204 - The Order of Time

Team Information

Team Name : HSPACE

Team Member : Jinung Lee, Beomjun Park, DoHyeon Kim, Soyoung Cho

Email Address : hspacedigitalforensicslab@gmail.com

Teams must:

- Provide a detailed, step-by-step description of their problem-solving approach to ensure reproducibility by another examiner.
- List all tools used to arrive at their conclusions.

Tools used:

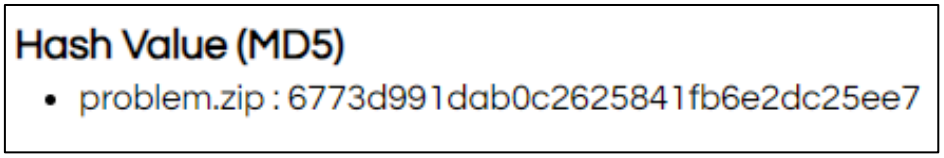
Name:	ffmpeg	Publisher:	FFmpeg
Version:	7.1.1		
URL:	https://ffmpeg.org/		

Name:	Python	Publisher:	Python Software Foundation
Version:	3.12.6150.0		
URL:	https://www.python.org/		

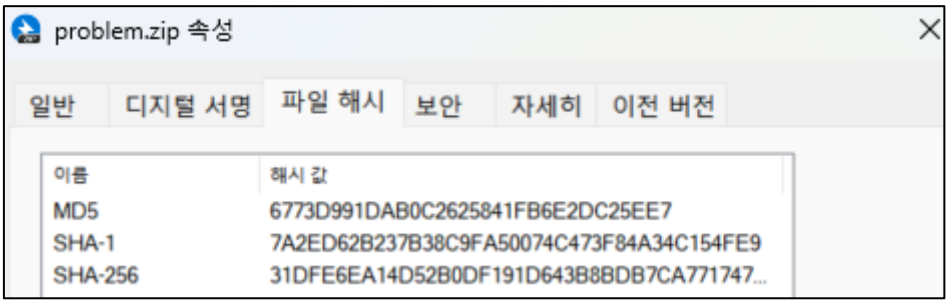
Name:	곰플레이어	Publisher:	GOM & Company
Version:	2.3.110.5380		
URL:	https://www.gomlab.com/		

Step-by-step methodology:

문제 풀이에 앞서, dfchallenge.org에 공지된 문제 해시와 다운로드 받은 문제 해시를 비교함으로써 분석 대상이 동일한 파일임을 증명한다.



[그림 1] dfchallenge.org에 공지된 문제 해시(MD5) 값.



[그림 2] HashTab을 통해 확인한 문제 해시(MD5) 값.

1. 손상된 DVR 백업 데이터를 전용뷰어 프로그램으로 재생 하였을때, 실제 영상 데이터는 있지만 전용뷰어 프로그램 에서는 재생되지 않는 시간대의 명언을 제출 하시요. (200 points)

문제 파일인 problem.zip 을 압축해제 하면, 다음과 같은 구성의 디렉토리를 확인할 수 있다.

파일 경로	설명
₩BackupPlayer.exe	icctv 영상 플레이어
₩Ch03₩Ch03_CH 03.rmi	인덱스 정보가 담긴 파일
₩Ch03₩Ch03_CH 03.rms	영상 정보가 담긴 파일

[표 1] problem 디렉토리 구조 정리.

손상된 구간을 확인하기 위하여 일단 인덱스 정보가 담긴, Ch03_CH 03.rmi 파일을 분석하였다. 분석 결과는 [표 2]와 같이 파일 헤더와 레코드 배열로 나뉜 것을 알 수 있다.

구성 요소	크기	설명
파일 헤더	가변	RFHD 시그니처 + JSON 메타데이터
레코드 배열	24바이트 x N개	각 영상 구간의 정보

[표 2] Ch03_CH 03.rmi파일의 구조 정리.

본 풀이에서 중요한 요소는 레코드 배열이다. 이를 분석하여 손상된 시간을 파악할 수 있었으며, 해당 파일 구조를 분석한다면 [표 3]와 같다.

오프셋	크기	필드명	설명
0x00	4	Flags	레코드 상태 플레그
0x04	4	Size	영상 데이터 크기
0x08	4	Start_Unix	시작 시간 (Unix timestamp)
0x0C	4	Start_High	시작 시간 상위 비트
0x10	4	End_Unix	종료 시간 (Unix timestamp)
0x14	4	End_Unix	종료 시간 상위 비트

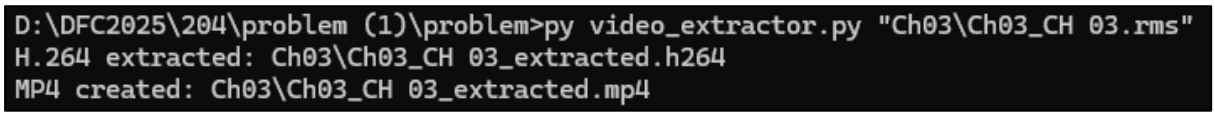
[표 3] 레코드 배열 구조 정리.

해당 구조를 바탕으로 작성된 python script인 analyze_rmi.py[별첨 1]을 통하여 총 레코드는 91개임을 알 수 있으며, 이 중 손상된 레코드는 6~12임을 알 수 있다. 또한 해당 시간을 분석하면, 2025-07-19 21:12:21 ~ 2025-07-19 21:12:35까지 약 14초의 시간이 손상되었음을 알 수 있다.

레코드 번호	시작 시간	종료 시간	손상 여부
1	2025-07-19 21:12:12	2025-07-19 21:12:14	정상
2	2025-07-19 21:12:14	2025-07-19 21:12:16	정상
3	2025-07-19 21:12:16	2025-07-19 21:12:18	정상
4	2025-07-19 21:12:18	2025-07-19 21:12:20	정상
5	2025-07-19 21:12:20	2025-07-19 21:12:21	정상
6			손상
7			손상
8			손상
9			손상
10			손상
11			손상
12			손상
13	2025-07-19 21:12:35	2025-07-19 21:12:37	정상
14	2025-07-19 21:12:37	2025-07-19 21:12:38	정상
15	2025-07-19 21:12:38	2025-07-19 21:12:40	정상
90	2025-07-19 21:15:12	2025-07-19 21:15:14	정상
91	2025-07-19 21:15:14	2025-07-19 21:15:16	정상

[표 4] Ch03_CH 03.rmi 분석 결과.

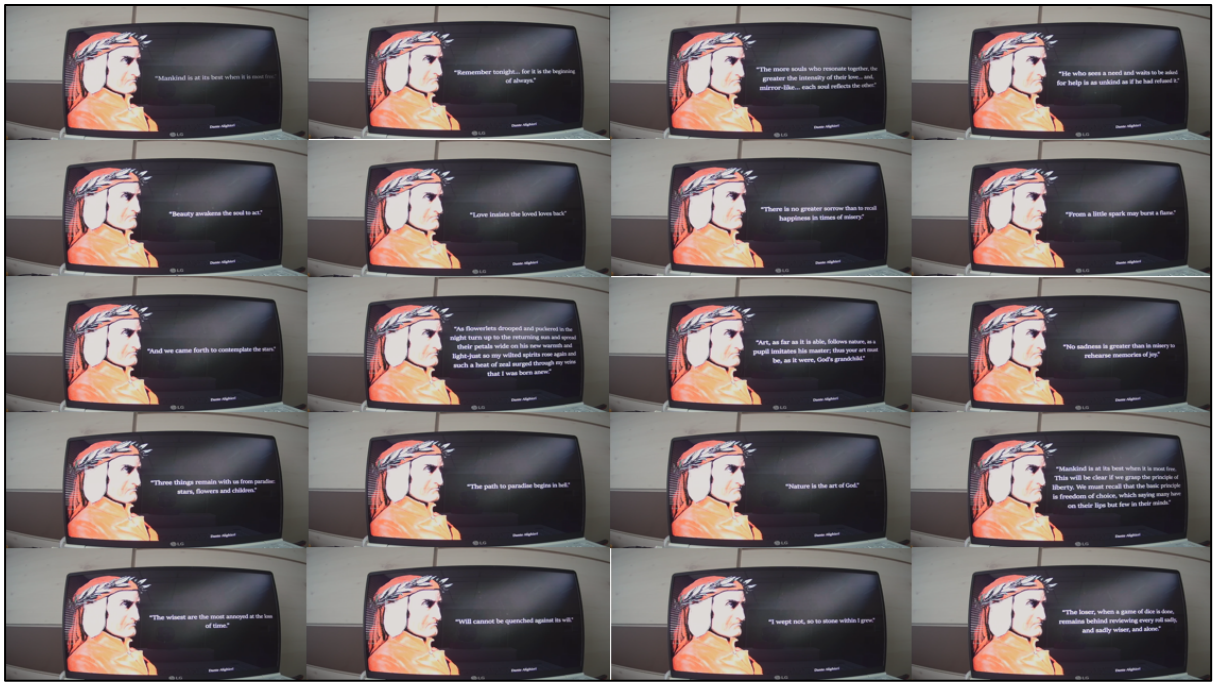
Ch03_CH 03.rms의 경우, 동영상에 대한 영상 정보가 담긴 파일이다. 해당 파일을 통하여 전체 영상파일을 추출할 수 있었다. 이의 경우 FFmpeg를 이용하여 정상적으로 추출할 수 있었으며, video_extractor.py[별첨 2]를 통해, 추출할 수 있다.



[그림 3] video_extractro.py를 통한 영상 추출.

Video_extractor.py를 통하여 추출한 영상 파일과, BackupPlayer.exe를 통해 본 영상을 비교한 결과는 [그림 4]와 [그림 5]와 같다. 분석 결과 3번 째 명언이 BackupPlayer에서는 확인할 수 없지만, 추출한 영상에서는 정상적으로 확인할 수 있었다.

[그림 4] BackupPlayer을 통해 확인한 영상의 명언 장면 모음.



[그림 5] 추출한 영상을 통해 확인한 영상의 명언 장면 모음

재생되지 않는 시간	명언
2025-07-19 21:12:21 ~ 2025-07-19 21:12:35	The more souls Who resonate together, the greater the intensity of their love... and, mirror-like... each soul reflects the other.

[표 5] 재생되지 않는 시간과 명언 정리.

```python
#!/usr/bin/env python3
import struct
import json
import datetime
import sys

def to_datetime(timestamp):
    """Unix timestamp를 날짜/시간으로 변환"""
    if timestamp == 0:
        return None
    try:
        dt = datetime.datetime.fromtimestamp(timestamp)
        return dt.strftime('%Y-%m-%d %H:%M:%S')
    except:
        return None

def parse_rmi(file_path):
    """RMI 파일 파싱"""
    with open(file_path, 'rb') as f:
        data = f.read()

    # 헤더 파싱
    magic = data[:4]
    if magic != b'RFHD':
        print("올바른 RMI 파일이 아닙니다.")
        return

    header_size = struct.unpack('<I', data[4:8])[0]

    # JSON 메타데이터 추출
    json_start = 8
    json_end = json_start + header_size
    json_data = data[json_start:json_end].decode('utf-8', errors='ignore')

    channel_name = "Unknown"
    if '{' in json_data and '}' in json_data:
        try:
            start_idx = json_data.find('{')
            end_idx = json_data.rfind('}') + 1
            json_str = json_data[start_idx:end_idx]
            header_info = json.loads(json_str)
            channel_name = header_info.get('cam_title', 'Unknown')
        except:
            pass

    print(f"채널: {channel_name}")
    print("-" * 50)
```

```
    # 레코드 파싱
    offset = 8 + header_size
    record_size = 24
    records = []

    while offset + record_size <= len(data):
        try:
            record_data = data[offset:offset + record_size]

            # 구조: flags(4) + size(4) + start_unix(4) + start_high(4) +
end_unix(4) + end_high(4)
            flags = struct.unpack('<I', record_data[0:4])[0]
            size = struct.unpack('<I', record_data[4:8])[0]
            start_unix = struct.unpack('<I', record_data[8:12])[0]
            end_unix = struct.unpack('<I', record_data[16:20])[0]

            start_time = start_unix if start_unix != 0 else 0
            end_time = end_unix if end_unix != 0 else 0

            # 손상 여부 확인
            is_damaged = (start_time == 0 and end_time == 0)

            records.append({
                'start_time': start_time,
                'end_time': end_time,
                'is_damaged': is_damaged,
                'flags': flags,
                'size': size
            })

            offset += record_size

        except Exception as e:
            break

    # 레코드 출력
    print(f"총 {len(records)}개 레코드:")
    print()

    for i, record in enumerate(records, 1):
        start_str = to_datetime(record['start_time']) or "손상됨"
        end_str = to_datetime(record['end_time']) or "손상됨"
        status = "손상" if record['is_damaged'] else "정상"

        print(f"레코드 {i:2d}: {start_str} ~ {end_str} [{status}]")
```

```
    # 손상된 구간 분석
    print("\n" + "=" * 50)
    print("손상된 구간 분석")
    print("=" * 50)

    damaged_periods = []
    i = 0

    while i < len(records):
        if records[i]['is_damaged']:
            # 손상 구간 시작점
            start_time = None
            if i > 0 and not records[i-1]['is_damaged']:
                start_time = to_datetime(records[i-1]['end_time'])

            # 연속된 손상 레코드 개수 세기
            count = 0
            j = i
            while j < len(records) and records[j]['is_damaged']:
                count += 1
                j += 1

            # 손상 구간 끝점
            end_time = None
            if j < len(records) and not records[j]['is_damaged']:
                end_time = to_datetime(records[j]['start_time'])

            start_display = start_time or "Unknown"
            end_display = end_time or "Unknown"

            if count > 1:
                period_info = f"{start_display} ~ {end_display} (연속
{count}개 레코드)"
            else:
                period_info = f"{start_display} ~ {end_display}"

            damaged_periods.append(period_info)
            i = j
        else:
            i += 1

    if damaged_periods:
        print(f"\n손상된 구간 ({len(damaged_periods)}개):")
        for i, period in enumerate(damaged_periods, 1):
            print(f"  {i}. {period}")
    else:
        print("\n손상된 구간이 발견되지 않았습니다.")
```

```
def main():
    if len(sys.argv) != 2:
        print("사용법: python script.py <RMI 파일경로>")
        return

    file_path = sys.argv[1]
    try:
        parse_rmi(file_path)
    except Exception as e:
        print(f"오류: {e}")

if __name__ == "__main__":
    main()
```

[별첨 1] analyze_rmi.py.

```
#!/usr/bin/env python3
import os
import sys
import subprocess
from pathlib import Path

def extract_video(rms_file):
    """RMS 파일에서 H.264 영상 추출"""
    with open(rms_file, 'rb') as f:
        data = f.read()

    # H.264 NAL 유닛 찾기
    nal_positions = []
    start_codes = [b'\x00\x00\x00\x01', b'\x00\x00\x01']

    for start_code in start_codes:
        pos = 0
        while pos < len(data):
            pos = data.find(start_code, pos)
            if pos == -1:
                break
            nal_positions.append(pos)
            pos += 1

    nal_positions = sorted(set(nal_positions))

    if not nal_positions:
        return None

    # H.264 raw 파일 생성
```

```
    output_h264 = rms_file.replace('.rms', '_extracted.h264')

    with open(output_h264, 'wb') as out_f:
        for i, pos in enumerate(nal_positions):
            if i + 1 < len(nal_positions):
                size = nal_positions[i + 1] - pos
            else:
                size = len(data) - pos

            if size > 0:
                out_f.write(data[pos:pos + size])

    return output_h264

def convert_to_mp4(h264_file):
    """H.264 를 MP4 로 변환"""
    mp4_file = h264_file.replace('.h264', '.mp4')

    cmd = [
        'ffmpeg', '-y',
        '-f', 'h264',
        '-r', '25',
        '-i', h264_file,
        '-c', 'copy',
        mp4_file
    ]

    try:
        subprocess.run(cmd, capture_output=True, check=True)
        return mp4_file
    except:
        return None

def main():
    if len(sys.argv) != 2:
        print("Usage: python script.py <rms_file>")
        sys.exit(1)

    rms_file = sys.argv[1]

    if not os.path.exists(rms_file):
        print(f"File not found: {rms_file}")
        sys.exit(1)

    # H.264 추출
    h264_file = extract_video(rms_file)
    if h264_file:
```

```
        print(f"H.264 extracted: {h264_file}")

        # MP4 변환
        mp4_file = convert_to_mp4(h264_file)
        if mp4_file:
            print(f"MP4 created: {mp4_file}")
        else:
            print(f"FFmpeg conversion failed, use: {h264_file}")
    else:
        print("No H.264 data found")

if __name__ == "__main__":
    main()
```

[별첨 2] video_extractor.py.