 8.2.1 (GCC) 20181201
  configuration: --enable-gpl --enable-version3 --enable-sdl2 --enable-fontconfig --enable-gnutls --enable-iconv --enable-libass --enable-libbluray --enable-libfreetype --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libopenjpeg --enable-libopus --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libtheora --enable-libtwolame --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2 --enable-libzimg --enable-lzma --enable-zlib --enable-gmp --enable-libvidstab --enable-libvorbis --enable-libvo-amrwbenc --enable-libmysofa --enable-libspeex --enable-libxvid --enable-libaom --enable-libmfx --enable-amf --enable-ffnvcodec --enable-cuvid --enable-d3d11va --enable-nvenc --enable-nvdec --enable-dxva2 --enable-avisynth --enable-libopenmpt
```

Play video with VLC Player

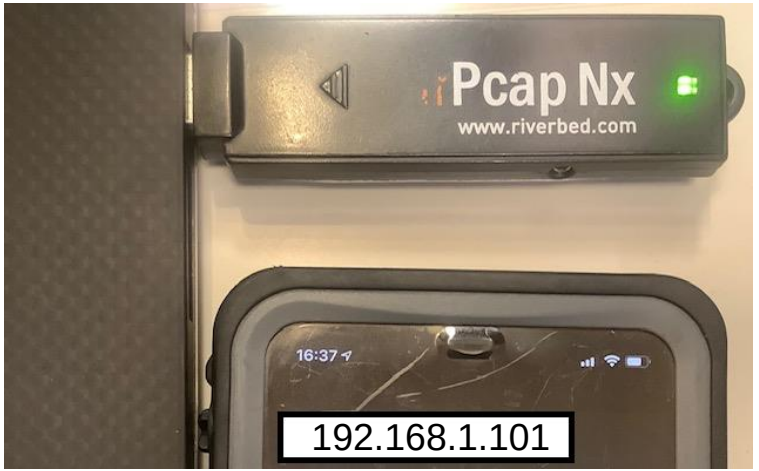
- Open camera.mp4 using VLC Player
- Finally we can get the movie from trace file !!

#9 Unknown Drive Recorder

- Many Drive recorders are made in China or Taiwan.
- They use WiFi to connect smartphone App to check the camera recordings.
- Let’s dissect with Wireshark
- Recorder IP :192.168.1.1

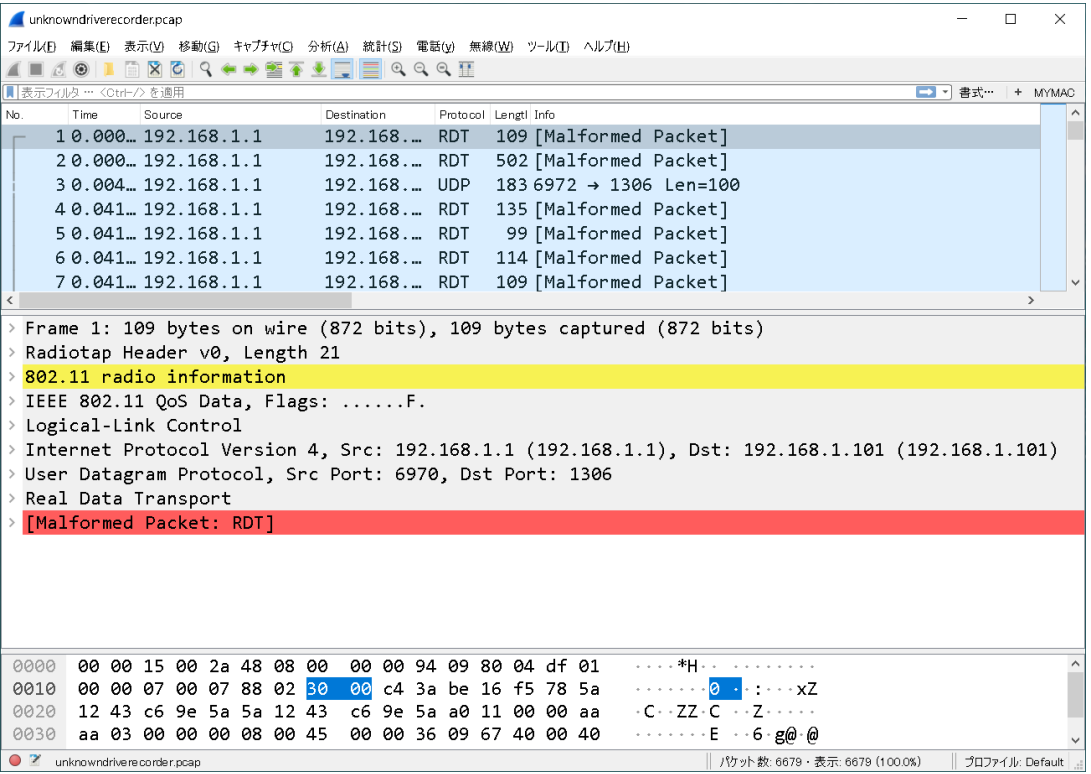
Trace file information

- Unknown drive recorder 192.168.1.1
- Smartphone of viewer app 192.168.1.101
- Open Smartphone App to check the video, then I collect packets using AirPcap and Wireshark



192.168.1.1

unknowndriverecorder.pcap



- The trace file contains traffic between unknown drive recorder and smartphone app in Wi-Fi Channel 9, MCS 7, IEEE802.11n (20MHz)

What including tons of small packets

- Open trace file unknowndriverecorder.pcapng
- We can see many small packets
- Sometimes we need to decrypt WPA2 or TLS.
- Open small chunk with binary editor, it is lucky if the chunk is one of graphic file format, we can use the same way of surveillance camera, but No…
- Port number is usually changed from original.
- In this case, we think about RTP streams Statistics > Conversation to check UDP stream

Conversation > UDP streams

- A multimedia traffic is sent by a few streams.
- Audio and video streams are divided by two in usual.
- Wireshark finds streams below,
  192.168.1.1:6970 (RDT) →192.168.1.101:1306
  192.168.1.1:6971(RTCP)→192.168.1.101:1307
  192.168.1.1:6972 (DATA) →192.168.1.101:1306
  192.168.1.1:6973(RTCP)→192.168.1.101:1307
- Note: port numbers are usually changed
  its time to determine what protocol they truly use

Think about the role of Port

Wireshark · Conversations · unknowndriverecorder.pcap

Ethernet | IPv4 · 1 | IPv6 | TCP | UDP · 5

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.1.1	6970	192.168.1.101	1306	4,339	2140 k	4,339	2140 k	0	0	0.000000	69.6021	246 k	
192.168.1.1	6972	192.168.1.101	1306	2,279	420 k	2,279	420 k	0	0	0.004733	69.5957	48 k	
192.168.1.1	6971	192.168.1.101	1307	32	4129	32	4129	0	0	0.099017	65.6265	503	
192.168.1.1	6973	192.168.1.101	1307	8	1048	8	1048	0	0	4.080670	64.8931	129	
192.168.1.101	53207	192.168.1.1	53	1	136	1	136	0	0	16.059984	0.0000	—	

- Which stream is sending audio / video ? Statistics > Conversation > UDP to look for UDP socket.
- Port 6970 is used for video and 6972 is for audio because the bytes of UDP6970 are bigger.
- Port 6971,6973 may be used for control streams

Dissector of the port is correct ?

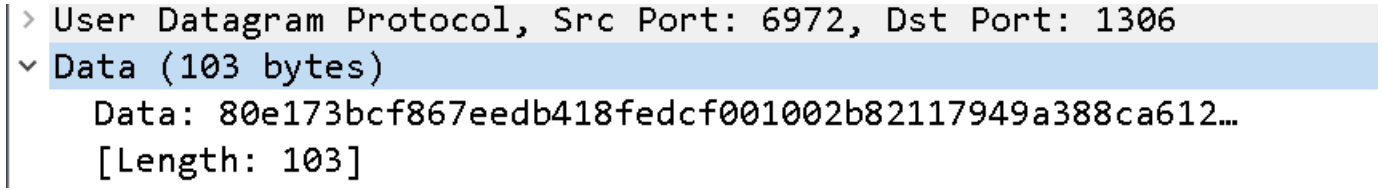
- UDP 6970 is well known port of RDT (Real Data Transport), but packets are broken. Ex. Packet #28

```
> User Datagram Protocol, Src Port: 6970, Dst Port: 1306
v Real Data Transport
  v RDT packet (Data)
    > Length-included=1, need-reliable=0, stream-id=0, is-reliable=0
      Sequence number: 24667
      Packet length: 14199
v [Malformed Packet: RDT]
  v [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
      [Malformed Packet (Exception occurred)]
      [Severity level: Error]
      [Group: Malformed]
```

- Well known port is not always correct.

Assume protocol for decoding

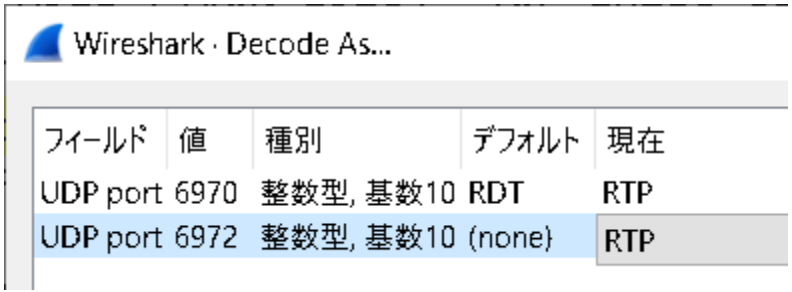
- UDP stream from port 6972 is not decoded as Audio protocol, Wireshark says just Data.



- Its time to assume protocol for decoding.
  The cheap drive recorder tends to use general protocols, like RTP (Real-time Transport Protocol)

Analyze > Decode As...

- Use “Analyze > Decode As...” to change UDP 6970 and 6972 as RTP (Real-time transport protocol)
- Port 6971 and 6973 is used for RTCP (Real-time Transport Control Protocol) as default

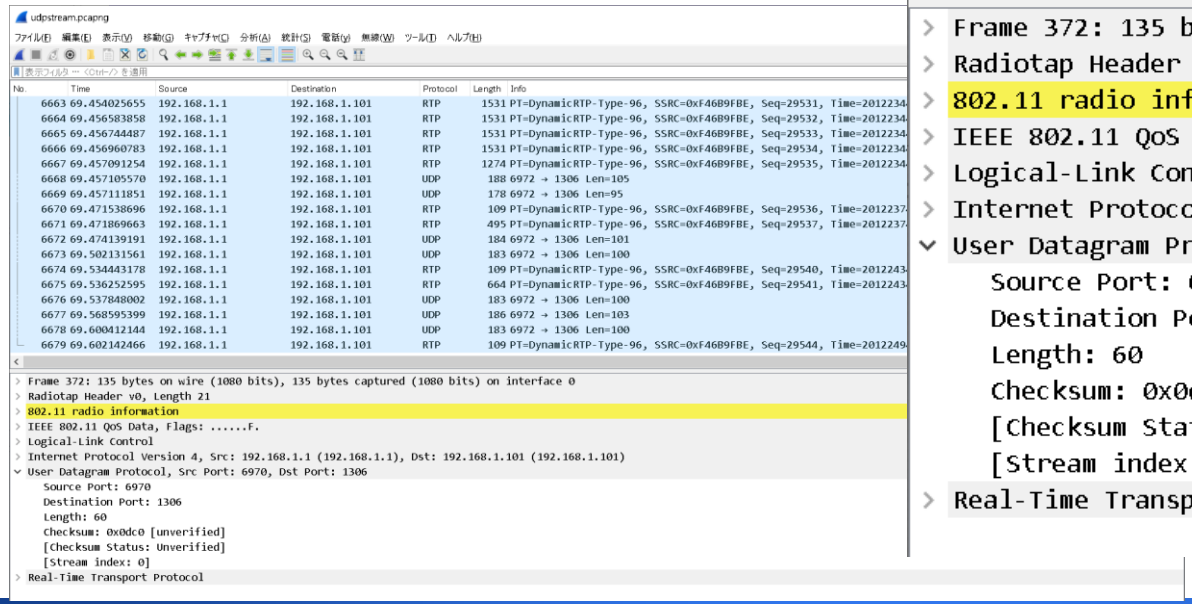


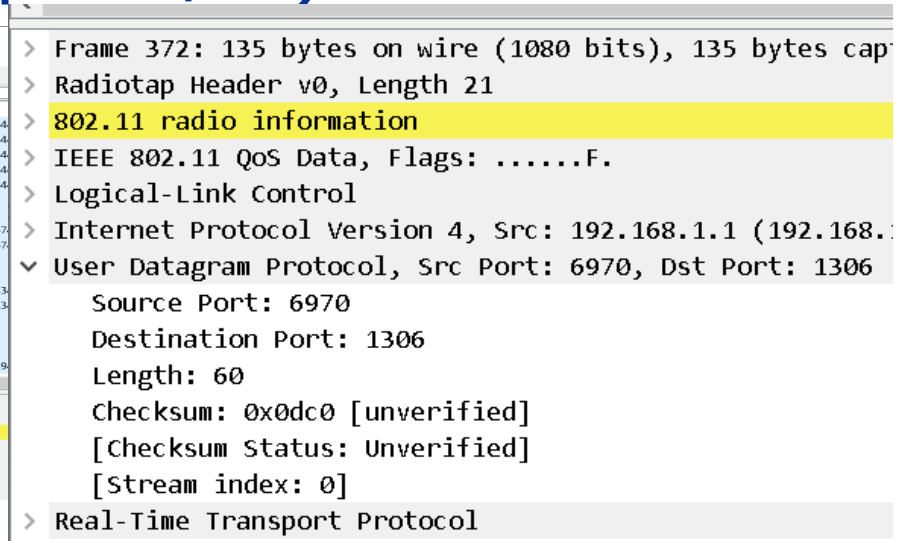

```
> User Datagram Protocol, Src Port: 6971, Dst Port: 1307
> Real-time Transport Control Protocol (Sender Report)
> Real-time Transport Control Protocol (Source description)
```

```
> User Datagram Protocol, Src Port: 6973, Dst Port: 1307
> Real-time Transport Control Protocol (Sender Report)
> Real-time Transport Control Protocol (Source description)
```

Decode as Dynamic RTP Type 96/97

- This time we succeed to decode udp streams as RTP (DynamicRTP-Type-96/97)





Recognized as RTPType-96/97

- RTP Type-96 means dynamic assignment RFC3551 by IANA https://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml

RTP Payload Types (PT) for standard audio and video encodings - Closed

Registration Procedure(s)
Registry closed; see [RFC3551], Section 3
Reference
[RFC3551]

96-127	dynamic	?

- Unfortunately there are few information to determine the codec.

SSRC (Synchronization source)

- Use “Telephony>RTP streams” to check SSRC

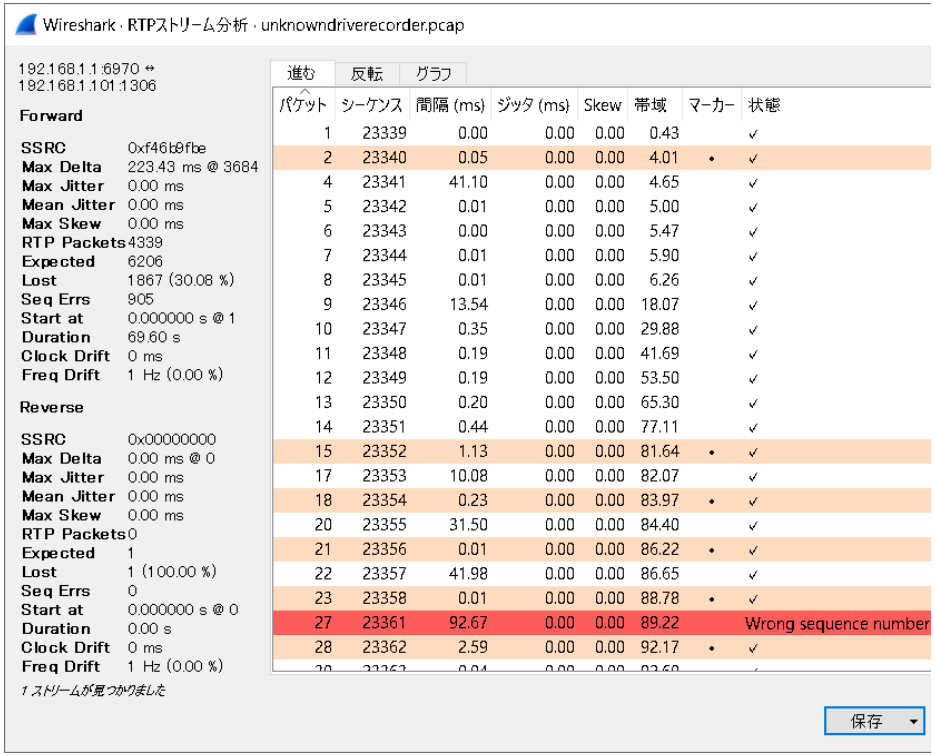
Wireshark · RTPストリーム · unknowndriverecorder.pcap

送信元アドレス	送信元ポート	宛先アドレス	宛先ポート	SSRC	ペイロード	パケット	欠落	最大間隔(ms)	最大ジッタ	平均ジッタ	状態
192.168.1.1	6972	192.168.1.101	1306	0x418fedcf	RTPType-97	2279	1 (984%)	0.000	0.000	0.000	
192.168.1.1	6970	192.168.1.101	1306	0xf46b9fbe	RTPType-96	4339	1 (1,867%)	223.434	0.000	0.000	

SSRC means 32 bit id of RTP source and the receiver for playing.

- Select video stream (UDP 6970 SSRC 0x1f46b9fbe) and click analyze button.
- We can check payload type, number of packets, lost, max delta, max jitter, average jitter and so on.

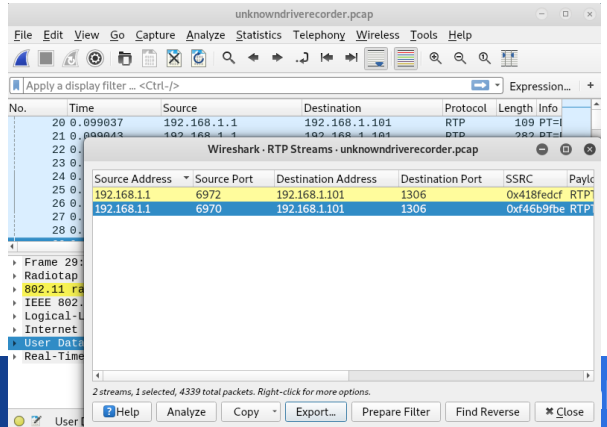
RTP Stream Analysis

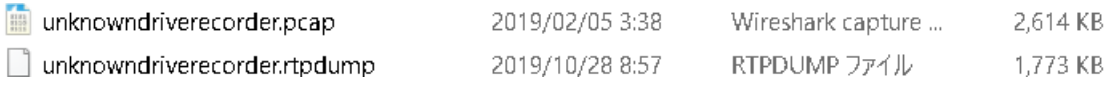


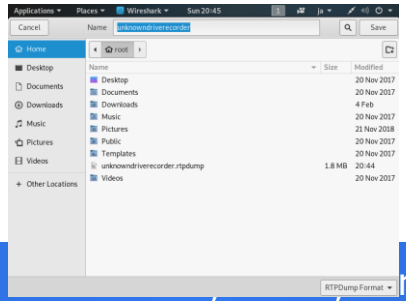
We can check Skew value, error and so on.
If the codec is the common such as G.711, we just press “Play Stream” button to play audio, or “Save” as au file format, but in this case, we don’t know the codec

Export streams as RTPDump format

- Go back to RTP Stream, select video stream (UDP6970) then press “Export…” to export in RTPDump format.
- Note: Exporting rtpdump may fail in Windows10 with the newest Wireshark3.x (2.x works well)







RTP Tools (http://www.cs.columbia.edu/irt/software/rtptools/)

- RTPDump is one of RTP Tools for processing RTP data by Columbia university (http://www.cs.columbia.edu/irt/software/rtptools/)
- rtpdump parses and prints RTP packets and generating output files suitable for rtpplay and rtpsend.
- rtpplay plays back RTP sessions by rtpdump
- rtpsend generates RTP packets by rtpdump or text.
- rtptrans translates between VAT and RTP format

Send a video RTP Stream

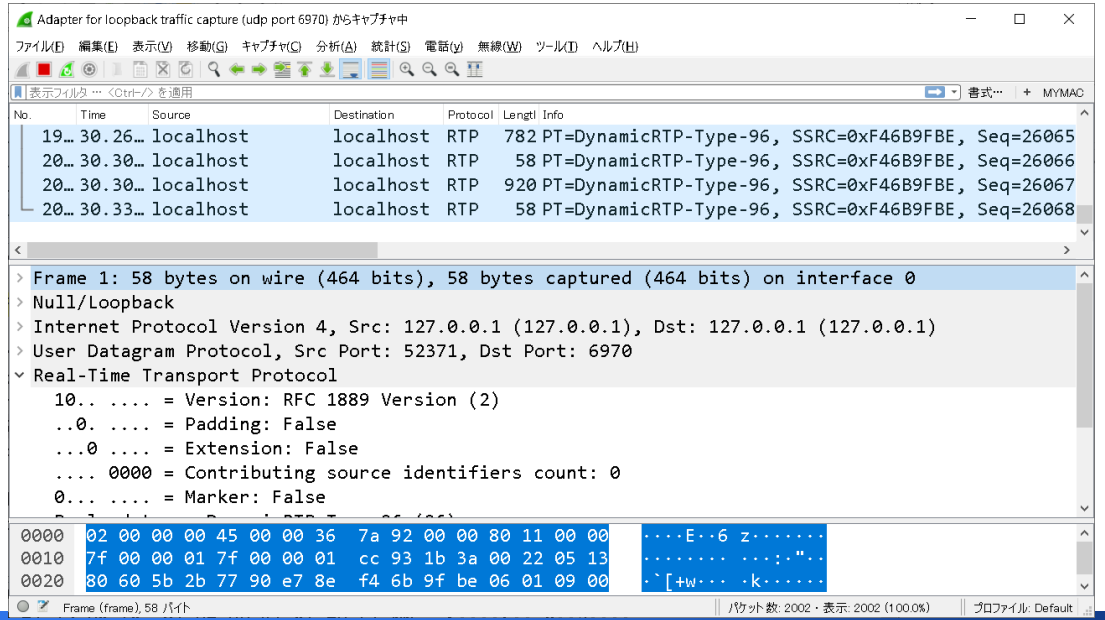
- Download RTP Tools windows rtptools-1.18 binary
  http://www.cs.columbia.edu/irt/software/rtptools/
- rtpplay [-T] [-v] [-f file] [-p profile] [-s sourceport] [-b begin] [-e end] destination/port[/ttl]
  rtpplay -T -f unknowndriverecorder.rtpdump 127.0.0.1/6970

```
rtpplay -T -f ../unknowndriverecorder.rtpdump 127.0.0.1/6970
```

- We can send the RTP video streams from trace file.
- Then we need to determine which codec RTP use

Check RTP stream sending

- Send the RTP Stream using rtpplay
  rtpplay -T -f unknowndriverecorder.rtpdump 127.0.0.1/6970
- Set capture filter as “udp port 6970”, start capturing at loopback interface.
- Look the trace file to check video RTP stream is sent.



Assume codec

- We need to determine codec, so we try most common and BSD license codec, VP8 by Google.
- A RTP video distribution needs the SDP file (Session Description Protocol) including codec information
- Smartphone app knows the codec, but we don’t.
  so let’s start assume codec manually.
  Then we create a SDP file including assumed codec.
- If there are rtsp (Real Time Streaming Protocol - RFC 2326) packets, we can use this.

Create a sample SDP file (RFC4566)

```
v=0
c=IN IP4 127.0.0.1
m=video 6970 RTP/AVP 96
a=rtpmap:96 VP8/90000
```

v: Version
c: Connection Data
m: Media Descriptions
a: Attributes

- There are example SDP description in RFC4566 https://tools.ietf.org/html/rfc4566
- Create text file, input these information as vp8.sdp

ffmpeg (ffmpeg.org)

- ffmpeg can listen streams and create movie too.
- Read settings from unknowndriverecorder.sdp, capture RTP stream, then create mkv file. (mkv is intermediate file of the video)
- ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i vp8.sdp –copyts –c copy –y ../unknowndriverecorder.mkv

-v loglevel -protocol_whitelist allowed protocols
-f format -i input file -copyts Do not process input timestamps
-c copy copy codec -y overwrite output files

Using ffmpeg to record RTP stream

ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i vp8.sdp –copyts –c copy –y ../unknowndriverecorder.mkv

```
C:¥Users¥megumi¥Desktop¥ffmpeg>ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i vp8.sdp -c
opyts -c copy -y unknowndriverecorder.mkv
[udp @ 00000256596ca680] 'circular_buffer_size' option was set but it is not supported on this build (
pthread support is required)
[udp @ 00000256596da980] 'circular_buffer_size' option was set but it is not supported on this build (
pthread support is required)
```

Streaming and recording video

- We can simulate unknown drive recorder and smartphone app by sending / receiving packets.
- Start recording and then streaming video

Streaming RTP video from trace file using rtpplay
rtpplay -T -f unknowndriverecorder.rtpdump 127.0.0.1/6970

Recording RTP video from streaming
ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i unknowndriverecorder.sdp –copyts –c copy –y ../unknowndriverecorder. mkv

Codec mismatch !!

- There are mismatch of codec if ffmpeg outputs errors, so try another.

  [sdp @ 000002687e5ea380] Could not find codec parameters for stream 0 (Video: vp8, yuv420p): unspecified size
  Consider increasing the value for the 'analyzeduration' and 'probesize' options
  Output file #0 does not contain any stream

```
rtpplay -T -f unknowndriverecorder.rtpdump 127.0.0.1/6970

[sdp @ 000002687e5ea380] Could not find codec parameters for stream 0 (Video: vp8, yuv420p): unspecif
ed size
Consider increasing the value for the 'analyzeduration' and 'probesize' options
Output file #0 does not contain any stream
```

Change codec as H.264

```
v=0
c=IN IP4 127.0.0.1
m=video 6970 RTP/AVP 96
a=rtpmap:96 H264/90000
```

- create text file as h264.sdp
- H.264 aka MPEG-4 AVC is not open source but famous codec for many IOT devices.

Use H264 encoding instead of VP8

- Sending rtpplay -T -f unknowndriverecorder.rtpdump 127.0.0.1/6970

```
rtpplay -T -f unknowndriverecorder.rtpdump 127.0.0.1/6970
```

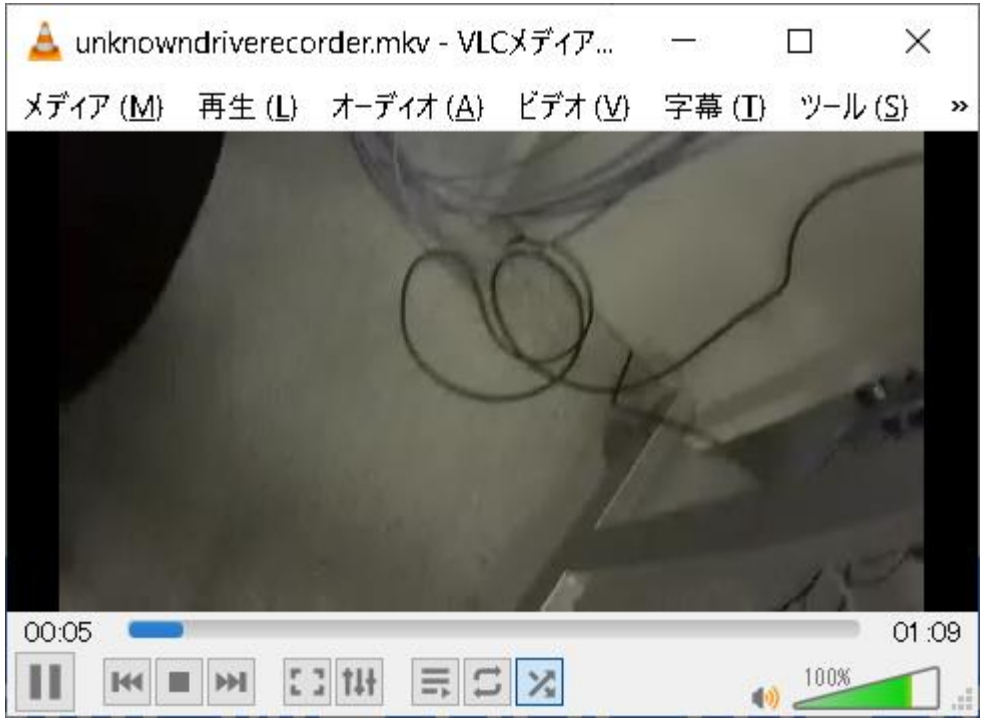
- Receiving ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i h264.sdp -copyts -c copy -y ../unknowndriverecorder.mkv

```
[sdp @ 000001def605a340] RTP: missed 2 packets
[NULL @ 000001def6069d00] missing picture in access unit with size 18
h264.sdp: Unknown error
```

- Output contains some error but we can get mkv data

```
2019/10/28  11:16             1,760,295 unknowndriverecorder.mkv
```

Finally we got the video !!



Try again to send and receive the RTP stream, ffmpeg outputs mkv file (unknowndriverecorder.mkv)
Let's play mkv file using VLC Player

Use Wireshark for multimedia

- We need to config sdp file with adequate timing and fps and other value to make a better quality video.
- And there are no audio, we need to do the same things for RTP stream in UDP6972 port

Wireshark is a nice multimedia analyzer too

Wireshark is not only the most used network analyzer but also a useful audio and video capturing, decoding, processing tool, USE WIRESHARK

Thank you for attending !!

Please complete the SharkFest Europe app-based survey

Supplemental file

http://www.ikeriri.ne.jp/sharkfest

ikeriri network service

http://www.ikeriri.ne.jp