204 – Find & Insert Resolution

Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc_luckyvicky@googlegroups.com

Instructions

Description DVR manufacturers apply security features to the stored video data. Video data is characterized by large data size and is already encoded with a specific codec, so it is easy to apply security by removing decoding parameters. As video data without decoding parameters cannot be played back, manufacturers are choosing to embed decoding parameters into their proprietary viewer programs. However, if directory entries at the filesystem level are deleted, or if file-level organization is not possible, the dedicated viewer will not play them. A forensic investigator analyzes a specific DLL file of a dedicated viewer program to check some of the decoding parameters and finds that the resolution information in the manual is 1920*1080, but the resolution of the decoding parameters is different.

Target	Hash (MD5)
Problem_Frame_Data.264	821d3a9acca2a5b69352387fadf23fbf
Extracted Header.hed	9a95ce76c6b02ad2d98866abbc248439

Questions

1. Submit the resolution information for the H.264 encoding parameters given in the Extracted header file. (50 points)
2. Decode the video and submit the artist and name of the recorded masterpiece. (150 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	HashTab	Publisher:	Implbits Software
Version:	6.0.0		
URL:	https://implbits.com		

Name:	HxD	Publisher:	Maël Hörz
Version:	2.5.0.0		
URL:	https://www.mh-nexus.de		

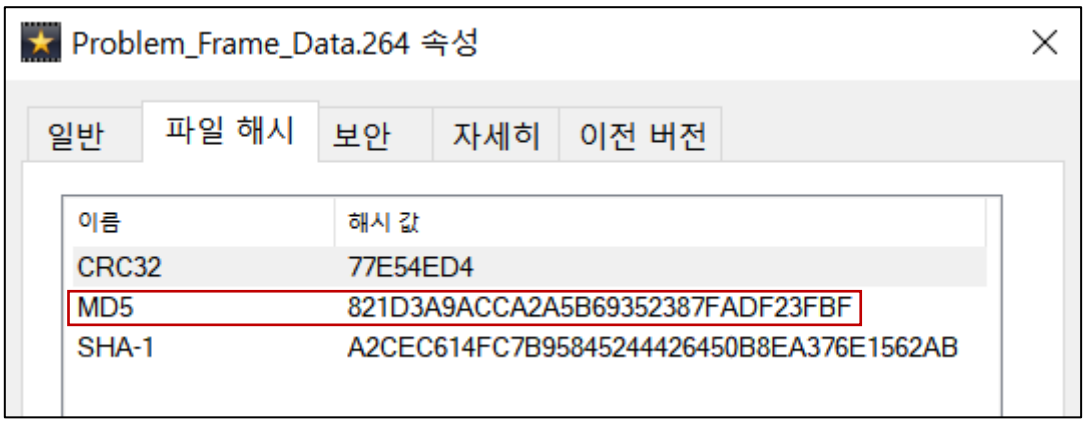
Name:	Untrunc-gui	Publisher:	anthwlock
Version:	v367-13cafed		
URL:	https://github.com/anthwlock/untrunc		

Name:	kmplayer	Publisher:	PandoraTV
Version:	4.2.2.68		
URL:	https://www.kmplayer.com/kr/home		

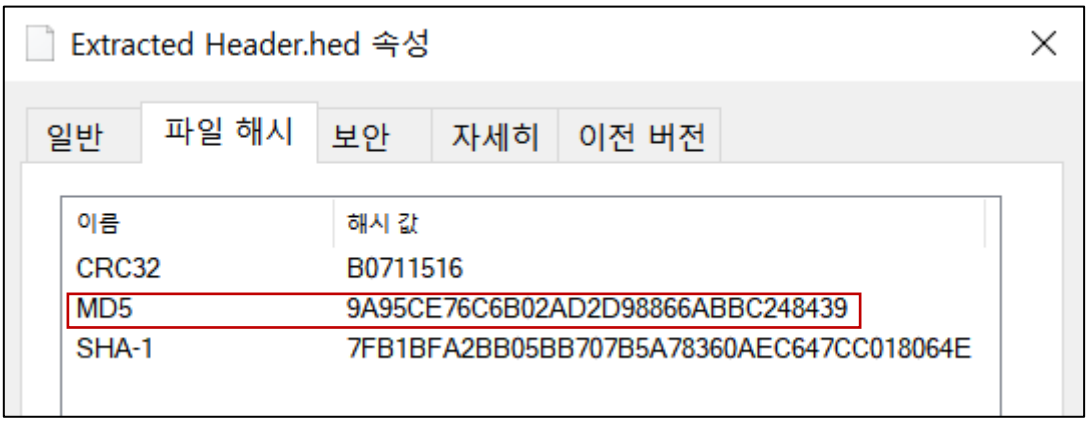
Name:	H264 bitstream viewer	Publisher:	mradionov
Version:	-		
URL:	https://mradionov.github.io/h264-bitstream-viewer		

Name:	My Mp4box GUI	Publisher:	Matt bodin
Version:	0.6.0.6		
URL:	-		

Step-by-step methodology:



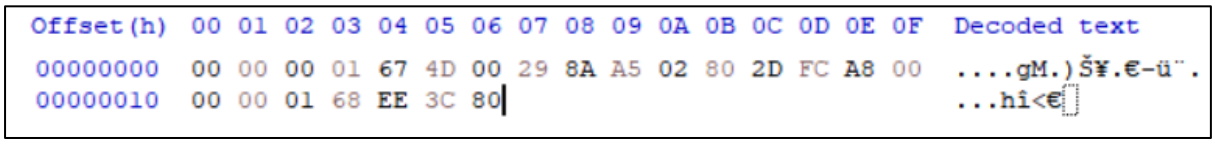
[그림 1] Problem_Frame_Data.264 md5 hash 확인



[그림 2] Extracted Header.hed md5 hash 확인

주어진 target 파일들의 md5 hash 값이 일치함을 확인하였습니다.

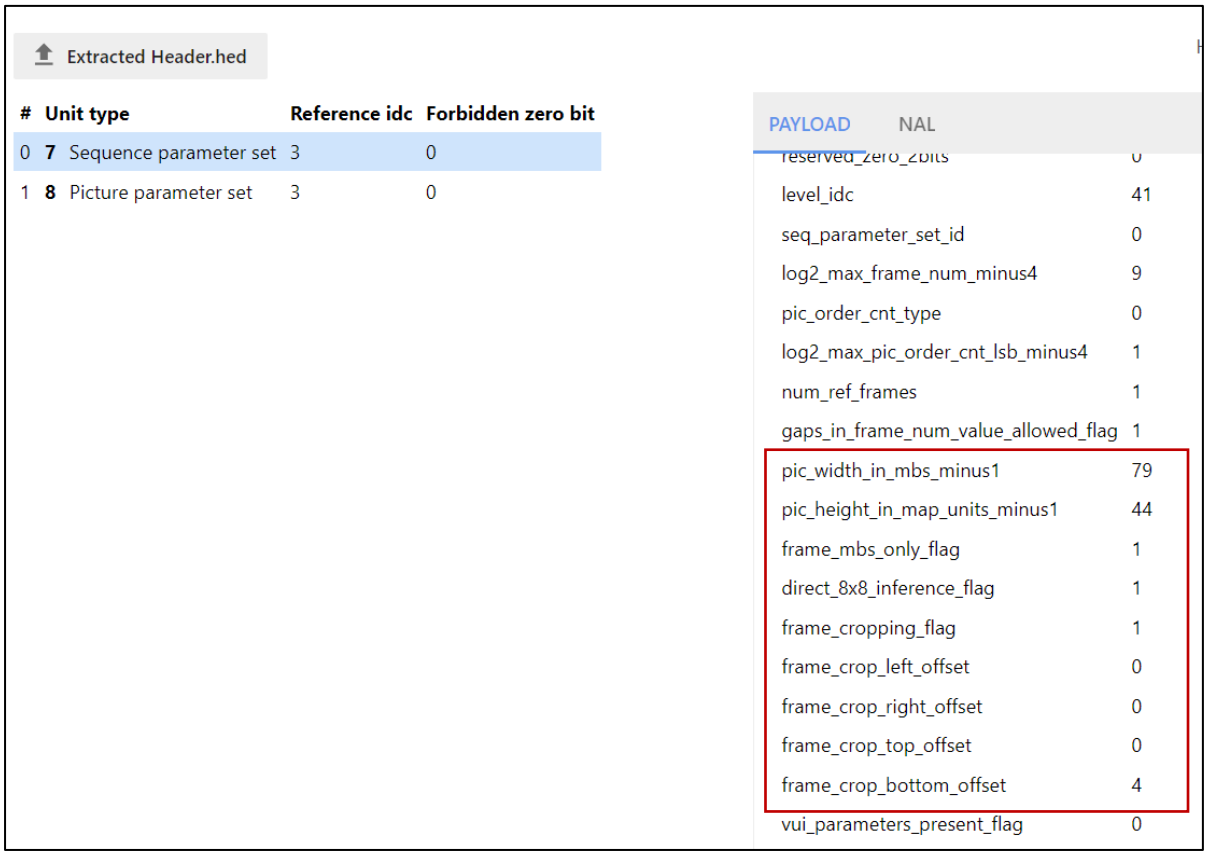
1. Submit the resolution information for the H.264 encoding parameters given in the Extracted header file. (50 points)



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	00	00	00	01	67	4D	00	29	8A	A5	02	80	2D	FC	A8	00	....gM.)Š¥.€-ü¨.
00000010	00	00	01	68	EE	3C	80										...hî<€

[그림 3] Extracted Header.hed 데이터

주어진 Extracted Header.hed 파일을 살펴보면, NAL Unit(00 00 00 01)과 0x04 offset의 0x67값에서 7을 통한 SPS(Sequence Parameter Set), 그리고 0x13 offset에 위치한 0x68값에서 8을 통한 PPS(Picture Parameter Set)을 확인할 수 있습니다.



[그림 4] 주어진 h264 sps와 pps stream 파싱

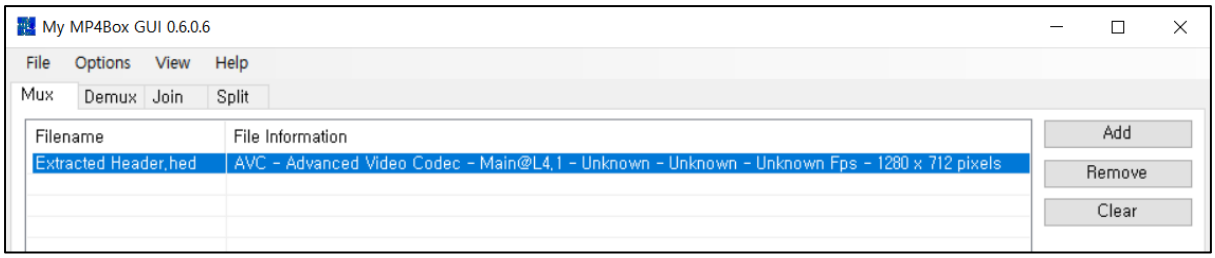
이를 통해 h264 bitstream viewer 사이트에서 해당 encoding parameter 내용을 파악했습니다. Resolution 정보를 알기 위해서는 위 빨간색 박스에 해당하는 sps 스트림의 일부분을 파악해야 했습니다.

Width와 height는 ITU-T Rec. H.264 Advanced video coding [1]공식 문서를 참조해볼 때, 다음과 같이 계산할 수 있습니다.

width = (pic_width_in_mbs_minus1 + 1) * 16 - frame_crop_left_offset * 2 - frame_crop_right_offset * 2

height = (2 - frame_mbs_only_flag) * (pic_height_in_map_units_minus1 + 1) * 16 - frame_crop_top_offset * 2 - frame_crop_bottom_offset * 2

따라서, width의 경우 (79 + 1) * 16 – 0 * 2 – 0 * 2로 1280이었고, height의 경우에는 (2 – 1) * (44 + 1) * 16 – 0 * 2 – 4 * 2 로 712라는 값을 계산할 수 있었습니다.



[그림 5] My Mp4Box GUI 도구 내 File information

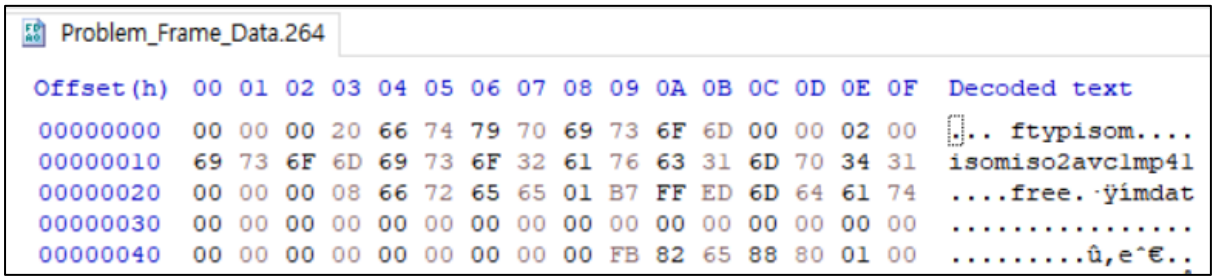
또한, My Mp4Box GUI도구에서도 해당 hed 파일 내 sps와 pps 스트림을 통해 1280x712 pixels로 계산하는 것을 알 수 있습니다.

따라서, resolution information은 **1280x712**로 확인됩니다.

답 : 1280x712

[1] http://wikil.lwwhome.cn:28080/wp-content/uploads/2018/08/T-REC-H.264-201704%E8%8B%B1%E6%96%87.pdf

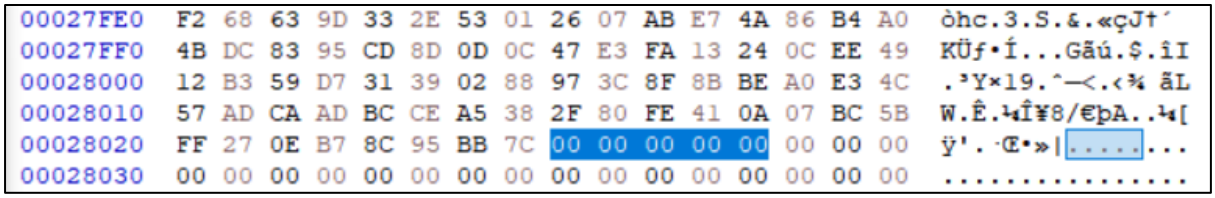
2. Decode the video and submit the artist and name of the recorded masterpiece. (150 points)


Problem_Frame_Data.264

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	00	00	00	20	66	74	79	70	69	73	6F	6D	00	00	02	00	... ftypisom....
00000010	69	73	6F	6D	69	73	6F	32	61	76	63	31	6D	70	34	31	isomiso2avc1mp41
00000020	00	00	00	08	66	72	65	65	01	B7	FF	ED	6D	64	61	74	....free.·ÿímdat
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000040	00	00	00	00	00	00	00	00	00	FB	82	65	88	80	01	00	.........û‚e^€..

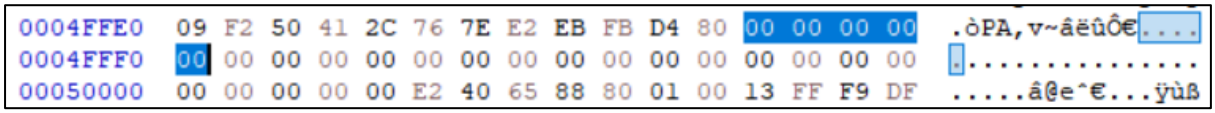
[그림 6] Problem_Frame_Data.264 데이터 일부 - 1

주어진 Problem_Frame_Data.264 파일을 HxD로 열었을 때, mp4에서 사용하는 ftyp, free, mdat 등의 atom을 확인할 수 있었습니다.



00027FE0	F2	68	63	9D	33	2E	53	01	26	07	AB	E7	4A	86	B4	A0	òhc.3.S.&.«çJ†´
00027FF0	4B	DC	83	95	CD	8D	0D	0C	47	E3	FA	13	24	0C	EE	49	KÜƒ•Í...Gãú.$.îI
00028000	12	B3	59	D7	31	39	02	88	97	3C	8F	8B	BE	A0	E3	4C	.³Y×19.ˆ—<.‹¾ ãL
00028010	57	AD	CA	AD	BC	CE	A5	38	2F	80	FE	41	0A	07	BC	5B	W.Ê.¼Î¥8/€þA..¼[
00028020	FF	27	0E	B7	8C	95	BB	7C	00	00	00	00	00	00	00	00	ÿ'.·Œ•»|.....
00028030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

[그림 7] Problem_Frame_Data.264 데이터 일부 - 2

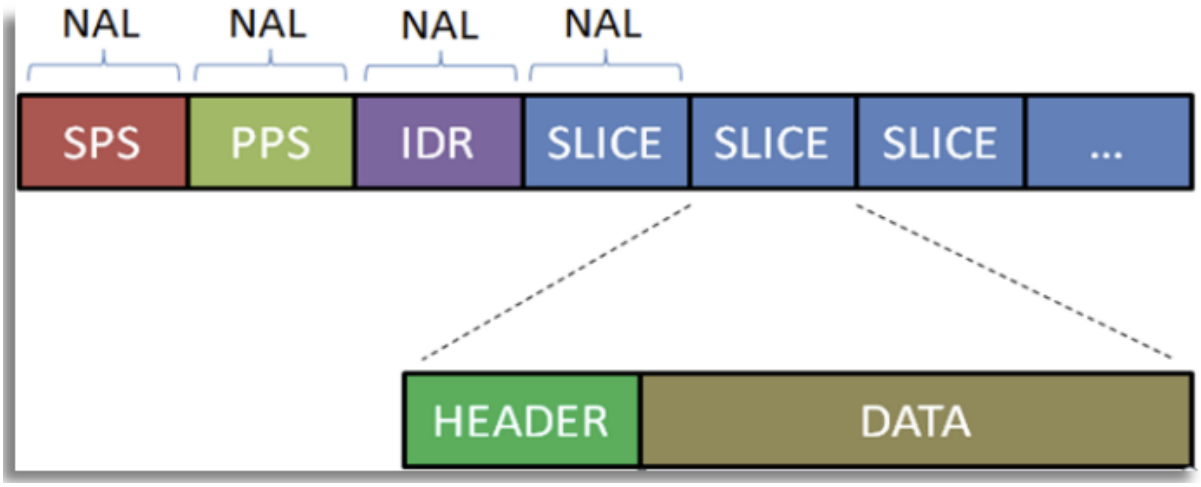


0004FFE0	09	F2	50	41	2C	76	7E	E2	EB	FB	D4	80	00	00	00	00	.òPA,v~âëûÔ€....
0004FFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00050000	00	00	00	00	00	E2	40	65	88	80	01	00	13	FF	F9	DF	.....â@e^€...ÿùß

[그림 8] Problem_Frame_Data.264 데이터 일부 - 3

그리고 데이터 중간 중간에 0x19 크기만큼 0x00으로 값이 잘려 있는 부분들을 확인할 수 있었습니다.

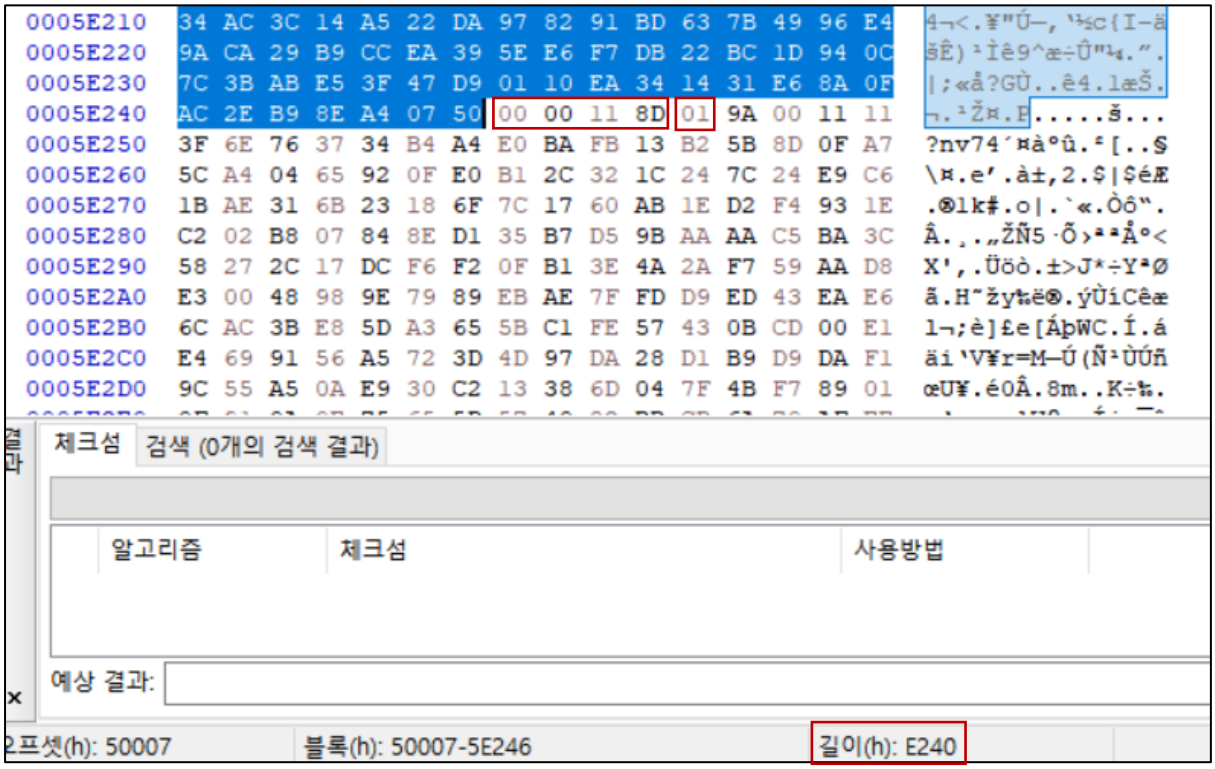
H264 stream 구조 상 SPS, PPS, IDR, non-IDR...이 온다는 것과 지문에서 주어진 디코딩 파라미터의 누락을 생각해볼 때, Extracted Header.hed에서 주어진 0x17의 SPS와 PPS 데이터가 Problem_Frame_Data.264 파일에 누락되어 있다는 생각을 할 수 있었습니다. 그리고 0x17 이후 0x02만큼의 0x00은 뒤에 이미 mdat atom에 저장되면서 length-prefix 형식으로 NALU의 크기를 의미하는 값을 넣어 스트리밍을 수행하기 위해 변환되어 있는 값이며, 0x65가 존재하는 점을 통해 IDR의 일부인 것도 파악할 수 있었습니다.



[그림 9] types of NAL

```
00050000  00 00 00 00 00 E2 40 65 88 80 01 00 13 FF F9 DF  ...  ..â@e^€...ÿùß
00050010  03 CF 10 B2 4D E1 4F 6D 92 32 6C CB 02 F7 4B F6  .Ï.²MáOm'2lË.÷Kö
```

[그림 10] IDR frame의 length 예시

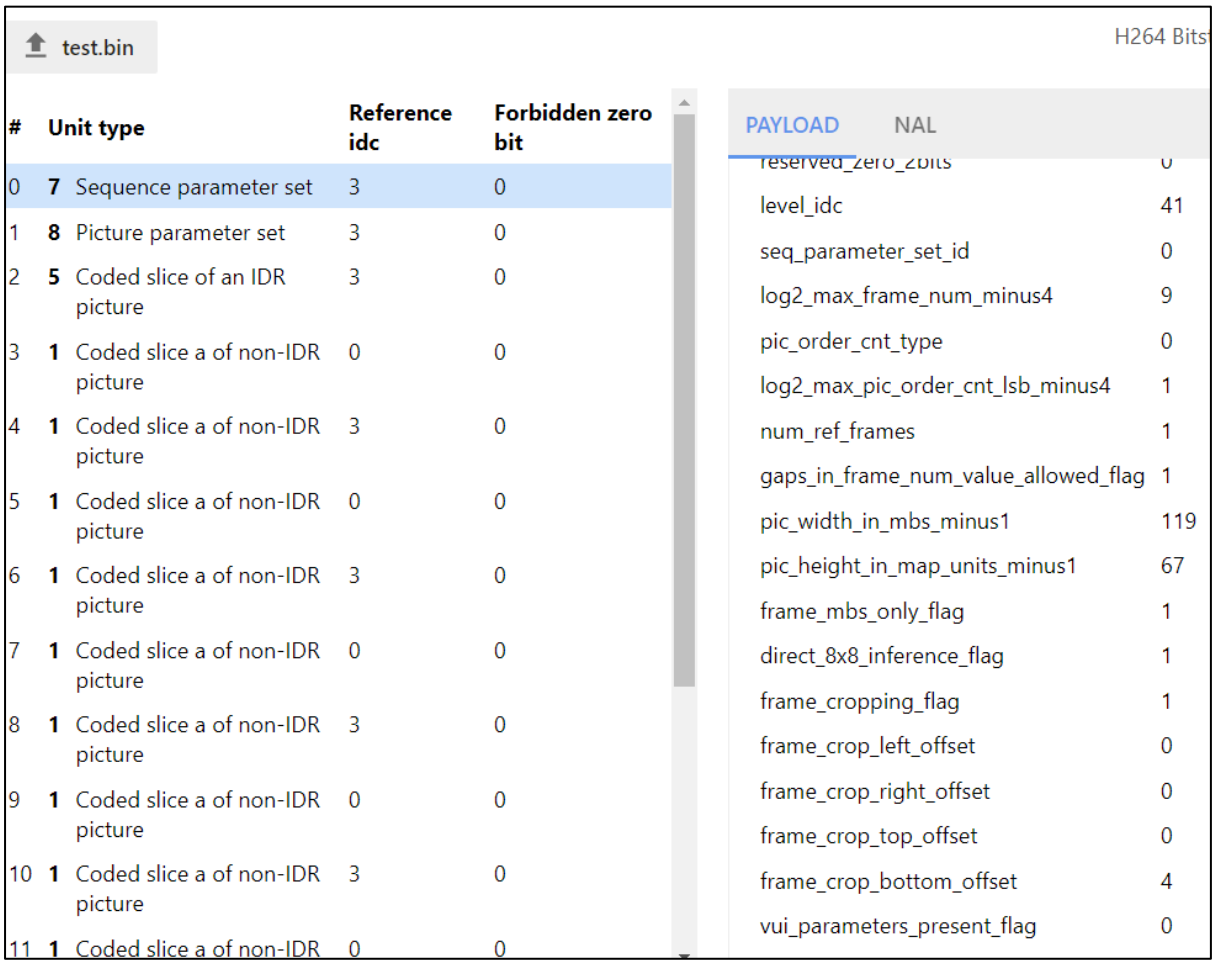


[그림 11] IDR frame의 length에 해당하는 데이터 및 다음 non-IDR frame length

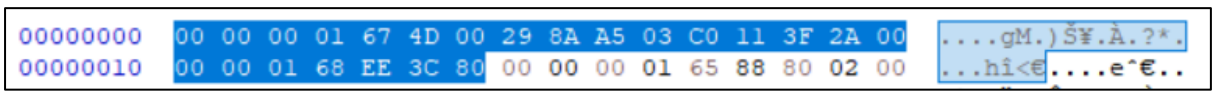
위 그림 9에 따른 구조를 통해 그림 10에서 0x65 & 0x1F = 5인 IDR(I-frame)의 크기가 length-prefix를 통해 0xE240만큼의 크기임을 알았고, 따라가보면 다음 non-IDR frame(0x01 & 0x1F) length-prefix(4byte)이전까지 데이터가 오는 것을 알 수 있습니다.

위 분석들을 토대로, 주어진 Problem_Frame_Data.264의 0x17만큼 빈 구간을 sps, pps 데이터로 채우고자 하였습니다.

문제 지문에서 주어진 정보를 통해서 수 차례의 테스트 과정을 통해 기존의 sps 값으로는 영상 재생이 되지 않음을 확인하였고, 대신 1920x1080 해상도를 가진 sps data를 활용하고자 하였습니다.



[그림 12] 해상도 1920x1080인 stream활용 - 1



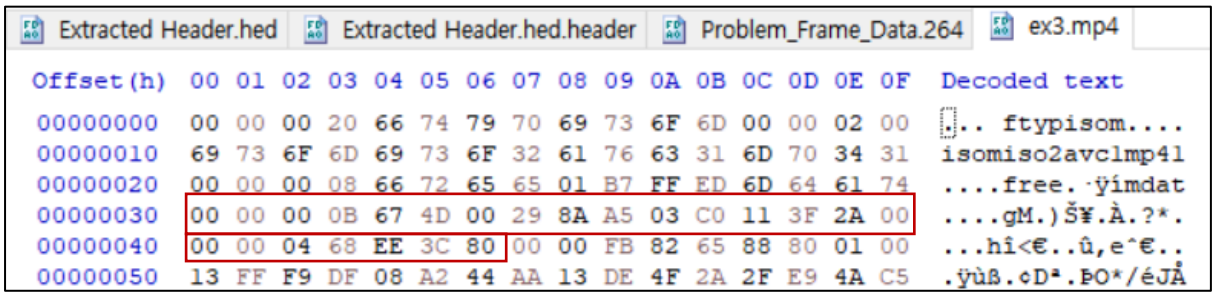
[그림 13] 해상도 1920x1080인 stream활용 - 2

그림 12와 그림 13을 통해 기존에 Extracted Header.hed와 pps값은 같지만, sps에서 해상도 값만 다른 251문제에서의 주어진 타깃 파일(Problem.bin) 내 sps값을 활용하였습니다.

```python
def replace_pattern_in_file(file_path):
    original_pattern = bytes([0x00] * 23)

    new_pattern = bytes([0x00, 0x00, 0x00, 0x0B, 0x67, 0x4D, 0x00, 0x29, 0x8A, 0xA5,
                         0x03, 0xC0, 0x11, 0x3F, 0x2A, 0x00, 0x00, 0x00, 0x04, 0x68, 0xEE, 0x3C, 0x80])

    with open(file_path, 'rb') as file:
        data = file.read()

    modified_data = data.replace(original_pattern, new_pattern)

    with open(file_path, 'wb') as file:
        file.write(modified_data)

file_path = 'Problem_Frame_Data.264'
replace_pattern_in_file(file_path)
```

[그림 14] Problem_Frame_Data.264에 sps, pps 데이터 주입 코드

위 코드를 통해 비어져있는 부분에 sps, pps 데이터를 삽입하였습니다. 기존에 NALU header 0x00 0x00 0x00 0x01에서 길이에 맞도록 sps는 0x0B, pps는 0x04로 수정해주었습니다.



[그림 15] 데이터 삽입 확인

[그림 16] mdat 이후 남는 데이터

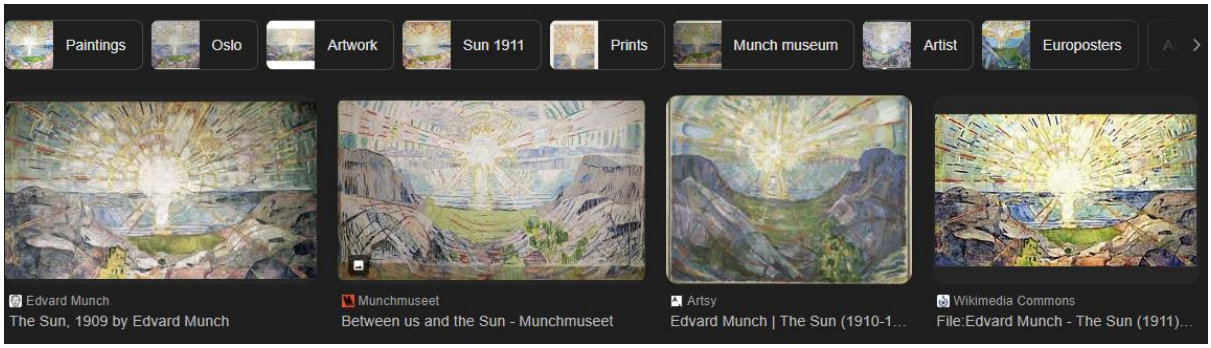
이후 mdat크기인 0x1B7FFED 이후 4바이트 남는 부분은 mp4 구조 특성상 moov atom에 대한 size값으로 추측되지만 확실하게 판단할 수 없어, moov atom을 재생성해볼 수 있는 untrunc 도구를 활용하였습니다.

[그림 17] untrunc-gui 도구를 통해 mp4 구조 repair

1920x1080 해상도를 가지는 샘플 mp4를 https://file-examples.com/index.php/sample-video-files/sample-mp4-files/ 사이트에서 다운로드하여, sps와 pps를 채워주었지만 moov atom이 없어 재생이 정상적으로 불가능한 truncated 파일인 ex3.mp4(이전 Problem_Frame_Data.264)에 moov atom을 복구해주었습니다.

[그림 18] kmplayer를 통해 확인한 repair된 mp4파일(ex3.mp4_fixed.mp4)

그 후, kmplayer를 통해 복구된 영상을 재생시켜보았을 때 명작이 나오는 것을 확인할 수 있었습니다.



[그림 19] 구글 이미지 검색결과

이미지 검색결과 해당 masterpiece는 **Edvard Munch – The Sun**로 확인되었습니다.

답 : Edvard Munch – The Sun