



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



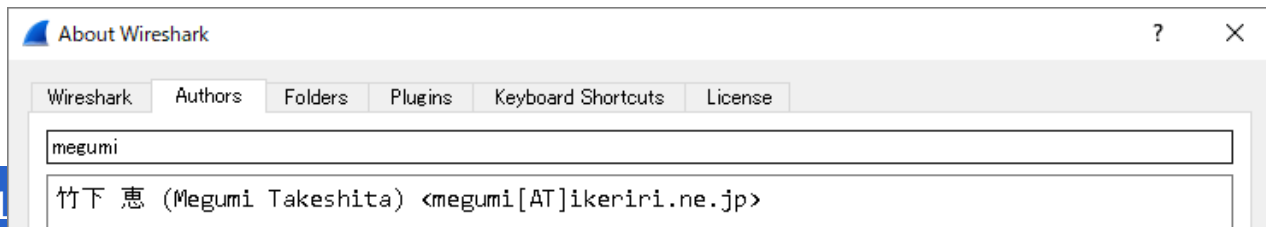
Megumi Takeshita, ikeriri network service



- 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



#sf1





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)

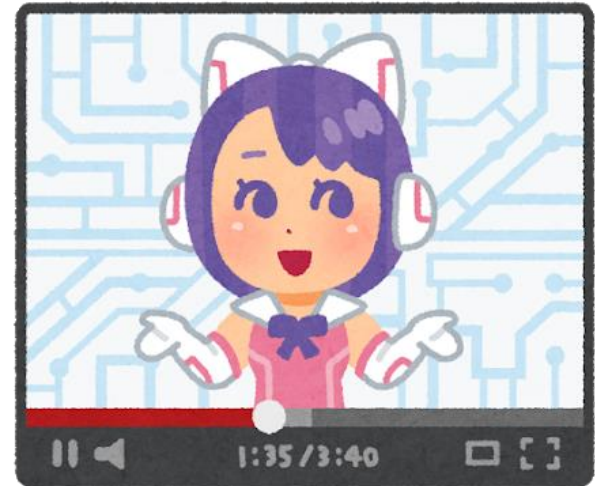
Trace files/tools www.ikeriri.ne.jp/sharkfest/



Live streaming and streaming video



- There are many live or on-demand video services in today's 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

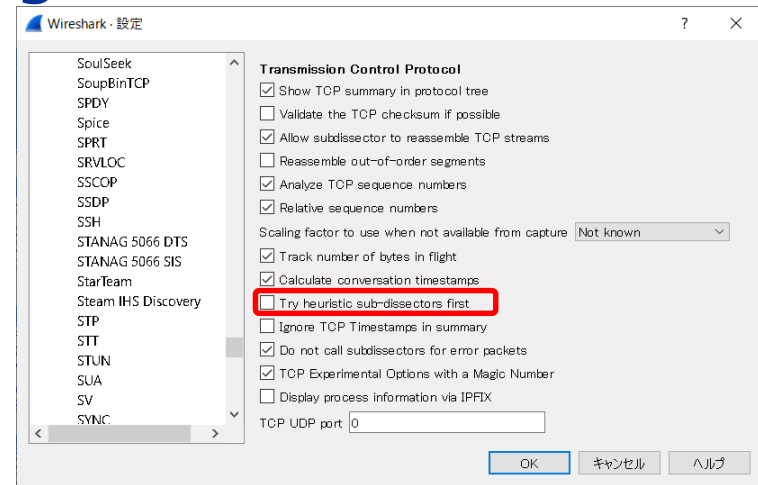
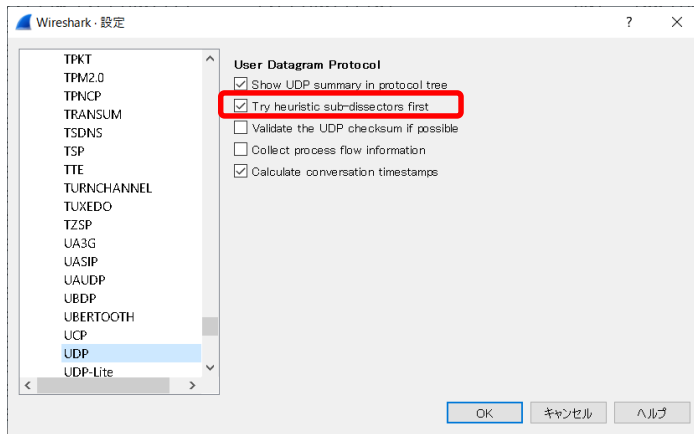


- Plain RTMP uses TCP port 1935, but we use web browser to watch the video, so RTMPT (RTMP Tunneled) is used between web server and browser.
- RTMPS uses HTTPS to encrypt RTMP, is known as RTMP over SSL/TLS, and is UDP version of RTMP
- Some web service uses different port number. In those case, check “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.



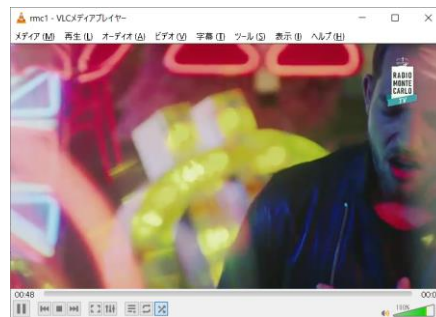
- Or set manually assign the port by “Decode As...”



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"

rtmp.pcapng

ファイル(F) 編集(E) 表示(V) 移動(G) キャプチャ(C) 分析(A) 統計(S) 電話(y) 無線(W) ツール(T) ヘルプ(H)

rtmpt

No.	Time	Source	Destination	Protocol	Length	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



Real Time Messaging Protocol (AMF0 Command connect('live'))

[Response to this call in frame: 15](#)

RTMP Header

00.. = Format: 0
..00 0011 = Chunk Stream ID: 3
Timestamp: 0
Body size: 198
Type ID: AMF0 Command (0x14)
Stream ID: 0

RTMP Body

String 'connect'

AMF0 type: String (0x02)
String length: 7
String: connect

Number 1

AMF0 type: Number (0x00)
Number: 1

Object (8 items)

AMF0 type: Object (0x03)
> Property 'app' String 'live'
> Property 'flashVer' String 'LNX 9.0.124.2'
> Property 'tcUrl' String 'rtmp://fms.105.net:1935/live'
> Property 'fpad' Boolean false
> Property 'capabilities' Number 15
> Property 'audioCodecs' Number 4071
> Property 'videoCodecs' Number 252
> Property 'videoFunction' Number 1
End Of Object Marker

> Ethernet II, Src: IntelCor_a7:a1:b6 (0c:54:15:a7:a1:b6), Dst: R
> Internet Protocol Version 4, Src: 172.16.4.83 (172.16.4.83), Dst
> Transmission Control Protocol, Src Port: 28208, Dst Port: 1935,
> Real Time Messaging Protocol (AMF0 Command connect('live'))

[Response to this call in frame: 15](#)

#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

```

v Real Time Messaging Protocol (Video Data)
  v RTMP Header
    01.. .... = Format: 1
    ..00 0111 = Chunk Stream ID: 7
    Timestamp delta: 40
    Timestamp: 25173458 (calculated)
    Body size: 3163
    Type ID: Video Data (0x09)
  v RTMP Body
    v Control: 0x27 (inter-frame H.264)
      0010 .... = Type: inter-frame (2)
      .... 0111 = Format: H.264 (7)
      Video data: 010000280000c52419a66cc05dbe07d8cbbbf715ddfc4...
  > Real Time Messaging Protocol (Audio Data)
v Real Time Messaging Protocol (Audio Data)
  v RTMP Header
    01.. .... = Format: 1
    ..00 0110 = Chunk Stream ID: 6
    Timestamp delta: 22
    Timestamp: 8043 (calculated)
    Body size: 334
    Type ID: Audio Data (0x08)
  v RTMP Body
    v Control: 0xaf (HE-AAC 44 kHz 16 bit stereo)
      1010 .... = Format: HE-AAC (10)
      .... 11.. = Sample rate: 44 kHz (3)
      .... ..1. = Sample size: 16 bit (1)
      .... ...1 = Channels: stereo (1)
      Audio data: 01210a0f77dffe95057b2682b24601042221692491080814...
```

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 -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)

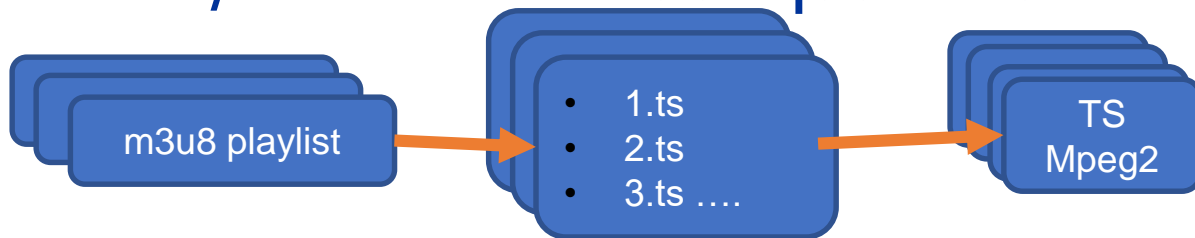


- The Flash age has been finished, off course there are still many FLV, but now we live in HTML5 generation.
- Apple may not like Adobe, so iPhone/iPad uses HLS.
<https://developer.apple.com/streaming/>
- HLS, a.k.a. HTTP Live Streaming is an HTTP-based streaming protocol by Apple. Not only Safari and 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.



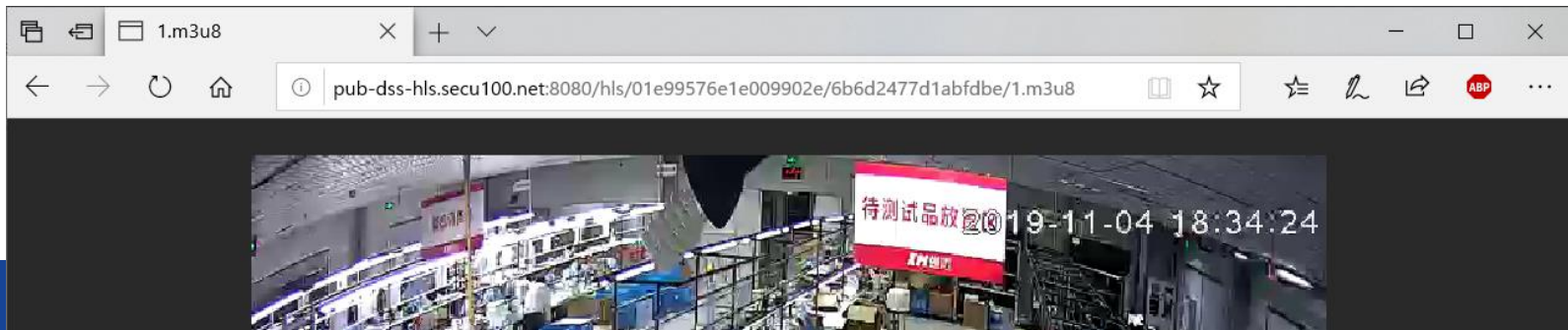


Test Network Camera Live Streaming



- 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)

No.	Time	Source	Destination	Protocol	Length	Info
6	0.086767	172.16.4.83	ec2-52-58-7...	HTTP	480	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
12	1.456887	ec2-52-58-7...	172.16.4.83	HTTP	398	HTTP/1.1 200 OK (application/x-mpegurl)
14	1.730486	172.16.4.83	ec2-52-58-7...	HTTP	590	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
16	3.668261	ec2-52-58-7...	172.16.4.83	HTTP	398	HTTP/1.1 200 OK (application/x-mpegurl)
21	14.44899	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
24	5.568968	ecs-119-3-5...	172.16.4.83	HTTP	636	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
25	5.580799	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-0.ts HTTP/1.1
36	6.009735	ecs-119-3-5...	172.16.4.83	MPEG	116	HTTP/1.1 206 Partial Content Program Association Table (PAT) [MP2T fragment of a reass...
38	6.020985	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-0.ts HTTP/1.1
47	6.455411	ecs-119-3-5...	172.16.4.83	MPEG	116	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
49	6.466544	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-1.ts HTTP/1.1
64	6.900856	ecs-119-3-5...	172.16.4.83	MPEG	756	[MP2T fragment of a reassembled packet] Program Association Table (PAT) [MP2T fragment
66	6.913420	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-2.ts HTTP/1.1
89	7.346513	ecs-119-3-5...	172.16.4.83	MPEG	1252	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
91	7.369323	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-3.ts HTTP/1.1
1	7.858879	ecs-119-3-5...	172.16.4.83	MPEG	544	[MP2T fragment of a reassembled packet] Program Association Table (PAT) [MP2T fragment
1	7.859056	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
1	8.291365	ecs-119-3-5...	172.16.4.83	HTTP	702	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
1	8.303738	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-4.ts HTTP/1.1
1	8.739710	ecs-119-3-5...	172.16.4.83	MPEG	1044	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
1	8.750790	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-5.ts HTTP/1.1
1	10.225824	ecs-119-3-5...	172.16.4.83	MPEG	1304	[MP2T fragment of a reassembled packet] Program Association Table (PAT) [MP2T fragment
1	10.242416	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-6.ts HTTP/1.1
1	10.300808	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
2	10.673135	ecs-119-3-5...	172.16.4.83	MPEG	468	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
2	10.728789	ecs-119-3-5...	172.16.4.83	HTTP	747	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
2	10.738267	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-7.ts HTTP/1.1
2	11.591326	ecs-119-3-5...	172.16.4.83	MPEG	1044	HTTP/1.1 206 Partial Content Program Association Table (PAT) Table ID 0x03[Malformed P
2	11.602820	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-8.ts HTTP/1.1
2	12.454011	ecs-119-3-5...	172.16.4.83	MPEG	708	HTTP/1.1 206 Partial Content Program Association Table (PAT) [MP2T fragment of a reass
2	12.740784	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
2	13.166645	ecs-119-3-5...	172.16.4.83	HTTP	769	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
2	13.179920	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-9.ts HTTP/1.1
2	15.176699	172.16.4.83	ecs-119-3-5...	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
4	15.726079	ecs-119-3-5...	172.16.4.83	HTTP	771	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
4	15.739466	172.16.4.83	ecs-119-3-5...	HTTP	547	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-10.ts HTTP/1.1
5	16.599441	ecs-119-3-5...	172.16.4.83	MPEG	1492	Table ID 0x06[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
5	16.614427	172.16.4.83	ecs-119-3-5...	HTTP	547	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-11.ts HTTP/1.1

時間	Source	Destination	Protocol	Length	Info
0.086767	172.16.4.83	ec2-52-58-71-125.eu-central-1.compute.amazonaws.com	HTTP	480	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
1.456887	172.16.4.83	172.16.4.83	HTTP	398	HTTP/1.1 200 OK (application/x-mpegurl)
1.730486	172.16.4.83	ec2-52-58-71-125.eu-central-1.compute.amazonaws.com	HTTP	590	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
3.668261	172.16.4.83	172.16.4.83	HTTP	398	HTTP/1.1 200 OK (application/x-mpegurl)
4.414899	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
5.568968	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	HTTP	636	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
5.580799	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-0.ts HTTP/1.1
6.009735	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	116	HTTP/1.1 206 Partial Content Program Association Table (PAT) [MP2T fragment of a reass...
6.020985	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-0.ts HTTP/1.1
6.455411	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	116	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
6.466544	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-1.ts HTTP/1.1
6.900856	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	756	[MP2T fragment of a reassembled packet] Program Association Table (PAT) [MP2T fragment
6.913420	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-2.ts HTTP/1.1
7.346513	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	1252	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
7.369323	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-3.ts HTTP/1.1
7.858879	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	544	[MP2T fragment of a reassembled packet] Program Association Table (PAT) [MP2T fragment
7.859056	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
8.291365	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	HTTP	702	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
8.303738	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-4.ts HTTP/1.1
8.739710	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	1044	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
8.750790	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-5.ts HTTP/1.1
10.225824	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	1304	[MP2T fragment of a reassembled packet] Program Association Table (PAT) [MP2T fragment
10.242416	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-6.ts HTTP/1.1
10.300808	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
10.673135	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	468	Table ID 0x03[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
10.728789	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	HTTP	747	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
10.738267	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-7.ts HTTP/1.1
11.591326	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	1044	HTTP/1.1 206 Partial Content Program Association Table (PAT) Table ID 0x03[Malformed P
11.602820	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-8.ts HTTP/1.1
12.454011	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	708	HTTP/1.1 206 Partial Content Program Association Table (PAT) [MP2T fragment of a reass
12.740784	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
13.166645	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	HTTP	769	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
13.179920	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-9.ts HTTP/1.1
15.176699	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	546	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1.m3u8 HTTP/1.1
15.726079	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	HTTP	771	HTTP/1.1 206 Partial Content (application/vnd.apple.mpegurl)
15.739466	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	547	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-10.ts HTTP/1.1
16.599441	ecs-119-3-50-168.compute.amazonaws.com	172.16.4.83	MPEG	1492	Table ID 0x06[Malformed Packet] Program Map Table (PMT) video-stream [MP2T fragment of
16.614427	172.16.4.83	ecs-119-3-50-168.compute.amazonaws.com	HTTP	547	GET /hls/01e99576e1e009902e/6b6d2477diabfde/1-11.ts HTTP/1.1



HLS Flows(#6/#12)

GET /hls/01e99576e1e009902e/6b6d2477d1abfdbbe/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: *

Client requests m3u8 Playlist

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

#EXTM3U

#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=512000

<http://119.3.50.168:8088/hls/01e99576e1e009902e/6b6d2477d1abfdbbe/1.m3u8>



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: *
```

Client requests alternative m3u8 Playlist

Server responds bandwidth and m3u8
playlist including links to TS files

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



HLS Flows (#22 / #24)



```
GET /hls/01e99576e1e009902e/6b6d2477d1abfdbe/1.m3u8 HTTP/1.1
Referer: http://pub-dss-hls.secu100.net:8080/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

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

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

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



mpeg_dump_lua Scripts



Wireshark extension to dump MPEG2 transport stream packets to file, removing the network headers and leaving just an MPEG2 transport stream.

To use this script:

1. Download the attachment [mpeg_packets_dump.lua](#)
2. Save it in the Wireshark home directory e.g. c:\Program Files\Wireshark -- as "mpeg_packets_dump.lua"
3. Edit init.lua in the Wireshark home directory and add the following line:
dofile("mpeg_packets_dump.lua")
4. Restart Wireshark to add the extension
5. Capture some traffic which includes some MPEG transport packets, for example, it has been tested with MPEG transmitted via UDP multicast.
6. Stop the capture, and select Tools -> Dump MPEG TS Packets
7. Enter the file where the mpeg stream should be saved.
8. In order to select only one of many streams, enter a wireshark filter -- expression, or you can leave the filter blank.
9. Press okay. Any MPEG packets in the current capture which were detected by the MPEG dissector and that match your filter will be dumped to your output file.

Tested with Wireshark 1.4.3. 2011-04-01. ryan dot gorsuch - echostar com Modified and tested with Wireshark 1.11.3. 2014-02-17. hadrielk at yahoo dot com

- 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")`

The screenshot shows the Wireshark interface for a file named 'hls.pcapng'. The menu bar includes options like 'ファイル(F)', '編集(E)', '表示(V)', '移動(G)', 'キャプチャ(C)', '分析(A)', '統計(S)', '電話(y)', '無線(W)', 'ツール(T)', and 'ヘルプ(H)'. The 'Tools' menu is open, showing options: 'ファイアウォール ACL ルール', 'Dump MPEG TS Packets', and 'Lua'. The packet list table below shows a single entry:

No.	Time	Source	Destination	Protocol	
42	6.455409	ecs-119-3-5...	172.16.4.83	TCP	1514 8088 → 34445

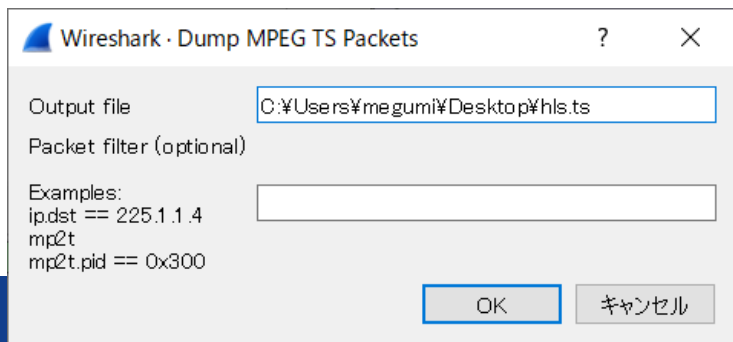
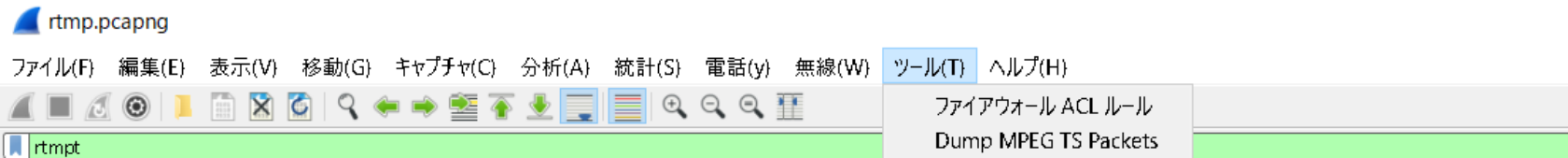


Export MPEG2(TS) from HLS



```
init.lua - TeraPad
ファイル(F) 編集(E) 検索(S) 表示(V) ウィンドウ(W) ツール(T) ヘルプ(H)
695 ↓
696 if not running_superuser or run_user_scripts_when_superuser then
697     dofile(DATA_DIR.."console.lua")↓
698 end↓
699 --dofile(DATA_DIR.."dtd_gen.lua")↓
700 [EOF]
```

- 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 today's 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

Wireshark · VoIP通話 · voip.pcapng

開始時間	停止時間	初期話者	送信元	宛先	プロトコル	時間	パケット	状態	コメント
269.939789	291.960672	voip3022.agile.ne.jp	"0906508253....196.187.6>	<sip:0000002115@...ce57d83031d79c2>	SIP	00:00:22	17	COMPLETED	INVITE 200 200

- 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

Wireshark · Call Flow - voip.pcapng

時間 voip3022.agile.ne.jp 192.168.0.7 コメント

Time	Length	Protocol	Details
269.939789	5060	INVITE SDP (green)	SIP INVITE From: "09065082538" <sip:0906...
270.158223	5060	100 Trying	SIP Status 100 Trying
270.725743	5060	180 Ringing	SIP Status 180 Ringing
271.126366	5060	INVITE SDP (green)	SIP INVITE From: "09065082538" <sip:0906...
271.162886	5060	180 Ringing	SIP Status 180 Ringing
271.510926	5060	INVITE SDP (green)	SIP INVITE From: "09065082538" <sip:0906...
271.523430	5060	180 Ringing	SIP Status 180 Ringing
275.185635	5060	200 OK SDP (green)	SIP Status 200 OK
275.677902	5060	200 OK SDP (green)	SIP Status 200 OK
276.076032	5060	ACK	SIP Request INVITE ACK 200 CSeq:102
276.152881	18522	RTP (RTPType=1...)	RTP, 4 packets. Duration: 10.349s SSRC: 0x...
276.925111	5060	ACK	SIP ACK From: "09065082538" <sip:0906508...
277.435776	18522	RTP (RTPType=1...)	RTP, 567 packets. Duration: 13.033s SSRC: ...
290.485711	5060	BYE	SIP Request BYE CSeq:103
290.543189	5060	200 OK	SIP Status 200 OK
291.256742	5060	BYE	SIP Request BYE CSeq:103
291.257234	5060	200 OK	SIP Status 200 OK
291.960253	5060	BYE	SIP Request BYE CSeq:103

2 ノード, 19 項目

Save As... 閉じる ヘルプ



SIP (Session Initiation Protocol)



```
voip.pcapng
ファイル 編集 表示 移動 切り取り 分析 統計 電話 無線 ツール ヘルプ
表示形式: 表示形式... Ctrl+F を適用 書式... MYMAC
No. Time Source Destination Protocol Length Info
936 269.9... voip3022.agile... 192.168... SIP... 10... Request: INVITE sip:000002115@192.168.0.7:5060
037 270.0... 192.168.0.4... 192.168... NRNS... 02 Name query NR TSUWUMOTAN/20...
> Frame 936: 1083 bytes on wire (8664 bits), 1083 bytes captured (8664 bits) on interface 0
> Ethernet II, Src: NecPlatf_36:a2:78 (1c:b1:7f:36:a2:78), Dst: Toshiba_ef:fd:53 (e8:e0:b7:ef:fd:53)
> Internet Protocol Version 4, Src: voip3022.agile.ne.jp (61.196.187.6), Dst: 192.168.0.7 (192.168.0.7)
> User Datagram Protocol, Src Port: 5060, Dst Port: 5060
< Session Initiation Protocol (INVITE)
  < Request-Line: INVITE sip:000002115@192.168.0.7:5060;rinstance=fce57d83031d79c2 SIP/2.0
    Method: INVITE
    > Request-URI: sip:000002115@192.168.0.7:5060;rinstance=fce57d83031d79c2
      [Resent Packet: False]
    > Message Header
  < Message Body
    < Session Description Protocol
      Session Description Protocol Version (v): 0
      > Owner/Creator, Session Id (o): root 23847 23847 IN IP4 61.196.187.6
      Session Name (s): session
      > Connection Information (c): IN IP4 61.196.187.6
      > Time Description, active time (t): 0 0
      > Media Description, name and address (m): audio 18522 RTP/AVP 0 110 3 101
      > Media Attribute (a): rtpmap:0 PCMU/8000
      > Media Attribute (a): rtpmap:110 speex/8000
      > Media Attribute (a): rtpmap:3 GSM/8000
      > Media Attribute (a): rtpmap:101 telephone-event/8000
      > Media Attribute (a): fmp:101 0-16
0310 32 39 32 0d 0a 0d 0a 76 3d 30 0d 0a 6f 3d 72 6f 292...v =0...o=pc
0320 5f 74 20 32 33 38 34 37 20 32 33 38 34 37 20 49 0t 23847 23847 I
0330 4e 20 49 50 34 20 36 31 2e 31 39 36 2e 31 38 37 N IP4 61 .196.187
0340 2e 36 0d 0a 73 3d 73 65 73 73 69 6f 6e 0d 0a 63 .6...s=se sson...c
0350 3d 49 4e 20 49 50 34 20 36 31 2e 31 39 36 2e 31 =IN IP4 61.196.1
798-801 1/1件 Owner Username (sdpowner.username) パケット数: 1627 表示: 1627 (100.0%) プロファイル: Default
```

- 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)



```
voip.pcapng
ファイル 編集(E) 表示(V) 移動(G) キャプチャ(C) 分析(A) 統計(S) 電話(y) 無線(W) ツール(T) ヘルプ(H)
表示フィルタ... <Ctrl-F> を適用
No. Time Source Destination Protocol Length Info
955 277.4... voip3022.agile... 192.168... RTP 214 PT=ITU-T G.711 PCMU
956 277.4... voip3022.agile... 192.168... RTP 214 PT=ITU-T G.711 PCMU
> Frame 955: 214 bytes on wire (1712 bits), 214 bytes captured (1712
> Ethernet II, Src: NecPlatf_36:a2:78 (1c:b1:7f:36:a2:78), Dst: Tosh
> Internet Protocol Version 4, Src: voip3022.agile.ne.jp (61.196.187
> User Datagram Protocol, Src Port: 18522, Dst Port: 5062
v Real-Time Transport Protocol
> [Stream setup by SDP (frame 949)]
10.. .... = Version: RFC 1889 Version (2)
..0. .... = Padding: False
...0 .... = Extension: False
.... 0000 = Contributing source identifiers count: 0
0... .... = Marker: False
Payload type: ITU-T G.711 PCMU (0)
Sequence number: 48268
[Extended sequence number: 48268]
Timestamp: 2434837936
Synchronization Source identifier: 0x2de8a9b6 (770222518)
Payload: fe7dfefe7e7c7efd7d7efe7efdfe7dfdfe7dfefefe7efe7e...
```

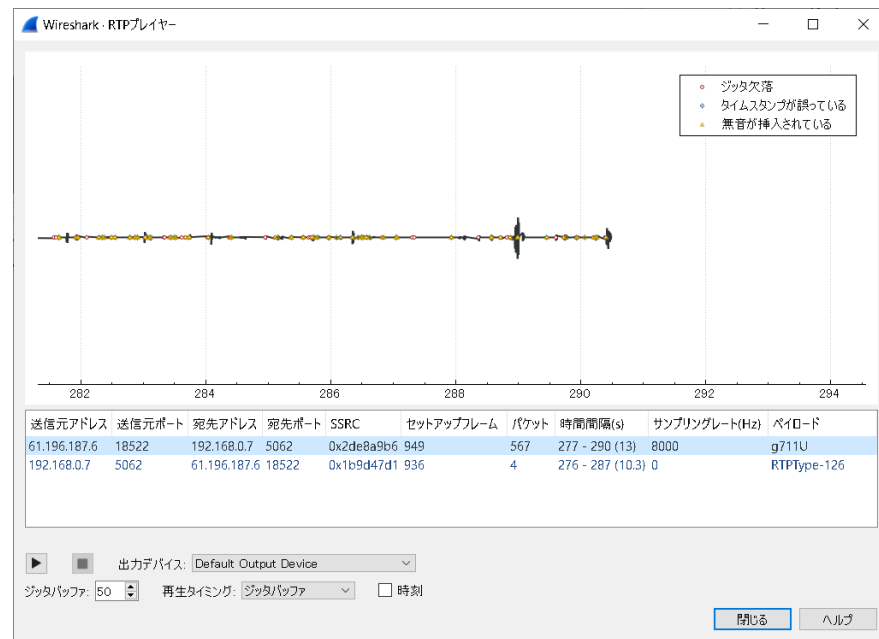
- Look at #955 RTP packet sent by UDP
- RTP packets contains Payload type, Sequence number, Timestamp and SSRC (Synchronization Source identifier) and Payload of actual media.



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



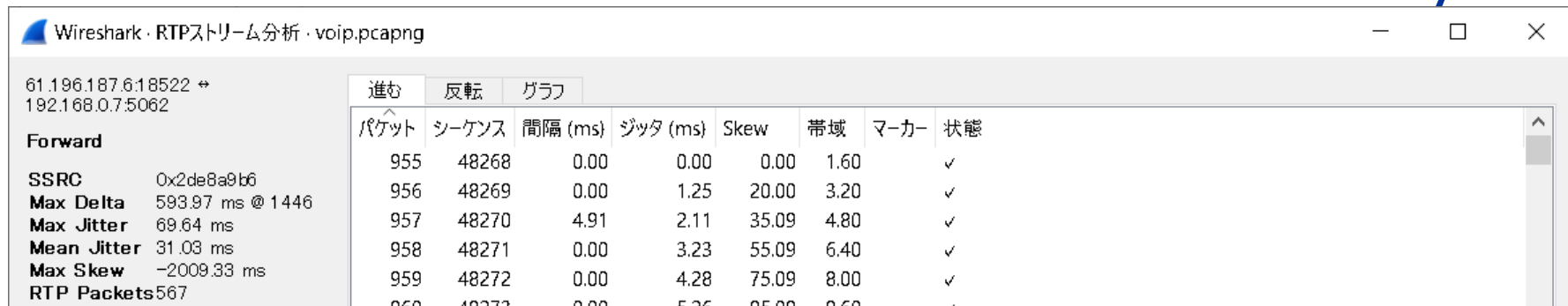
- Telephony > RTP Streams summarize RTP streams



Wireshark · RTPストリーム · voip.pcapng

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
voip3022.agile.ne.jp	18522	192.168.0.7	5062	0x2de8a9b6	g711U	567	1 (0%)	593.971	69.638	31.030	
192.168.0.7	5062	voip3022.agile.ne.jp	18522	0x1b9d47d1	RTPTYPE-126	4	1 (-2%)	10348.744	0.000	0.000	•

- Shift click to select both streams and click Analyze



Wireshark · RTPストリーム分析 · voip.pcapng

61.196.187.6:18522 ↔ 192.168.0.7:5062

		進む	反転	グラフ							
	パケット	シーケンス	間隔 (ms)	ジッタ (ms)	Skew	帯域	マーカー	状態			
Forward											
SSRC	0x2de8a9b6	955	48268	0.00	0.00	1.60		✓			
Max Delta	593.97 ms @ 1446	956	48269	0.00	1.25	20.00	3.20	✓			
Max Jitter	69.64 ms	957	48270	4.91	2.11	35.09	4.80	✓			
Mean Jitter	31.03 ms	958	48271	0.00	3.23	55.09	6.40	✓			
Max Skew	-2009.33 ms	959	48272	0.00	4.28	75.09	8.00	✓			
RTP Packets	567	960	48273	0.00	5.76	95.00	9.60	✓			



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.

Wireshark · RTPストリーム分析 · voip.pcapng

61.196.187.6:18522 ↔
192.168.0.7:5062

		進む	反転	グラフ					
		パケット	シーケンス	間隔 (ms)	ジッタ (ms)	Skew	帯域	マーカー	状態
Forward		955	48268	0.00	0.00	0.00	1.60		✓
SSRC	0x2de8a8b6	956	48269	0.00	1.25	20.00	3.20		✓
Max Delta	593.97 ms @ 1446	957	48270	4.91	2.11	35.09	4.80		✓
Max Jitter	69.64 ms	958	48271	0.00	3.23	55.09	6.40		✓
Mean Jitter	31.03 ms	959	48272	0.00	4.28	75.09	8.00		✓
Max Skew	-2009.33 ms	960	48273	0.00	5.26	95.09	9.60		✓
RTP Packets	567	961	48274	0.00	6.18	115.08	11.20		✓
Expected	567	962	48275	0.00	7.05	135.08	12.80		✓
Lost	0 (0.00 %)	963	48276	0.36	7.83	154.72	14.40		✓
Seq Errs	0	964	48277	0.00	8.59	174.72	16.00		✓
Start at	277.435776 s @ 955	965	48278	0.00	9.31	194.72	17.60		✓
Duration	13.03 s	966	48279	0.00	9.98	214.72	19.20		✓
Clock Drift	-11799 ms	967	48280	3.68	10.37	231.04	20.80		✓
Freq Drift	758 Hz (-90.53 %)	968	48281	0.00	10.97	251.03	22.40		✓
Reverse		969	48282	35.54	11.26	235.49	24.00		✓
SSRC	0x1b8d47d1	970	48283	0.46	11.78	255.03	25.60		✓
Max Delta	10348.74 ms @ 1409	972	48284	179.34	21.00	95.69	27.20		✓
Max Jitter	0.00 ms	973	48285	10.82	20.26	104.87	28.80		✓
Mean Jitter	0.00 ms	974	48286	0.00	20.24	124.86	30.40		✓
Max Skew	0.00 ms	975	48287	0.00	20.23	144.86	32.00		✓
RTP Packets	4	976	48288	0.00	20.21	164.86	33.60		✓
Expected	2	977	48289	0.00	20.20	184.86	35.20		✓
Lost	-2 (-100.00 %)	978	48290	0.00	20.19	204.86	36.80		✓
Seq Errs	2	979	48291	0.00	20.18	224.86	38.40		✓
Start at	276.152881 s @ 951	980	48292	0.00	20.18	244.86	40.00		✓
Duration	10.35 s								
Clock Drift	0 ms								
Freq Drift	1 Hz (0.00 %)								
Forward to reverse									
start diff	-1.282895 s @ -4								

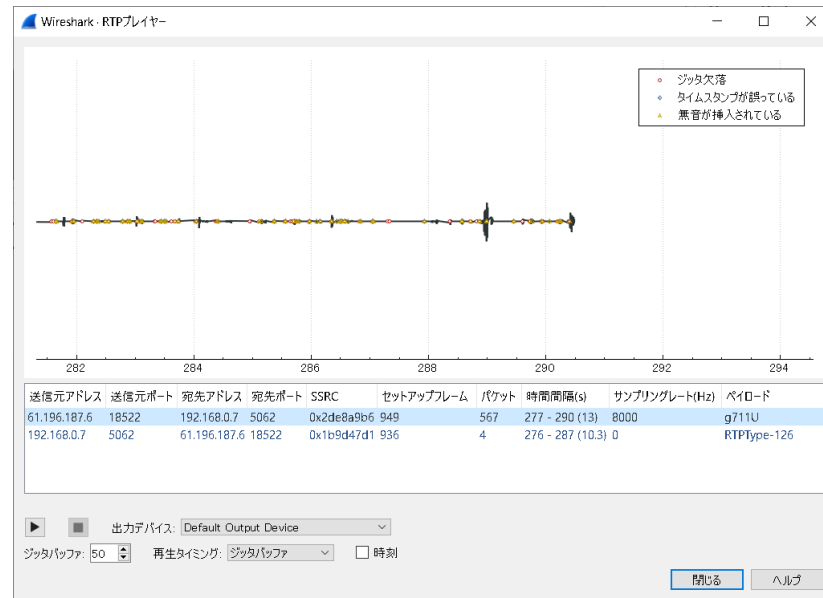
2 ストリームが覆つかりました



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...

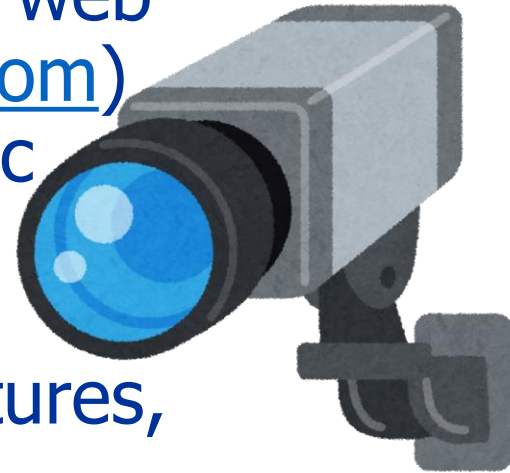




Surveillance camera



- 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.

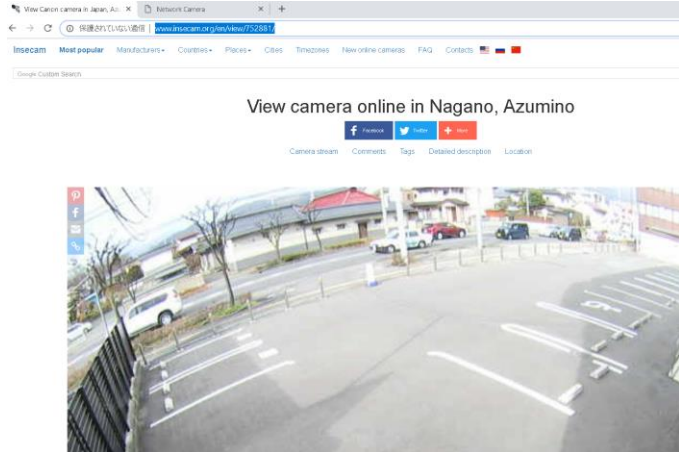




Canon camera in Nagano, Japan



- 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"



Country	Japan
Country code	.JP
Region	Nagano
City	Azumino
Latitude	36.263000
Longitude	137.817000
ZIP	398-7101
Timezone	+09:00
Manufacturer	Canon





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.

canon-camera-nagano.pcapng

ファイル(F) 編集(E) 表示(V) 移動(G) キャプチャ(C) 分析(A) 統計(S) 電話(y) 無線(W) ツール(T) ヘルプ(H)

表示フィルタ ... <Ctrl-/> を適用

No.	Time	Source	Destination	Protocol	Length	Info
8	0.007...	s243097.dynamic...	xps15.lo...	TCP	14...	8081 → 25308 [ACK] Seq=1 Ack=1 Win=6284 Len=1414
9	0.008...	s243097.dynamic...	xps15.lo...	TCP	14...	8081 → 25308 [ACK] Seq=1415 Ack=1 Win=6284 Len=...
10	0.008...	s243097.dynamic...	xps15.lo...	TCP	14...	8081 → 25308 [ACK] Seq=2829 Ack=1 Win=6284 Len=...
11	0.008...	s243097.dynamic...	xps15.lo...	TCP	14...	8081 → 25308 [ACK] Seq=4243 Ack=1 Win=6284 Len=...
12	0.008...	s243097.dynamic...	xps15.lo...	TCP	14...	8081 → 25308 [ACK] Seq=5657 Ack=1 Win=6284 Len=...

> Frame 8: 1468 bytes on wire (11744 bits), 1468 bytes captured (11744 bits) on interface 0

> Ethernet II, Src: Yamaha_6c:82:28 (00:a0:de:6c:82:28), Dst: IntelCor_a7:a1:b6 (0c:54:15:a7:a1:b6)

> Internet Protocol Version 4, Src: s243097.dynamic.ppp.asahi-net.or.jp (220.157.243.97), Dst: xps15

> Transmission Control Protocol, Src Port: 8081, Dst Port: 25308, Seq: 1, Ack: 1, Len: 1414



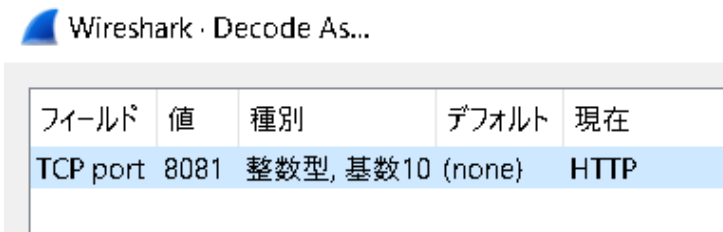
Decode TCP 8081 as HTTP



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 HTTP so try to decode TCP 8081 as HTTP.
- Select packet and right click to choose "Decode As..." and set 8081 as 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

The screenshot shows the Wireshark interface with the following details:

- Packet List:** Frame 15, Time 1.000000000, Source s243097.dynamic_xps15.lo, Destination 243007, Protocol TCP, Length 9608 bytes.
- Packet Details:**
 - ✓ Hypertext Transfer Protocol
 - HTTP/1.1 200 OK\r\n
 - Pragma: no-cache\r\n
 - Cache-Control: no-cache\r\n
 - Accept-Ranges: none\r\n
 - Content-Type: image/jpeg\r\n
 - Livescope-Frame-Number: 161\r\n
 - Livescope-Status: 0\r\n
 - Content-Length: 9608\r\n
 - Date: Sun, 09 Dec 2018 01:10:36 GMT\r\n
 - Server: VB\r\n
 - \r\n
 - [HTTP response 1/520]
 - [Next request in frame: 17]
 - [Next response in frame: 56]
 - [Request URI: http://220.157.243.97:8081/~vvhhttp-01-/image.cgi?s=d55b-f6dea041&seq=0.756628088]
 - File Data: 9608 bytes
 - [Expert Info (Note/Malformed): HTTP body subdissector failed, trying heuristic subdissector]
 - ✓ Media Type
 - Media type: image/jpeg (9608 bytes)
- Packet Bytes:** 0000 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d HTTP/1.1 200 OK
0010 0a 50 72 61 67 6d 61 5a 20 6e 6f 2d 63 61 63 68 Pragma: no-cach
0020 65 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 6f 6c Cache-Control

The right pane shows the selected frame (Frame 15) with the media type (image/jpeg) and a preview of the image. The image is a wide-angle shot of a street scene with buildings and a car.



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

パケット	ホスト名	コンテンツタイプ	サイズ	ファイル名
15	220.157.243.97:8081	image/jpeg	9608 bytes	image.cgi?s=d55b-f6dea041&sseq=0.4165893778
24			1274 bytes	
26			1414 bytes	
33			1414 bytes	
34			844 bytes	
56	220.157.243.97:8081	image/jpeg	9622 bytes	image.cgi?s=d55b-f6dea041&sseq=0.4165893778
58			1414 bytes	
63			1414 bytes	
65			1414 bytes	
66			88 bytes	
73		image/jpeg	1363 bytes	
75			1414 bytes	
76			1253 bytes	
77			1414 bytes	
85			1308 bytes	
90			1414 bytes	
98	220.157.243.97:8081	image/jpeg	9604 bytes	image.cgi?s=d55b-f6dea041&sseq=0.4637294934

テキストフィルタ:

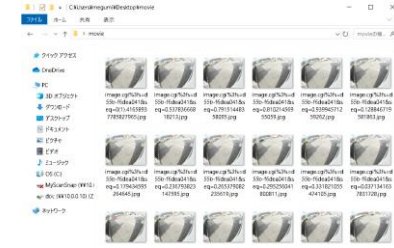
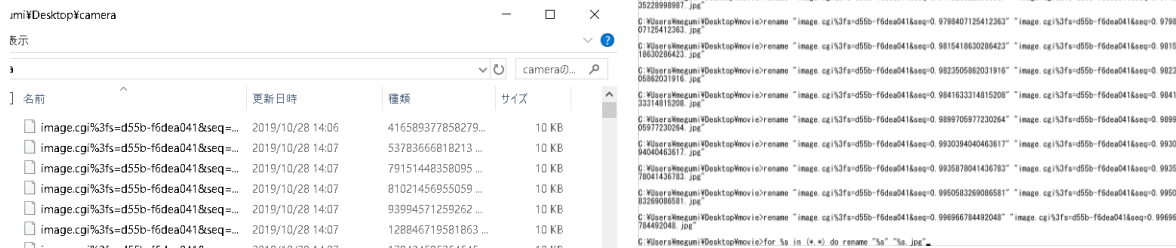
保存 すべて保存 閉じる ヘルプ



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

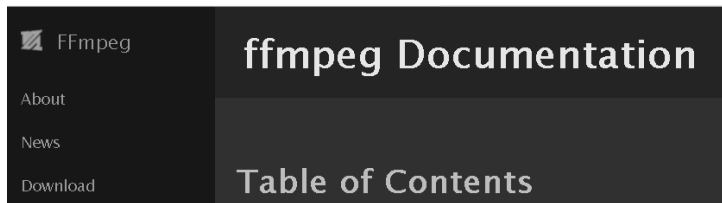


- ```
for /l %i in (1, 1, 514) do ren "camera (%i).jpg" "camera_%i.jpg"
```



# 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>)



- To set the video bitrate of the output file to 64 kbit/s:

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

- To force the frame rate of the output file to 24 fps:

```
ffmpeg -i input.avi -r 24 output.avi
```



# 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-libbmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libopenj
peg --enable-libopus --enable-libshine --enable-lbsnappy --enable-libsoxr --enable-libtheora --enable-libtwolame --enable-libvpx
--enable-libwavpack --enable-libwebp --enable-libx264 --enable-libx265 --enable-libxml2 --enable-libzimg --enable-lzma --enable-zl
ib --enable-gmp --enable-libvidstab --enable-libvorbis --enable-libvo-amrwbenc --enable-libmysofa --enable-libspeex --enable-libxv
id --enable-libaom --enable-libbfx --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



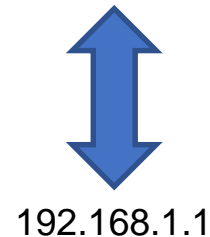
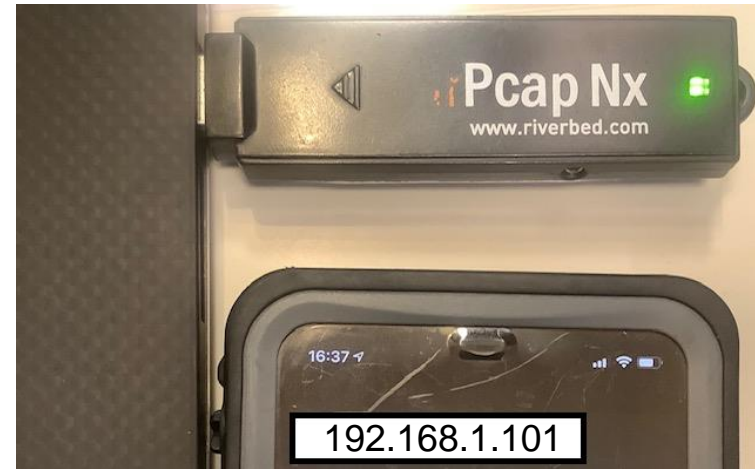
- Many Drive recorders are made in China or Taiwan.
- They use WiFi to connect smartphone App to check the camera recordings.
- Let's dissect with Wireshark
- Recorder IP :192.168.1.1



# 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





# unknowndrivererecorder.pcap



The screenshot shows a network traffic analysis tool window titled 'unknowndrivererecorder.pcap'. The main window displays a list of packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The first seven packets are highlighted in blue. The first packet is a Malformed Packet (RDT) of 109 bytes. The detailed view below shows the packet structure: Radiotap Header v0, Length 21; 802.11 radio information; IEEE 802.11 QoS Data, Flags: .....F.; Logical-Link Control; Internet Protocol Version 4, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.101 (192.168.1.101); User Datagram Protocol, Src Port: 6970, Dst Port: 1306; Real Data Transport; [Malformed Packet: RDT]. The hex dump at the bottom shows the raw bytes of the packet, with the first few bytes highlighted in blue.

| No. | Time     | Source      | Destination   | Protocol | Length | Info                |
|-----|----------|-------------|---------------|----------|--------|---------------------|
| 1   | 0.000... | 192.168.1.1 | 192.168.1.101 | RDT      | 109    | [Malformed Packet]  |
| 2   | 0.000... | 192.168.1.1 | 192.168.1.101 | RDT      | 502    | [Malformed Packet]  |
| 3   | 0.004... | 192.168.1.1 | 192.168.1.101 | UDP      | 183    | 6972 → 1306 Len=100 |
| 4   | 0.041... | 192.168.1.1 | 192.168.1.101 | RDT      | 135    | [Malformed Packet]  |
| 5   | 0.041... | 192.168.1.1 | 192.168.1.101 | RDT      | 99     | [Malformed Packet]  |
| 6   | 0.041... | 192.168.1.1 | 192.168.1.101 | RDT      | 114    | [Malformed Packet]  |
| 7   | 0.041... | 192.168.1.1 | 192.168.1.101 | RDT      | 109    | [Malformed Packet]  |

Frame 1: 109 bytes on wire (872 bits), 109 bytes captured (872 bits)  
> Radiotap Header v0, Length 21  
> 802.11 radio information  
> IEEE 802.11 QoS Data, Flags: .....F.  
> Logical-Link Control  
> Internet Protocol Version 4, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.101 (192.168.1.101)  
> User Datagram Protocol, Src Port: 6970, Dst Port: 1306  
> Real Data Transport  
> [Malformed Packet: RDT]

```
0000 00 00 15 00 2a 48 08 00 00 00 94 09 80 04 df 01 *H.....
0010 00 00 07 00 07 88 02 30 00 c4 3a be 16 f5 78 5a 0-...xZ
0020 12 43 c6 9e 5a 5a 12 43 c6 9e 5a a0 11 00 00 aa .C..ZZ..C..Z....
0030 aa 03 00 00 00 08 00 45 00 00 36 09 67 40 00 40 E..6.g@@@
```

- 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 unknowndrivererecorder.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.

```
> User Datagram Protocol, Src Port: 6972, Dst Port: 1306
 > Data (103 bytes)
 Data: 80e173bcf867eedb418fedcf001002b82117949a388ca612...
 [Length: 103]
```

- 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

Wireshark · Decode As...

| フィールド         | 値 | 種別        | デフォルト  | 現在  |
|---------------|---|-----------|--------|-----|
| UDP port 6970 |   | 整数型, 基数10 | RDT    | RTP |
| UDP port 6972 |   | 整数型, 基数10 | (none) | RTP |

- > 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)

udstreampcapng

ファイル(F) 編集(E) 表示(V) 移動(M) キャッチャ(C) 分析(A) 統計(S) 電装(t) 無線(W) ツール(T) ヘルプ(H)

表示形式: 0x00000000

| No.  | Time         | Source      | Destination   | Protocol | Length | Info                                                            |
|------|--------------|-------------|---------------|----------|--------|-----------------------------------------------------------------|
| 6663 | 69.454025655 | 192.168.1.1 | 192.168.1.101 | RTP      | 1531   | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29531, Time=2012234 |
| 6664 | 69.456583858 | 192.168.1.1 | 192.168.1.101 | RTP      | 1531   | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29532, Time=2012234 |
| 6665 | 69.456744487 | 192.168.1.1 | 192.168.1.101 | RTP      | 1531   | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29533, Time=2012234 |
| 6666 | 69.456960783 | 192.168.1.1 | 192.168.1.101 | RTP      | 1531   | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29534, Time=2012234 |
| 6667 | 69.457091254 | 192.168.1.1 | 192.168.1.101 | RTP      | 1274   | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29535, Time=2012234 |
| 6668 | 69.457105570 | 192.168.1.1 | 192.168.1.101 | UDP      | 188    | 6972 → 1306 Len=105                                             |
| 6669 | 69.457111851 | 192.168.1.1 | 192.168.1.101 | UDP      | 178    | 6972 → 1306 Len=95                                              |
| 6670 | 69.47138696  | 192.168.1.1 | 192.168.1.101 | RTP      | 109    | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29536, Time=2012237 |
| 6671 | 69.471869663 | 192.168.1.1 | 192.168.1.101 | RTP      | 495    | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29537, Time=2012237 |
| 6672 | 69.474139191 | 192.168.1.1 | 192.168.1.101 | UDP      | 184    | 6972 → 1306 Len=101                                             |
| 6673 | 69.502131561 | 192.168.1.1 | 192.168.1.101 | UDP      | 183    | 6972 → 1306 Len=100                                             |
| 6674 | 69.534443178 | 192.168.1.1 | 192.168.1.101 | RTP      | 109    | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29540, Time=2012243 |
| 6675 | 69.536252595 | 192.168.1.1 | 192.168.1.101 | RTP      | 664    | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29541, Time=2012243 |
| 6676 | 69.537848002 | 192.168.1.1 | 192.168.1.101 | UDP      | 183    | 6972 → 1306 Len=100                                             |
| 6677 | 69.568959399 | 192.168.1.1 | 192.168.1.101 | UDP      | 186    | 6972 → 1306 Len=103                                             |
| 6678 | 69.600412144 | 192.168.1.1 | 192.168.1.101 | UDP      | 183    | 6972 → 1306 Len=100                                             |
| 6679 | 69.602142466 | 192.168.1.1 | 192.168.1.101 | RTP      | 109    | PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=29544, Time=2012249 |

- Frame 372: 135 bytes on wire (1080 bits), 135 bytes captured (1080 bits) on interface e
- Radiotap Header v0, Length 21
- 802.11 radio information
- IEEE 802.11 QoS Data, Flags: .....F.
- Logical-Link Control
- Internet Protocol Version 4, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.101 (192.168.1.101)
- User Datagram Protocol, Src Port: 6970, Dst Port: 1306
  - Source Port: 6970
  - Destination Port: 1306
  - Length: 60
  - Checksum: 0x0dc0 [unverified]  
[Checksum Status: Unverified]
  - [Stream index: 0]
- Real-Time Transport Protocol



# 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 · unknowndriverrecorder.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 0xf46b9fbe) 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



Wireshark · RTPストリーム分析 · unknowndriverrecorder.pcap

192.168.1.1:6970 ↔  
192.168.1.101:1306

**Forward**

| 進む                                | 反転 | グラフ | パケット | シーケンス | 間隔 (ms) | ジッタ (ms) | Skew | 帯域    | マーカー | 状態                    |
|-----------------------------------|----|-----|------|-------|---------|----------|------|-------|------|-----------------------|
|                                   |    |     | 1    | 23339 | 0.00    | 0.00     | 0.00 | 0.43  |      | ✓                     |
|                                   |    |     | 2    | 23340 | 0.05    | 0.00     | 0.00 | 4.01  | •    | ✓                     |
| <b>SSRC</b> 0xf46b9fbc            |    |     | 4    | 23341 | 41.10   | 0.00     | 0.00 | 4.65  |      | ✓                     |
| <b>Max Delta</b> 223.43 ms @ 3684 |    |     | 5    | 23342 | 0.01    | 0.00     | 0.00 | 5.00  |      | ✓                     |
| <b>Max Jitter</b> 0.00 ms         |    |     | 6    | 23343 | 0.00    | 0.00     | 0.00 | 5.47  |      | ✓                     |
| <b>Mean Jitter</b> 0.00 ms        |    |     | 7    | 23344 | 0.01    | 0.00     | 0.00 | 5.90  |      | ✓                     |
| <b>Max Skew</b> 0.00 ms           |    |     | 8    | 23345 | 0.01    | 0.00     | 0.00 | 6.26  |      | ✓                     |
| <b>RTP Packets</b> 4339           |    |     | 9    | 23346 | 13.54   | 0.00     | 0.00 | 18.07 |      | ✓                     |
| <b>Expected</b> 6206              |    |     | 10   | 23347 | 0.35    | 0.00     | 0.00 | 29.88 |      | ✓                     |
| <b>Lost</b> 1867 (30.08 %)        |    |     | 11   | 23348 | 0.19    | 0.00     | 0.00 | 41.69 |      | ✓                     |
| <b>Seq Errs</b> 905               |    |     | 12   | 23349 | 0.19    | 0.00     | 0.00 | 53.50 |      | ✓                     |
| <b>Start at</b> 0.000000 s @ 1    |    |     | 13   | 23350 | 0.20    | 0.00     | 0.00 | 65.30 |      | ✓                     |
| <b>Duration</b> 69.60 s           |    |     | 14   | 23351 | 0.44    | 0.00     | 0.00 | 77.11 |      | ✓                     |
| <b>Clock Drift</b> 0 ms           |    |     | 15   | 23352 | 1.13    | 0.00     | 0.00 | 81.64 | •    | ✓                     |
| <b>Freq Drift</b> 1 Hz (0.00 %)   |    |     | 17   | 23353 | 10.08   | 0.00     | 0.00 | 82.07 |      | ✓                     |
|                                   |    |     | 18   | 23354 | 0.23    | 0.00     | 0.00 | 83.97 | •    | ✓                     |
| <b>Reverse</b>                    |    |     | 20   | 23355 | 31.50   | 0.00     | 0.00 | 84.40 |      | ✓                     |
| <b>SSRC</b> 0x00000000            |    |     | 21   | 23356 | 0.01    | 0.00     | 0.00 | 86.22 | •    | ✓                     |
| <b>Max Delta</b> 0.00 ms @ 0      |    |     | 22   | 23357 | 41.98   | 0.00     | 0.00 | 86.65 |      | ✓                     |
| <b>Max Jitter</b> 0.00 ms         |    |     | 23   | 23358 | 0.01    | 0.00     | 0.00 | 88.78 | •    | ✓                     |
| <b>Mean Jitter</b> 0.00 ms        |    |     | 27   | 23361 | 92.67   | 0.00     | 0.00 | 89.22 |      | Wrong sequence number |
| <b>Max Skew</b> 0.00 ms           |    |     | 28   | 23362 | 2.59    | 0.00     | 0.00 | 92.17 | •    | ✓                     |
| <b>RTP Packets</b> 0              |    |     | 29   | 23363 | 0.01    | 0.00     | 0.00 | 92.69 |      | ✓                     |
| <b>Expected</b> 1                 |    |     |      |       |         |          |      |       |      |                       |
| <b>Lost</b> 1 (100.00 %)          |    |     |      |       |         |          |      |       |      |                       |
| <b>Seq Errs</b> 0                 |    |     |      |       |         |          |      |       |      |                       |
| <b>Start at</b> 0.000000 s @ 0    |    |     |      |       |         |          |      |       |      |                       |
| <b>Duration</b> 0.00 s            |    |     |      |       |         |          |      |       |      |                       |
| <b>Clock Drift</b> 0 ms           |    |     |      |       |         |          |      |       |      |                       |
| <b>Freq Drift</b> 1 Hz (0.00 %)   |    |     |      |       |         |          |      |       |      |                       |

1 ストリームが戻りました

保存

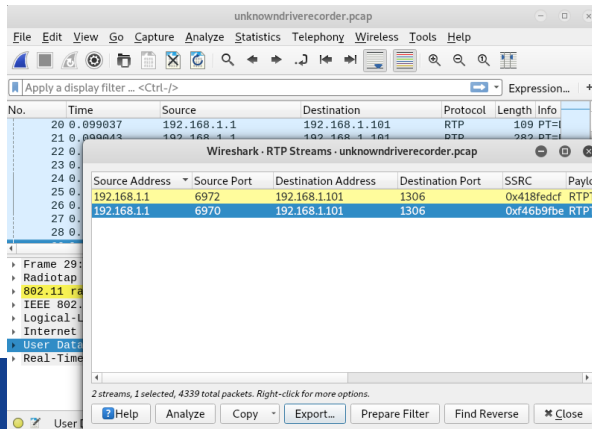
We can check Skew value, error and so on. If the codec is the common such as G.711, we just press "Play Stream" button to play audio, or "Save" as au file format, but in this case, we don't know the codec



# 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)



unknowndriverecorder.pcap

2019/02/05 3:38

Wireshark capture ...

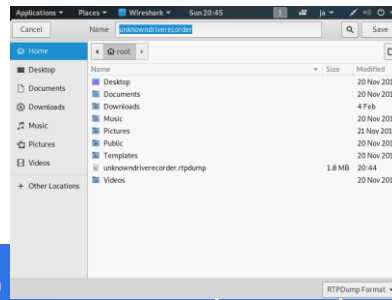
2,614 KB

unknowndriverecorder.rtpdump

2019/10/28 8:57

RTPDUMP ファイル

1,773 KB





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

The screenshot shows a network traffic capture window with the following content:

```
Adapter for loopback traffic capture (udp port 6970) からキャプチャ中
ファイル(E) 編集(E) 表示(V) 移動(G) キャプチャ(C) 分析(A) 統計(S) 電話(y) 無線(W) ツール(T) ヘルプ(H)
表示フィルタ ... <Ctrl+F> を適用
書式... + MYMAC

No. Time Source Destination Protocol Length Info
--- -
19... 30.26... localhost localhost RTP 782 PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=26065
20... 30.30... localhost localhost RTP 58 PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=26066
20... 30.30... localhost localhost RTP 920 PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=26067
20... 30.33... localhost localhost RTP 58 PT=DynamicRTP-Type-96, SSRC=0xF46B9FBE, Seq=26068

> Frame 1: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
> User Datagram Protocol, Src Port: 52371, Dst Port: 6970
v Real-Time Transport Protocol
 10.. = Version: RFC 1889 Version (2)
 ..0. = Padding: False
 ...0 = Extension: False
 0000 = Contributing source identifiers count: 0
 0... = Marker: False

0000 02 00 00 00 45 00 00 36 7a 92 00 00 80 11 00 00 E..6 z.....
0010 7f 00 00 01 7f 00 00 01 cc 93 1b 3a 00 22 05 13 :....."
0020 80 60 5b 2b 77 90 e7 8e f4 6b 9f be 06 01 09 00 [+w... .k.....

Frame (frame), 58 バイト
パケット数: 2002 · 表示: 2002 (100%)
プロファイル: Default
```



# 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 unknowndrivererecorder.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 ../unknowndrivererecorder.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 -c
copy -y ../unknowndrivererecorder.mkv
```

```
C:\Users\megumi\Desktop\ffmpeg>ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i vp8.sdp -c
copyts -c copy -y unknowndrivererecorder.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 -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 unknowndrivererecorder.rtpdump 127.0.0.1/6970
```

```
[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
```



# 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 unknowndrivererecorder.rtpdump 127.0.0.1/6970`  
`rtpplay -T -f unknowndrivererecorder.rtpdump 127.0.0.1/6970`
- **Receiving** `ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i h264.sdp -copyts -c copy -y ../unknowndrivererecorder.mkv`

```
[sdp @ 000001def605a340] RTP: missed 2 packets
```

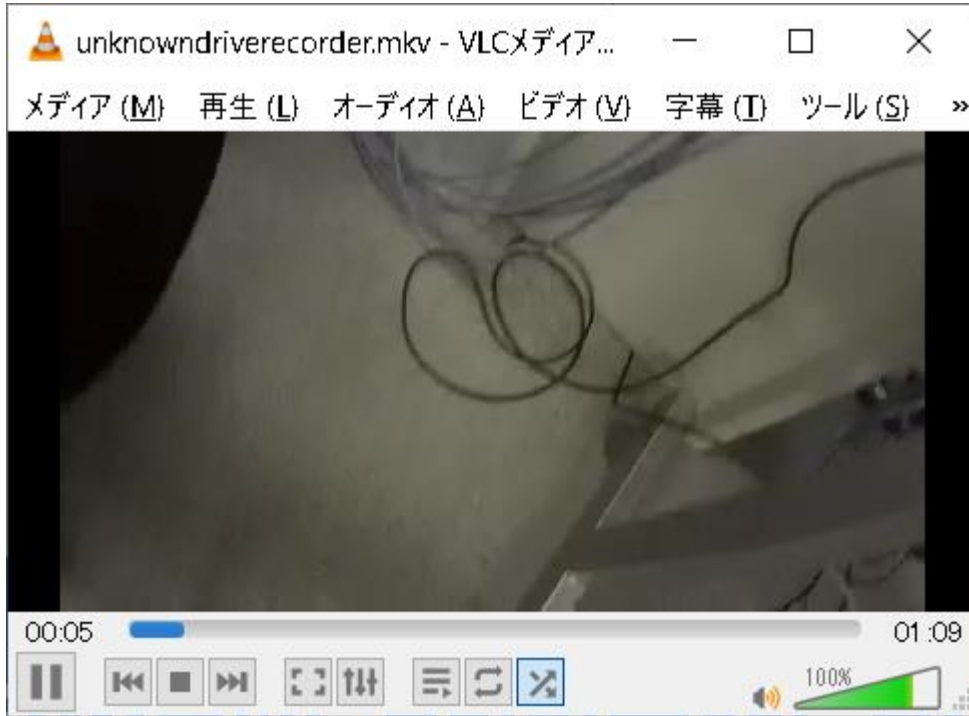
```
[NULL @ 000001def6069d00] missing picture in access unit with size 18
h264.sdp: Unknown error
```

- **Output contains some error but we can get mkv data**

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



# Finally we got the video !!



Try again to send and receive the RTP stream, ffmpeg outputs mkv file (unknowndrivererecorder.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



# 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>