




Extracting h264 video stream from pcap

PoC



In this brief presentation, I will demonstrate the process of extracting and building video stream from full packet capture pcap files without going too much into details. As I learned while working on this PoC, reconstructing h264 streams from large factory surveillance camera networks can be bit complex, especially with limited information about the network or the system overall, except about 1Tb of network data.



Case description

- Carving and reconstructing surveillance camera video from network data.
- 1Tb of 2Gb pcap files (about 1 month worth of continuous network data) from large-scale factory network.
- Prior information about the network is limited (addresses and locations of cameras and other hosts/devices, software used, protocols in use, encryption etc.), including the point where the data was recorded.

Finding the right stream

- Open pcap in Wireshark for inspection
- 2Gb pcap file is quite heavy for Wireshark to process
 - Using filters or inspectin streams takes a long time for Wireshark to process
 - If new filter added, Wireshark will process the whole 2Gb file, instead of previously filtered packets
- Inspecting packet capture in NetworkMiner first, provided some valuable information to start from

- Html file found by NetworMiner and further investigation indicated that several ip addresses are sending video related data to one main streaming server (192.168.112.175)
- `192.168.100.54/axis-cgi/view/param.cgi?camera=1&action=list&group=Audio,AudioSource.A0`
`192.168.100.54/axis-cgi/audio/receive.cgi?`
- Investigated data stream should be filtered and extracted as separate pcap file for more efficient processing (ip.addr == <ip> && ip.addr == <ip>) -> (Export Specified Packets)

- Traffic from 192.168.100.47 to 192.168.112.157 contained a large amount of data transferred, including actual payload, RTSP control sequences (Real Time Streaming Protocol), SDP information (Session Description Protocol) and information about the stream server.
- - ▼ Real Time Streaming Protocol
 - ▼ Request: DESCRIBE rtsp://192.168.100.47/axis-media/media.amp?videocodec=h264 RTSP/1.0\r\nMethod: DESCRIBEURL: rtsp://192.168.100.47/axis-media/media.amp?videocodec=h264CSeq: 201\r\nUser-Agent: AXIS RTSP Client (AXIS RTP Source Filter)\r\nAccept: application/sdp\r\n\r\n
 - Server: GStreamer RTSP server\r\n

Extract H264 data from pcap

- Use Wireshark to extract H264 stream from pcap
 - Decode UDP traffic as RTP
 - Set Wireshark H264 dynamic payload types to 96 (as in RTSP/SDP packet)

```
▼ Media Description, name and address (m): video 0 RTP/AVP 96
  Media Type: video
  Media Port: 0
  Media Protocol: RTP/AVP
  Media Format: DynamicRTP-Type-96
```

- Telephony → RTP → RTP Streams

- 2 RTP streams found:

Source Address	Source Port	Destination Address	Destination Port	SSRC	Payload	Packets	Lost	Max Delta (ms)	Max Jitter	Mean Jitter
192.168.100.47	50000	192.168.112.175	10018	0xcce37b93	h264	2852	0 (0.0%)	74.491	3.121	0.637
192.168.100.47	50000	192.168.112.175	27376	0x7d181813	h264	112099	0 (0.0%)	98.843	5.185	0.487


- Prepare filter and export stream as separate pcap file
- Find SDP file negotiated to exported stream
 - First packet of H264 stream (Start: IDR-Slice, No. 72851)
 - Delete stream separating filters and filter out tcp traffic
 - Locate and export SDP file before packet No. 72851 (found in RTSP DESCRIBE reply to server, No. 72841)

```
72841 2224.203945 192.168.100.47 192.168.112.175 RTSP/SDP 885 Reply: RTSP/1.0 200 OK
```




Playback

- For this PoC I tried few different approaches to extract and play video streams. Programs like Videosnarf was able to extract and make sample H264 pcap playable, but with larger amounts of traffic and more complex setups, the result was unreliable. For example misinterpreted NAL unit start codes in http requests.
- For better results, processed pcap file should be carefully filtered before feeding it to Videosnarf
- Videosnarf is capable of extracting the stream, but not in playable format in this case

- 
- Wireshark can be used to extract streams by filtering out streams or by using Lua scripts to find and extract h264 streams.
 - Tested Lua script found and extracted stream, but interpreted all traffic containing h264 as one stream.
 - In this case replaying the network traffic back to listening videoplayer was most straight forward and steady way to go
 - Highly recommend to read RTP Payload Format for H.264 Video (RFC 6184)

Replaying stream

- Replaying video stream to player server
 - Sending and receiving virtual machines (send: Centos, Rec: Debian)
 - tcprewrite to rewrite addresses
 - Any text editor to modify/create SDP file to ffmpeg
 - Tcpreplay to replay traffic to server
 - ffmpeg to receive and play stream

Sending machine

- Centos 7
- Use shell script or tcprewrite from command line to rewrite addresses to previously extracted h264 pcap file (part from script)

```
tcprewrite \  
  --fixcsum \  
  --srcipmap=${ORIG_SRC_IP}/32:${REPLAY_SRC_IP}/32 \  
  --enet-smac=${REPLAY_SRC_MAC} \  
  --dstipmap=${ORIG_DST_IP}/32:${REPLAY_DST_IP}/32 \  
  --enet-dmac=${REPLAY_DST_MAC} \  
  --infile="${IN_PCAP_FILE}" \  
  --outfile="${OUT_PCAP_FILE}"
```

- Use tcpreplay to replay traffic to player machine *tcpreplay -
intf1=<NIC> <input pcap>*

Receiving machine

- Debian
- Modify previously extracted SDP file in receiving machine. Set sender address to `c=IN IP4 <ip>` & port to `m=video <port> RTP/AVP 96` and feed as input to `ffplay`: `ffplay -protocol_whitelist file,udp,rtp -i <SDP file>` (`-protocol-whitelist` only if errors)

```
m=video 27376 RTP/AVP 96
c=IN IP4 192.168.198.134
b=AS:50000
a=rtpmap:96 H264/90000
a=fmtp:96 packetization-mode=1;profile-level-id=4d0029;sprop-parameter
-sets=Z00AKeKQGQd/EYC3AQEBpB4kRUA=,a048gA==
a=framerate:25.000000
a=transform:1.000000,0.000000,0.000000;0.000000,1.000000,0.000000;0.00
0000,0.000000,1.000000
```

- When ffplay launches with SDP file, it waits and listens to incoming streams. Launch tcpreplay from sending machine and stream should start playing in ffplay. (picture blurred)

