소프트웨어응용보안(F056-1)

Team #2

201520890 박세환

201420948 김동현

201320878 박용채

201520892 이종인

201421273 이세린

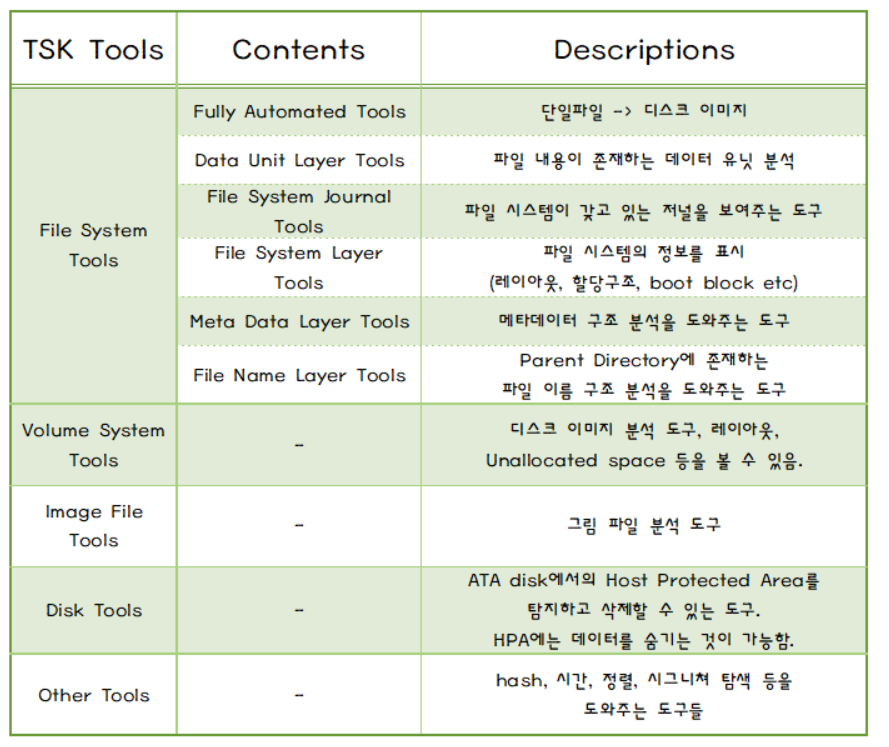
**Term Project#1 – TSK & Hikvision Analysis**

# **The Sleuth Kit**

**1. TSK 분석 개요**

The Sleuth Kit은 오픈소스 포렌식 도구로, 파일 및 볼륨 시스템부터 이미지 분석과 Hash, Signature 기반 분석 등을 수행한다. 그 중에서도 특정 이미지 파일에 대한 분석을 통해 삭제된 파일까지 복원해주는 기능인 tsk\_recover는 오픈소스 툴임에도 강력한 기능을 자랑한다. 현재 tsk\_recover가 지원하는 파일시스템은 대중적으로 사용되고 있는 NTFS, FAT32, Ext3/4 등이 있지만, 분석하고자 하는 디스크의 파일시스템을 지원하지 않는 경우가 생길 수 있다. 이때, TSK는 오픈소스이므로, 직접 소스를 분석하고 툴을 실행시켜 보면서 도구에 대한 깊은 이해를 바탕으로 특정 파일 시스템에 대한 복구가 가능하게 코드를 추가할 수 있다. 따라서, 본 보고서에서는 TSK의 tsk\_recovery가 지원하는 파일 시스템을 추가한다는 관점에서 TSK를 분석하는 과정을 담는다.

**2. TSK 구조**(1. file system tools 2. volume system tools 3. image file tools ...)

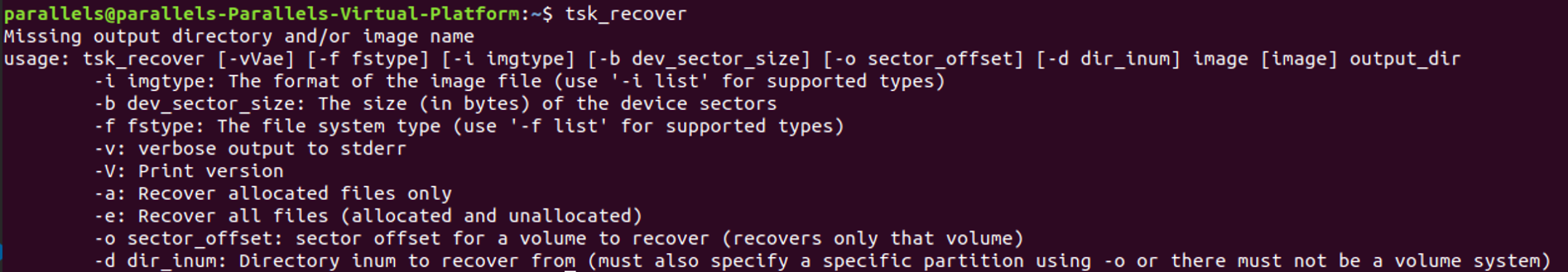
[표]

**3. 모듈별 기능 및 동작 구조 분석**(1.tsk\_recover 2. fsstat 3.ffind ...)

**3.1 tsk\_recover**

tsk\_recover는 TSK의 File System Tools 중 하나로서, 디스크 이미지 전체를 인자로 받아, 그 디스크의 파일들을 복원하는 기능을 한다. 지원하는 파일 시스템이 정해져 있으며, 디스크 이미지는 raw 파일만을 받는다.

우선, 어떻게 동작하는지를 보기 위해 기본값으로 실행한 결과이다.

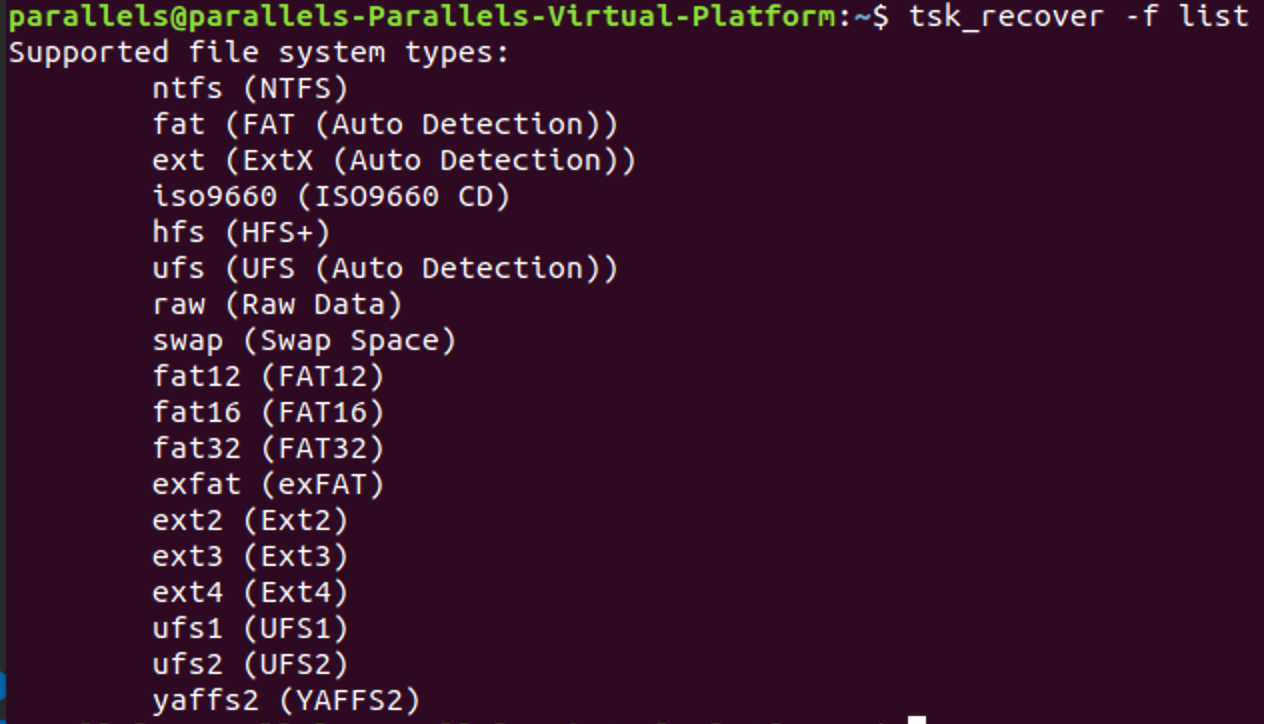
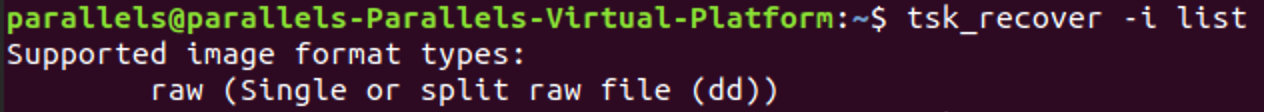


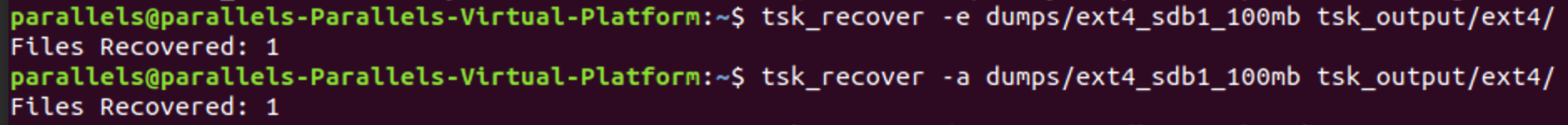
[그림] tsk\_recover 단순 실행 결과

‘Missing output directory and/or image name’ 라는 메시지와 usage가 출력된 것을 보아, 기본적으로 이미지 파일과, 출력 파일이 저장될 디렉토리를 필수적으로 지정해 주어야 한다는 것을 알 수 있다. 옵션값으로는 추가 인자가 필요없는 [vVae], 추가로 인자값이 필요한 [fibod]가 있다.

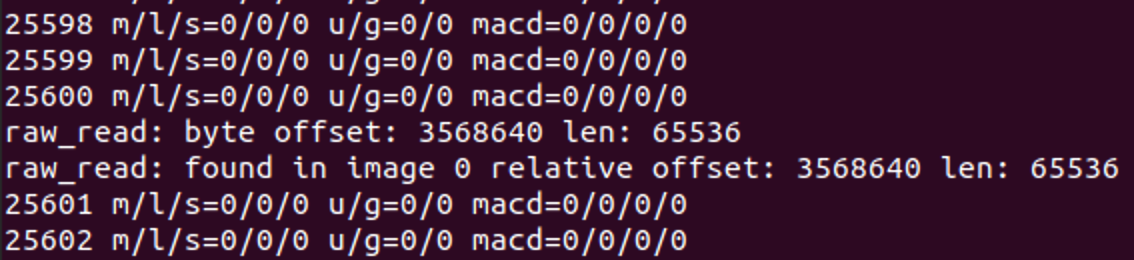
|  |  |  |
| --- | --- | --- |
| **옵션** | **기능** | **인자값** |
| -i | 입력값으로 넘겨 줄 디스크 이미지 파일의 형식을 지정한다.  인자값으로 imgtype이 아닌 ‘list’를 입력하면, 지원하는 이미지 파일의 형식을 출력해준다. | imgtype : 넘겨줄 이미지 파일의 형식. 현재는 raw 파일만을 지원한다. |
| -b | device sector의 size를 바이트단위로 입력한다. | dev\_sector\_size : 넘겨줄 디스크의 sector size |
| -f | 입력값으로 넘겨 줄 디스크 이미지의 파일 시스템 유형을 지정한다. 인자값으로 fstype이 아닌 ‘list’를 입력하면, 지원하는 파일 시스템을 출력해준다. | fstype : 넘겨줄 디스크의 파일 시스템 유형. ext4, ntfs ... |
| -v | 복구 과정에서의 모든 로그를 stderr로 출력해준다. | - |
| -V | 버전 정보를 출력하여 준다. | - |
| -a | allocated 영역의 파일들만 추출한다. | - |
| -e | unallocated를 포함한 모든 영역의 파일들을 추출한다. | - |
| -o | 복구할 볼륨의 시작 오프셋을 입력한다. | sector\_offset : 복구할 볼륨의 시작 오프셋 |
| -d | 복구할 볼륨의 디렉터리 inode 값을 전달한다. -o 옵션에서 해당 볼륨의 오프셋을 지정해 주어야 한다. | dir\_inum : 복구할 볼륨의 디렉터리 inode 값을 입력한다. |

[표] tsk\_recover 옵션 별 기능





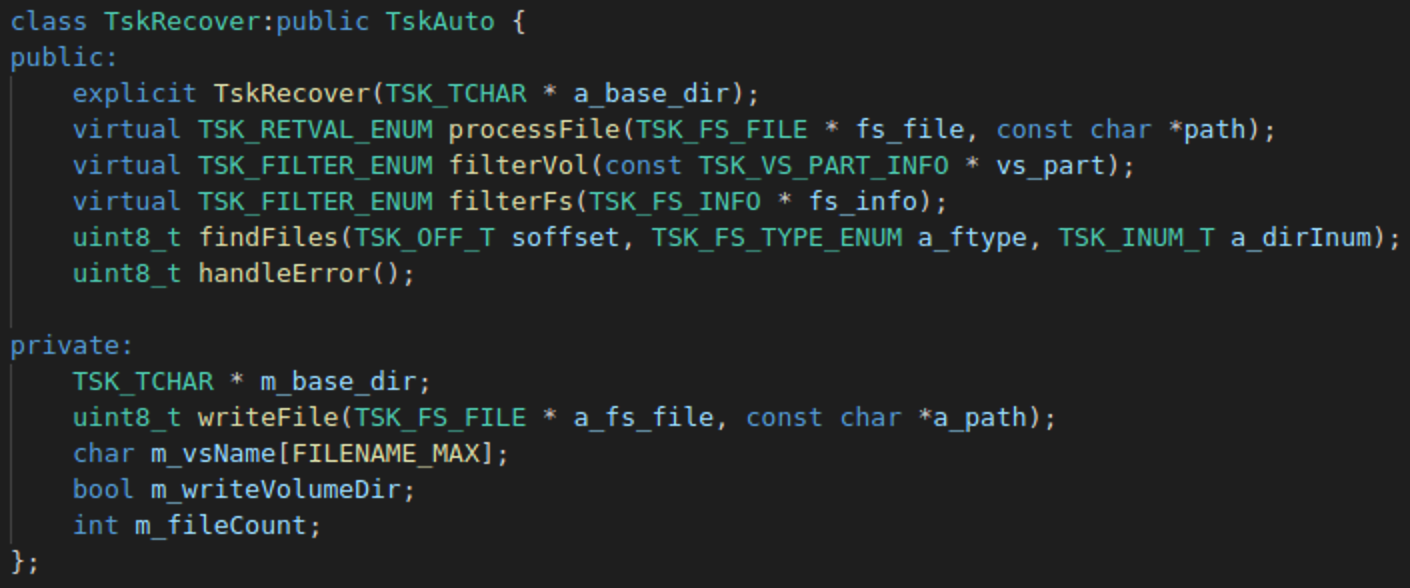




**3.1.1 tsk\_recover 소스 및 동작 구조 분석**

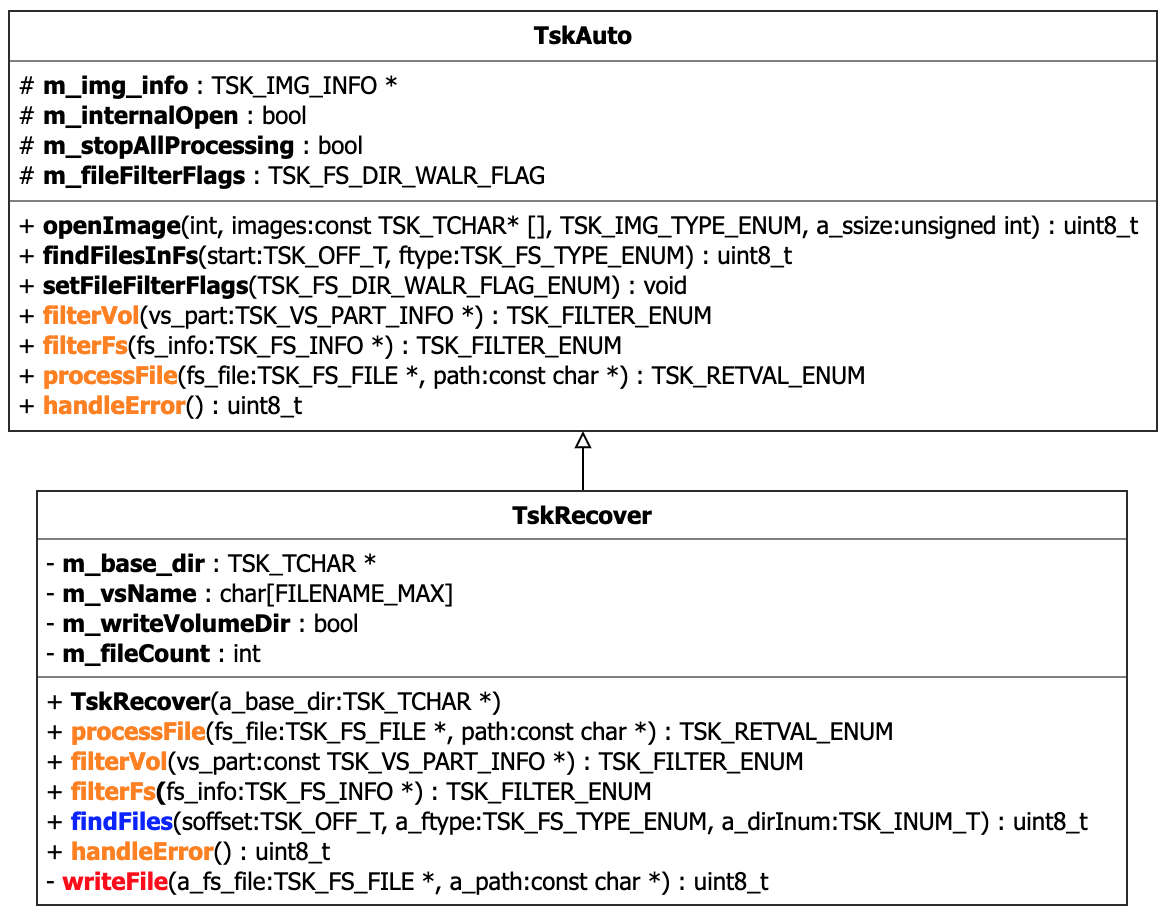
위에서 tsk\_recover라는 모듈이 어떻게 수행되는지를 빌드 및 실행을 통해 알아보았고, 다음으로는 소스를 분석하여 어떤 방식으로 파일들을 복구하는지, 동작 구조를 알아본다.

먼저, TSK의 File System Tools는 완전 자동화를 통해서 하나의 디스크 이미지를 여러가지 분석이 가능하도록 만들기 위해 TskAuto라는 클래스를 설계하고 있다. 그중에서도 tsk\_recover는 TskRecover라는 파생클래스를 정의해서 TskAuto를 상속받도록 하였다. tsk\_recover.cpp 에서 TskRecover를 정의하고 있는데, TskAuto 클래스로부터 파일 복구에 필요한 부분을 오버로딩 하고, 필요한 멤버 변수와 메소드를 추가로 지정하고 있다.



[그림] tsk\_recover.cpp 에서 TskRecover class를 정의하는 코드

아래의 그림은 TskRecover와 TskAuto의 상속 관계를 나타낸다. TskAuto는 TskRecover와 관련이 있는 부분만을 간략하게 나타내었다.



[그림] tsk\_recover.cpp에서 정의된 TskRecover class structure

TskRecover는 processFile, filterVol, filterFs와 handleError 메소드를 overloading한다. findFiles는 자체 정의된 메소드이나, 내부적으로 TskAuto의 findFilesInFs 메소드를 호출한다. writeFile은 private 메소드로 processFile 메소드 내부에서만 호출할 수 있도록 정의되어있다.

tsk\_recover는 실행하는 동안 하나의 TskRecover 인스턴스를 생성하고, 인스턴스 관리적인 측면은 모두 TskAuto로부터 virtual하게 처리된다. 다만 recovery라는 목적에서, 복구한 파일을 write 하는데에는 자체적으로 정의한 메소드를 사용하게 되고 그 외의 이미지 파일을 open해서 load하거나, 파일 시스템을 식별하고 파일들을 찾아내는 데에는 TskAuto에서 정의된 메소드를 그대로 사용하게 된다.

TskRecover 클래스를 정의하였으므로, 이 클래스를 인스턴스화 시켜 어떻게 복구를 하는지, 함수별로 분석하도록 한다.

**3.1.2 main**

|  |
| --- |
| TSK\_IMG\_TYPE\_ENUM imgtype = TSK\_IMG\_TYPE\_DETECT;  TSK\_FS\_TYPE\_ENUM fstype = TSK\_FS\_TYPE\_DETECT;  int ch;  TSK\_TCHAR \*\*argv;  unsigned int ssize = 0;  TSK\_OFF\_T soffset = 0;  TSK\_TCHAR \*cp;  TSK\_FS\_DIR\_WALK\_FLAG\_ENUM walkflag = TSK\_FS\_DIR\_WALK\_FLAG\_UNALLOC;  TSK\_INUM\_T dirInum = 0; |
| ch : 파싱된 옵션을 저장하기 위한 변수.  \*\*argv : 인자를 처리하기 위한 변수  ssize : sector size 저장 변수  soffset : offset 저장 변수  \*cp : sector size를 처리하는 과정에서 쓰일 포인터  walkflag : 복구할 부분을 확인하기 위한 변수  dirInum : 디렉토리 inum 저장 변수 |

|  |
| --- |
| case \_TSK\_T('a'):  walkflag = TSK\_FS\_DIR\_WALK\_FLAG\_ALLOC;  break; |
| 옵션 a는 할당된 영역에 해당되는 부분만 복구하기 위해 walkflag를 다음과 같이 설정해 준다. |

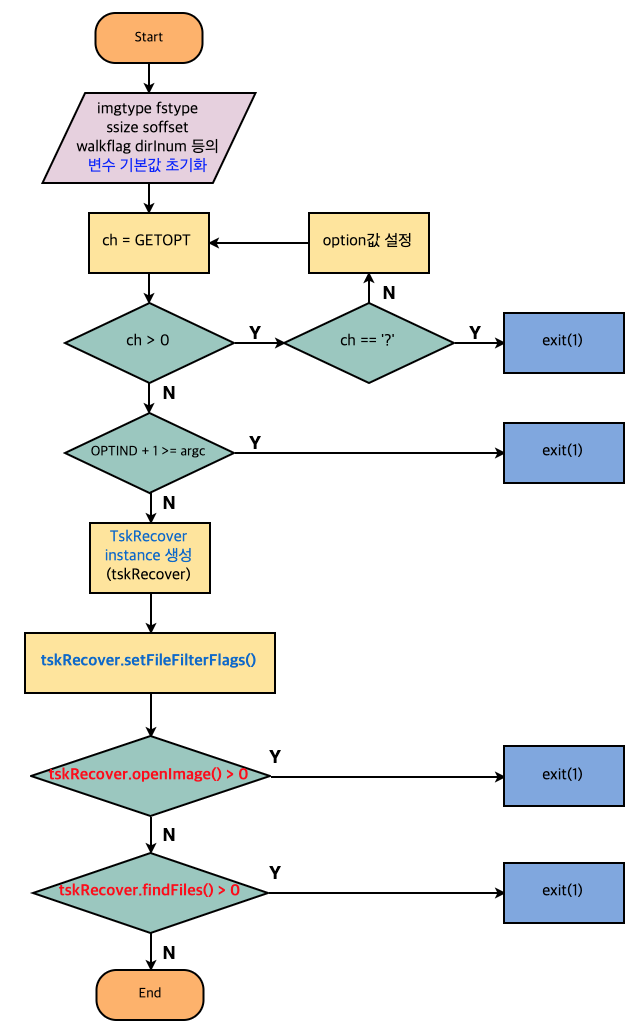
|  |
| --- |
| #define TSTRTOUL strtoul  case \_TSK\_T('b'):  ssize = (unsigned int) TSTRTOUL(OPTARG, &cp, 0);  if (\*cp || \*cp == \*OPTARG || ssize < 1) {  TFPRINTF(stderr,  \_TSK\_T  ("invalid argument: sector size must be positive: %s\n"),  OPTARG);  usage();  }  break; |
| TSTRTOUL은 라인#143에 strtoul을 TSTRTOUL로 정의하고 있다. strtoul은  unsigned long strtoul(const char \*nptr,char \*\*endptr,int base);  으로써 nptr로 가리키는 문자 스트링의 초기 부분을 부호없는 long integer값으로 변환한다. 그러다 숫자의 일부로 인식할 수 없는 첫 번째 문자에서 스트링 읽기를 중단한다. endptr 매개변수 값이 null이 아니면 스캔을 종료시킨 문자에 대한 포인터는 endptr에 저장된다. endptr이 null이거나 숫자 문자가 없거나 유효한 값이 아닐 경우 ERANGE(변환된 값이 범위를 벗어남) 따라서  if (\*cp || \*cp == \*OPTARG || ssize < 1)  //OPTARG가 숫자로 변환될 수 없거나 읽을 숫자가 없거나 sector size가 1보다 작은지를 검사 |

|  |
| --- |
| case \_TSK\_T('d'):  if (tsk\_fs\_parse\_inum(OPTARG, &dirInum, NULL, NULL, NULL, NULL)) {  TFPRINTF(stderr,  \_TSK\_T("invalid argument for directory inode: %s\n"),  OPTARG);  usage();  }  break; |
| 옵션 d의 경우 옵션 o의 경우 지정한 볼륨에서 복구할 디렉토리의 inum을 넘겨받은 parameter(오프셋)를 파싱해서 넘겨준다. |

|  |
| --- |
| case \_TSK\_T('e'):  walkflag =  (TSK\_FS\_DIR\_WALK\_FLAG\_ENUM) (TSK\_FS\_DIR\_WALK\_FLAG\_UNALLOC |  TSK\_FS\_DIR\_WALK\_FLAG\_ALLOC);  break; |
| 옵션 e의 경우 할당 영역과 비할당 영역을 모두 복구하기 때문에 다음과 같이 설정해준다. |

|  |
| --- |
| case \_TSK\_T('i'):  if (TSTRCMP(OPTARG, \_TSK\_T("list")) == 0) {  tsk\_img\_type\_print(stderr);  exit(1);  }  imgtype = tsk\_img\_type\_toid(OPTARG);  if (imgtype == TSK\_IMG\_TYPE\_UNSUPP) {  TFPRINTF(stderr, \_TSK\_T("Unsupported image type: %s\n"),  OPTARG);  usage();  }  break; |
| 입력값으로 넘겨 줄 디스크 이미지 파일의 형식을 지정한다. 인자값으로 imgtype이 아닌 ‘list’를 입력하면, 지원하는 이미지 파일의 형식을 출력해준다.  i와 list를 같이 입력하게 되면 오류메세지를 출력한다. |

|  |
| --- |
| progname = argv[0];  setlocale(LC\_ALL, ””); |
| 파일을 열고 생성할 때, 경로명 또는 파일명에 한글이 포함되었을 경우 파일 입출력이 동작하지 않는다. 이를 해결하기 위한 방법으로 컴퓨터 시스템이 지역 설정을 따르게 하는 코드이다. |



[그림] tsk\_recover.cpp main flowchart

**3.1.3 findFiles**

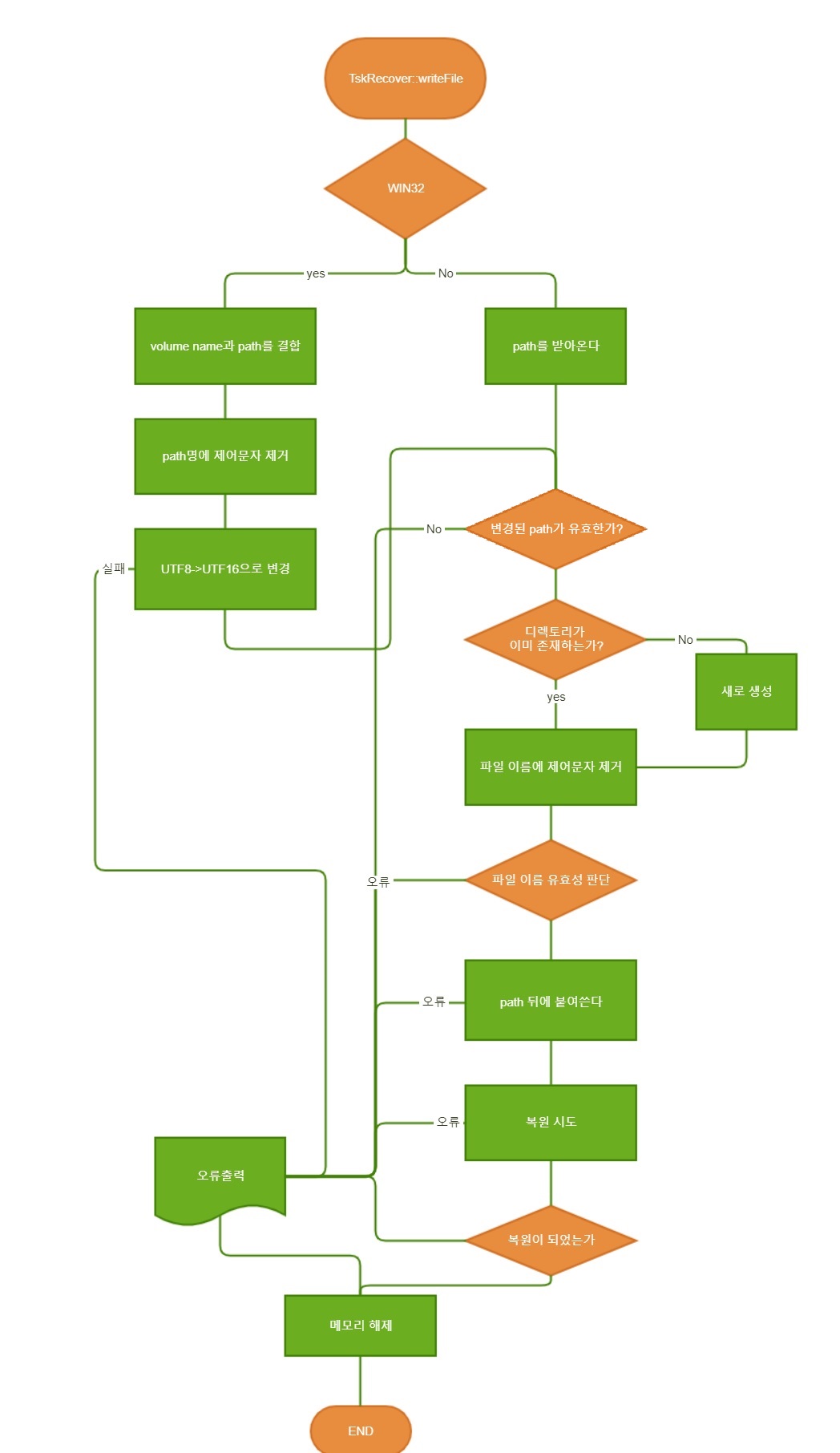
위 TskRecover에서 설명한 데로 findFiles는 자체 정의된 메소드이나, 내부적으로 TskAuto에서 상속받은 메소드 이다. tsk/auto/tsk\_auto.h를 확인해 보면 알 수 있다.

|  |
| --- |
| class TskAuto {  public:  ...  uint8\_t findFilesInImg();  uint8\_t findFilesInVs(TSK\_OFF\_T start);  uint8\_t findFilesInVs(TSK\_OFF\_T start, TSK\_VS\_TYPE\_ENUM vtype);  uint8\_t findFilesInFs(TSK\_OFF\_T start);  uint8\_t findFilesInFs(TSK\_OFF\_T start, TSK\_FS\_TYPE\_ENUM ftype);  uint8\_t findFilesInFs(TSK\_OFF\_T start, TSK\_INUM\_T inum);  uint8\_t findFilesInFs(TSK\_OFF\_T start, TSK\_FS\_TYPE\_ENUM ftype,  TSK\_INUM\_T inum);  uint8\_t findFilesInFs(TSK\_FS\_INFO \* a\_fs\_info);  TSK\_RETVAL\_ENUM findFilesInFsRet(TSK\_OFF\_T start,  TSK\_FS\_TYPE\_ENUM a\_ftype); |
| TskAuto 클래스 내에 선언되어 있다. 내용은 auto.cpp에서 확인 가능하다. |

**3.1.4 processFiles**

**3.1.5 writeFile**

file\_walk\_cb



[그림] tsk\_recover.cpp WriteFile Flowchart

**3.2 File System Layer Tools**

**3.2.1 fsstat**

fsstat은 레이아웃, 사이즈, 레이블, 메타 데이터 값(아이노드)과 콘텐츠 유닛(블록 또는 클러스터)의 범위, 마운트 시간과 같은 파일 시스템의 세부 정보를 출력한다.

출력값은 파일 시스템마다 다르게 나타난다. 그룹을 사용하는 파일 시스템(FFS, EXT2FS 등)의 경우에는 각 그룹의 레이아웃이 나열되며, FAT 파일 시스템의 경우에는 축약된 FAT 테이블이 표시된다.

사용할 수 있는 옵션값에는 [-t,f,i,o,b,v,V]가 있다.

|  |  |  |
| --- | --- | --- |
| **옵션** | **기능** | **인자값** |
| -t | 이미지 파일의 파일 시스템만 확인하여 출력한다. | - |
| -f | 이미지 파일의 파일 시스템을 입력해준다. ‘-f list’를 입력하여 지원되는 파일 시스템들을 확인할 수 있다. | [fstype] : 파일 시스템 타입 |
| -i | 이미지 파일의 타입을 입력해준다. ‘-i list’를 입력하여 지원되는 이미지 타입을 확인할 수 있다. | [imgtype] : 이미지 파일 타입 |
| -o | 이미지 파일의 파일 시스템의 시작 오프셋을 입력한다. | [imgoffset] : 시작 오프셋 |
| -b | 디바이스 섹터의 크기를 바이트 단위로 입력한다. | [dev\_sector\_size] : 섹터의 크기 |
| -v | 디버깅 내역의 상세한 정보를 표준 에러 스트림(stderr)으로 출력한다 | - |
| -V | 버전 정보를 출력한다. | - |

[표] fsstat 옵션 별 기능

**3.3 File Name Layer Tools**

**3.3.1 ffind**

ffind는 입력된 아이노드에 할당된 파일 또는 디렉토리 명을 찾아낸다. 기본적으로 가장 처음 찾아낸 파일명을 리턴하지만, 일부 파일 시스템의 경우에는 삭제된 파일명을 찾아내어 리턴하기도 한다.

또한 ffind는 입력된 아이노드를 찾는 모든 디렉토리 항목을 검색한다. 이는 ifind를 이용하는 디스크 유닛 주소로부터 아이노드가 확인되었을 때 유용하게 이용할 수 있다.

사용할 수 있는 옵션값에는 [-a,d,f,u,i,o,b,v,V]가 있다(중복되는 내용은 생략함).

|  |  |  |  |
| --- | --- | --- | --- |
| **옵션** | **기능** | | **인자값** |
| -a | 아이노드의 모든 occurrence를 찾아낸다. | | - |
| -d | 삭제된 항목만 출력한다. | | - |
| -u | 삭제되지 않은 항목만 출력한다. | | - |

[표] ffind 옵션 별 기능

**3.3.2 fls**

fls는 이미지 파일 내에 존재하는 파일 또는 디렉토리 명을 나열하고, 입력된 아이노드의 최근에 삭제된 파일명을 표시한다. 아이노드가 입력되지 않은 경우에는 루트 디렉토리의 아이노드가 사용된다.(NTFS 파일 시스템에서는 5, Ext3 파일 시스템에서는 2, …)

출력값에는 파일 타입, 메타데이터 주소, 파일명이 포함된다. 파일 타입은 ‘r/r’과 같이 표시되며, 첫 번째 파일 타입은 file name structure에 저장된 타입, 두 번째 파일 타입은 metadata structure에 저장된 타입을 의미한다. 보통은 두 파일 타입이 서로 일치하지만, 삭제된 파일의 경우에는 서로 다른 파일 타입이 표시될 수 있다.

|  |  |
| --- | --- |
| **타입** | **기능** |
| - | 알려지지 않은 형식 |
| r | 일반 파일 |
| d | 디렉토리 |
| c | 캐릭터 디바이스 |
| b | 블록 디바이스 |
| l | 심볼릭 링크 |
| p | named FIFO |
| s | 섀도우 |
| h | 소켓 |
| w | Whiteout |
| v | TSK 가상 파일/디렉토리 |

[표] fls에서 표시되는 파일 타입

사용할 수 있는 옵션값에는 [-a,d,D,f,F,l,m,p,r,s,i,o,b,u,v,V,z]가 있다(중복되는 내용은 생략함).

|  |  |  |
| --- | --- | --- |
| **옵션** | **기능** | **인자값** |
| -a | “.”과 “..” 디렉토리를 출력한다.(기본적으로는 출력하지 않음) | - |
| -D | 디렉토리 항목만 출력한다. | - |
| -F | 파일(디렉토리가 아닌 모든 항목) 항목만 출력한다. | - |
| -l | 파일 타입, 아이노드, 파일명, 마지막으로 수정한 시간, 마지막으로 액세스한 시간, 생성된 시간, 크기, uid, gid 와 같은 세부 정보를 출력한다 | - |
| -m | 타임라인이 형성될 수 있도록 time machine 형식으로 출력한다. | [mnt] : 마운팅 포인트 |
| -p | 각 항목의 전체 경로(full path)를 출력한다. | - |
| -r | 디렉토리를 재귀적으로 표시한다. 삭제된 디렉토리에는 해당되지 않는다. | - |
| -s | 오리지널 시스템과의 시간차를 초 단위로 입력한다. | [seconds] : 시간값 |
| -z | 시스템의 시간대(EST, GMT 등)를 입력한다. | [zone] : 시간대 |

[표] fls 옵션 별 기능

**3.4 Meta Data Layer Tools**

**3.2.1 icat**

icat는 주어진 이미지의 메타데이터를 기반으로 특정 파일의 내용을 보여주는 도구로 삭제된 파일을 복구 해 주는 기능을 제공한다.

사용할 수 있는 옵션값에는 [f, h, r, R, s, i, o, b, v, V, image, inode]가 있다(중복되는 내용은 생략함).

|  |  |  |
| --- | --- | --- |
| **옵션** | **기능** | **인자값** |
| -h | sparse files를 스킵한다면 절대 주소가 손상된다. 이 옵션은 sparse files를 복사할 때 공간을 절약한다. |  |
| -r | 삭제 된 파일을 복구한다. |  |
| -R | 삭제 된 파일을 복구할 때 복구 오류를 제어한다. |  |
| -s | 파일을 출력할 때 빈 공간을 포함한다.. | [imgoffset] : 시작 오프셋 |
| -i | 디바이스 섹터의 크기를 바이트 단위로 입력한다. | [dev\_sector\_size] : 섹터의 크기 |
| -images | 읽을 디스크 또는 파티션 이미지(-i로 지정된 형식)를 여러 세그먼트로 분할 할 경우 여러개의 이미지 파일 이름을 지정 할 수 있다. |  |
| -inode | inode 번호, icat는 모든 지정된 파일 내용을 연결한다. |  |

[표] icat 옵션 별 기능

**3.2.2 ifind**

ifind는 주어진 인자 값에 대한 메타데이터를 출력한다.

사용할 수 있는 옵션값에는 [a, d, l, n, p, i, b, f, o, v, V, z]가 있다(중복되는 내용은 생략함).

|  |  |  |
| --- | --- | --- |
| **옵션** | **기능** | **인자값** |
| -a | 모든 inode 출력한다.(블록으로 찾을 경우에만 해당) |  |
| -d | 인자로 주어진 블록의 메타데이터를 출력한다. | [unit\_addr] |
| -l | long형으로 메타데이터의 번호를 출력, -p,n,d 옵션이 필수적으로 주어져야한다. |  |
| -n | 주어진 인자의 파일이름의 메타데이터를 출력한다. | [file] |
| -p | 주어진 메타데이터 주소에서 비할당 MFT 엔트리 출력한다.-(NTFS 전용) | [par\_addr] |
| -z | 물리적인 지역의 표준 시간대, 만약 -l,p 옵션이 주어지면 자동적으로 시간대를 맞춘다. |  |

[표] icat 옵션 별 기능

**3.2.3 ils**

ils는 inode를리스트화 하여 출력 해 주는 도구이다. 기본적으로 비할당 inode를 출력한다.

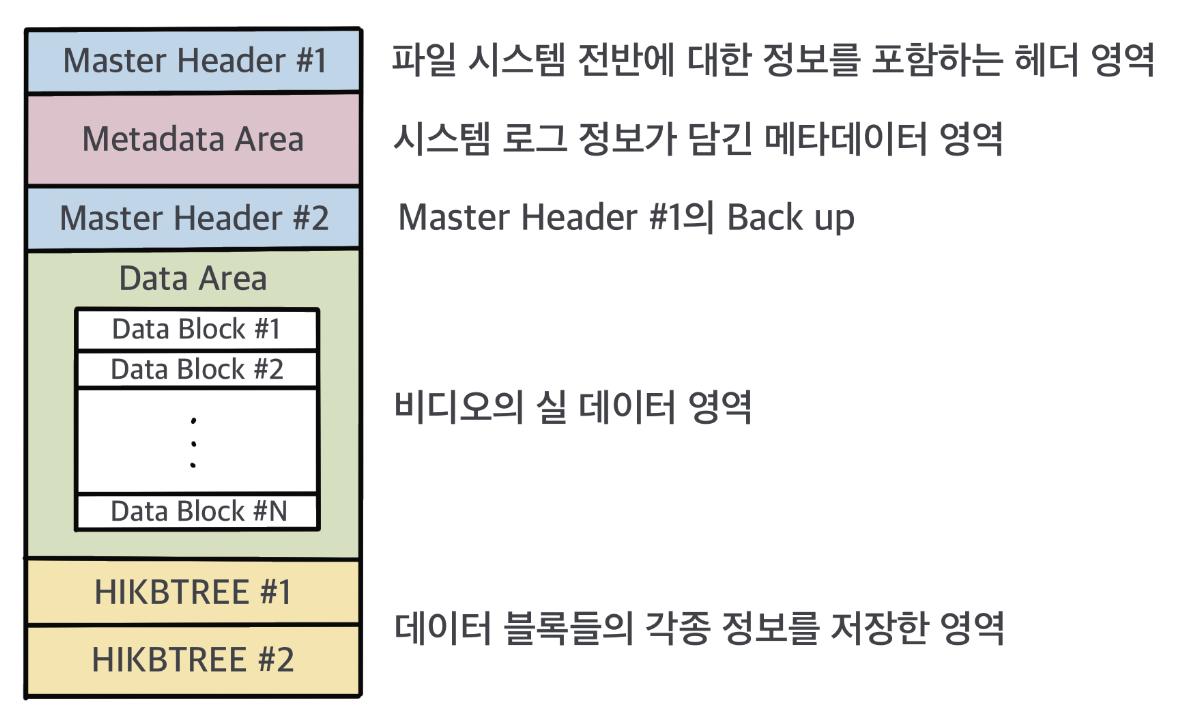
사용할 수 있는 옵션값에는 [e, m, O, p, a, A, l, L, z, Z, i, b, f, o, v, V]가 있다(중복되는 내용은 생략함).

|  |  |  |
| --- | --- | --- |
| **옵션** | **기능** | **인자값** |
| -e | 모든 inode 출력한다. |  |
| -m | MAC (Modify, Access, Create) 시간형식으로 출력한다. |  |
| -O | 비할당 되어 있으며 링크되지 않은 inode 출력한다.(UFS/Ext 전용) |  |
| -p | 고아 inode 출력한다. |  |
| -s | 시스템의 원래 시간을 지정한다. |  |
| -A | 비할당 inode 출력 |  |
| -l | 링크되어 있는 inode 출력 |  |
| -L | 링크되어 있지 않은 inode 출력 |  |
| -z | 사용 중이지 않은 inode 출력 |  |
| -Z | 사용 중인 inode 출력 |  |

[표] icat 옵션 별 기능

**The Hikvision Filesystem**

Hikvision사의 DVR 제품인 DS-7200 Series 모델은 독자적인 파일 시스템을 사용하는데, 이 파일시스템은 구조가 잘 알려져 있지 않다. 그런데 아직 직접 파일을 분석해 보지 못한 상황이므로, 정확하게 알려져 있는 부분만을 분석하여 정리하고자 한다.



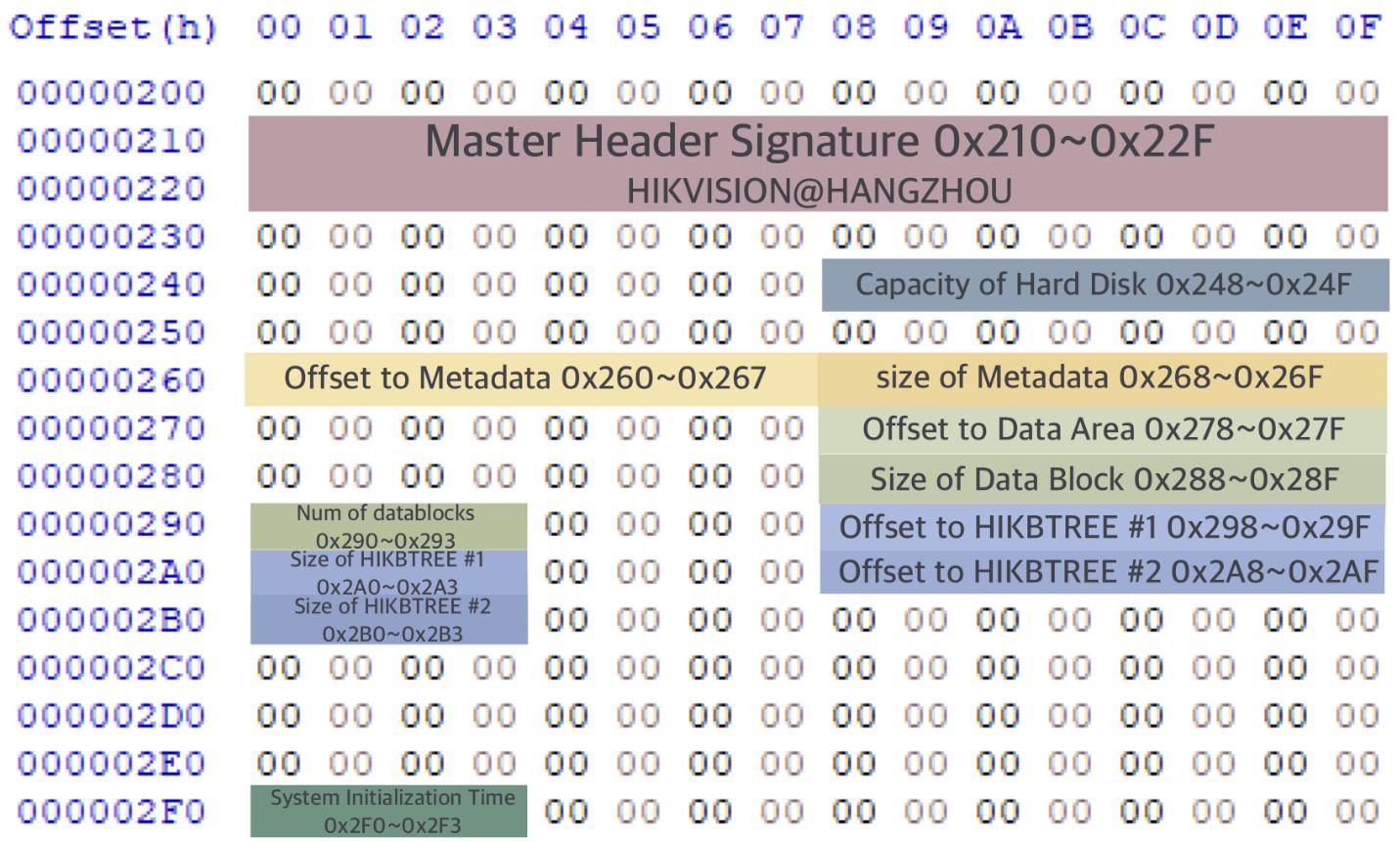
[그림 2-1] Hikvision File System Structure

[그림 2-1]은 Hikvision의 전반적인 구조를 나타낸다. 크게 네 부분으로 나눌 수 있고, Master Header, Metadata Area, Data Area 그리고 HIKBTREE영역이라고 부를 수 있다.

1. Master Header

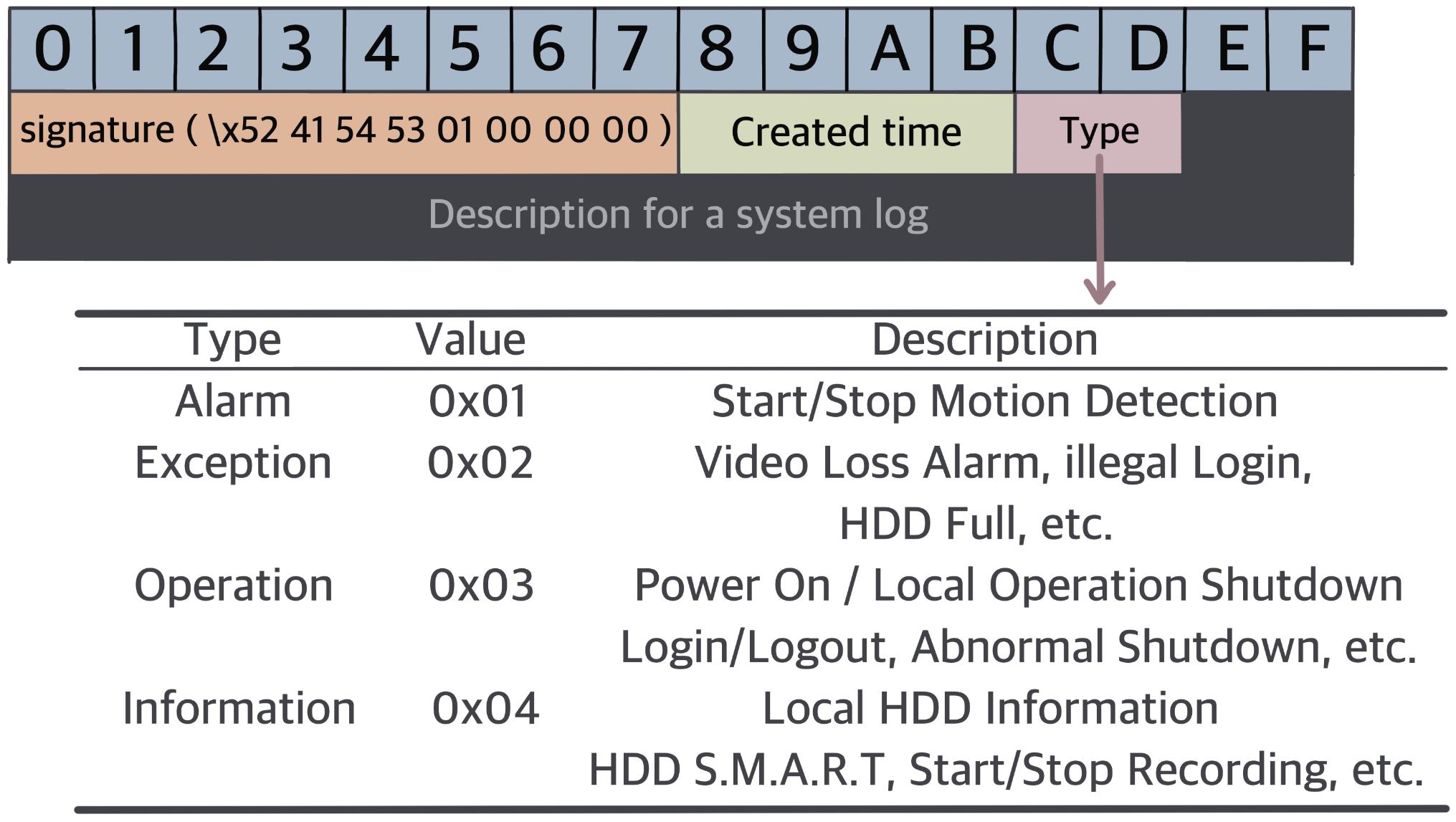
영역마다 특정 헤더영역을 포함하고 있기도 하므로, 전체 구조에서 가장 앞부분을 차지하는 헤더를 특별히 Master Header라고 칭한다. 이 영역은 파일의 0x200부터 0x100(256bytes)만큼을 차지하고, 어떤 파일시스템인지를 결정짓는 signature value로 ‘**HIKVISION@HANGZHOU**’를 포함한다. 이 Master Header의 Backup 데이터는 system logs의 뒤에 위치하며 완벽히 일치하는 데이터를 포함한다.

Master Header가 포함하는 중요 데이터들은 다음과 같다.



1. Metadata Area

Metadata Area는 시스템 로그 등의 metadata를 포함하고 있다. 주로 장치와 관련된 시스템 로그들을 포함하고 있으며, 시작 offset은 Master Header로부터 얻을 수 있다. 이 영역에서 시스템 로그들은 특정 패턴을 가지는데, 가장 앞부분에 signature로 ‘**RATS(0x52 41 54 53 01 00 00 00)’**을 가지고, 뒤이어 4byte Created time, 2byte Type을 고정적으로 갖는다. 그 뒤에 이어지는 Description은 어떤 로그인지에 따라 내용과 길이가 달라진다.



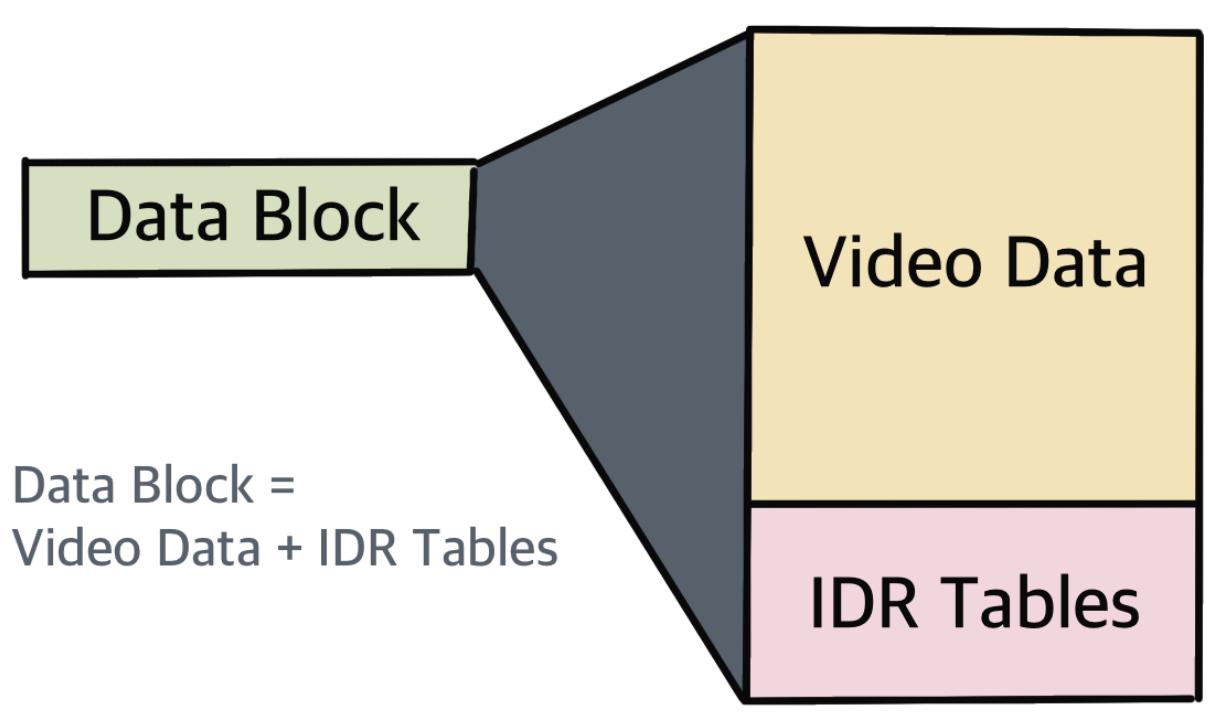
이 영역의 데이터들은 영상 데이터 복원 과정에서 유의미한 값은 아니라고 보여지므로, 데이터의 무결성을 위한 DVR의 ON|OFF 로그 정도만을 체크하고 넘어가면 될 것으로 판단된다.

1. Data Area

기록된 영상의 실제 데이터는 Data Area에 저장된다. Data는 Block단위로 기록되며, 보통 Block size는 1GB이다. 첫 Data Block과 Block size정보는 Master Header에서 얻을 수 있으나, HIKBTREE영역에서 접근되는 것이 일반적이다. 이는 보통의 경우 한 Block당 한 영상이 기록되지만, 용량이 작을 경우 한 Block에 두개의 영상이 기록되기도 하며, 데이터의 overwriting 여부를 Data Area에서는 알 수 없기 때문으로 보인다.

Data Block은 크게 Video Data와 IDR Tables로 구성된다. Video Data는 실제 영상 데이터인데, H.264로 인코딩 되어 저장된다. 이 Video Data가 Data Block의 대부분을 차지하며, 우리가 복원할 대상이 된다.

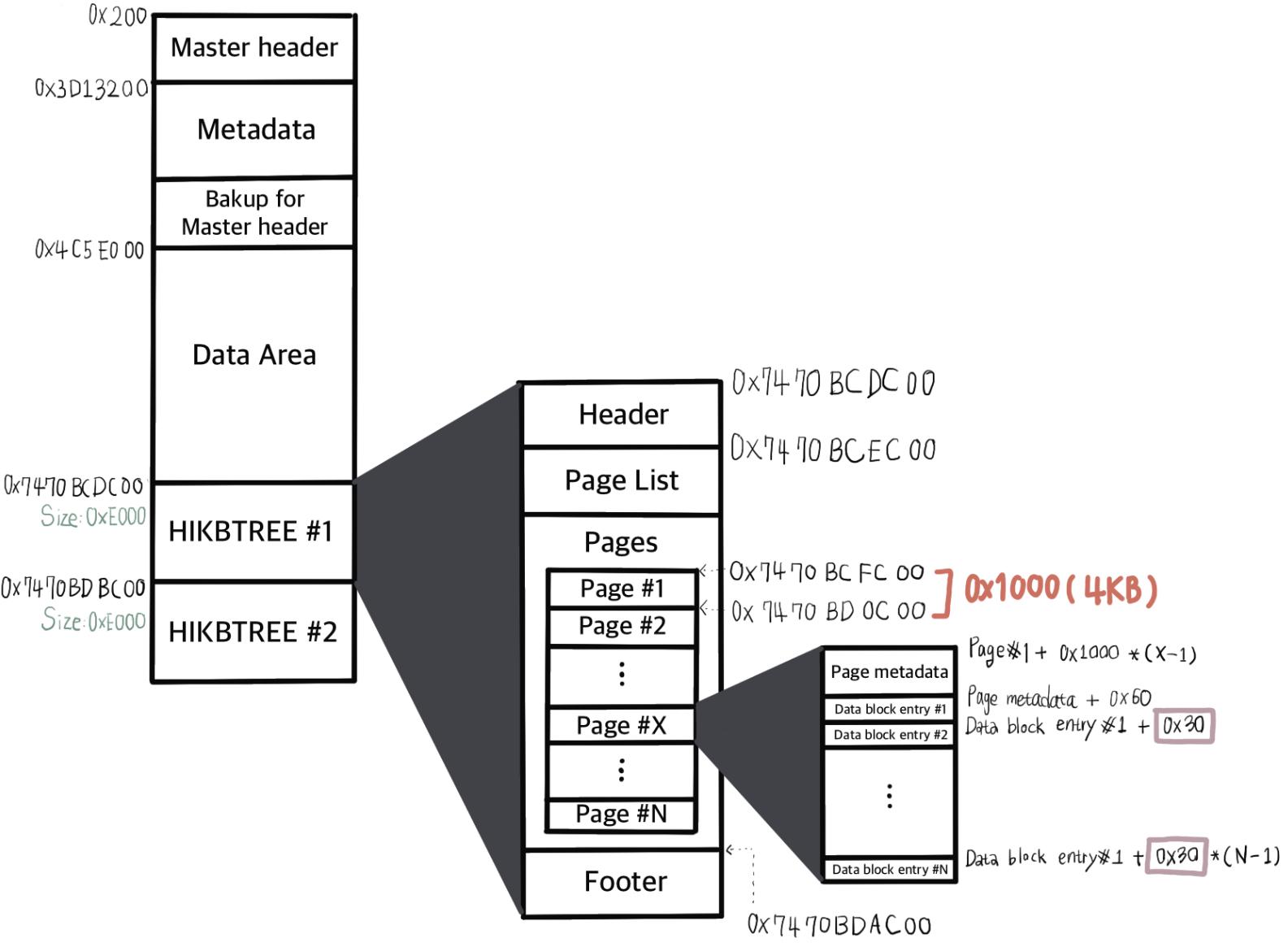
IDR Tables은 IDR의 metadata를 table 단위로 저장한다. 예를들어 index of IDR picture, 채널 번호 그리고 녹화 시간 등이 포함된다. 각 IDR table은 ‘OFNI(0x4F 46 4E 49)’라는 signature를 포함하여 총 56bytes의 길이를 가지며, IDR table가 여러개 연달아 이어지는 형태로 IDR Tables를 구성한다. IDR Table 내 metadata는 영상 복원에 영향을 줄 수 있는데, 파일이 overwrite되었는지 여부를 IDR Table의 녹화시간을 실제 기록된 녹화시간과 비교하여 판단할 수 있기 때문이다.



1. HIKBTREE

HIKBTREE는 Data Block에 대한 정보들을 담고 있다. 따라서 Master Header에서 HIKBTREE offset과 size 값을 얻어 이동한 뒤, 실제 영상 데이터들에 접근하는 방식으로 영상 데이터를 복원할 수 있다. HIKBTREE라고 불리우는 이유는 이 영역의 시작부분에 signature로 보여지는 **‘HIKBTREE’**가 존재하기 때문이다. HIKBTREE내에는 전반적인 데이터를 포함하는 Header부분과 Pages 데이터(Page list + Pages), 그리고 Footer 데이터가 존재한다. 여기서 Page는 Data Block Entry라고 불리우는 Data Block 영역의 정보를 가지는 데이터 단위이다. 즉, 어떤 영상이 어느 위치에 저장되어 있는가를 mapping시키는 역할을 한다.

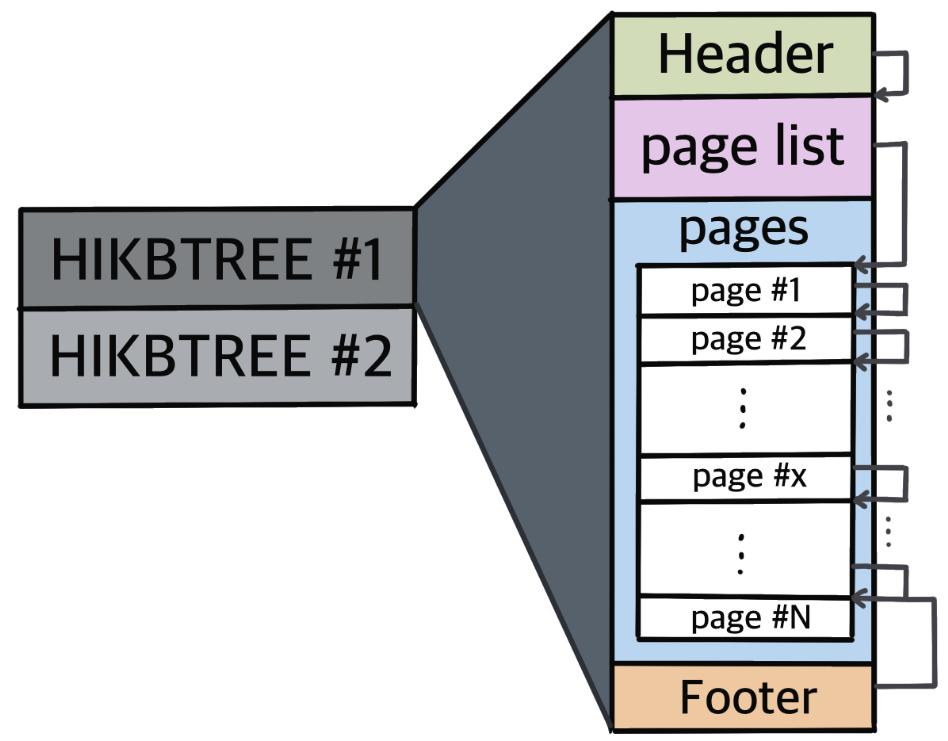
BTREE라는 이름에서 알 수 있듯이, HIKBTREE는 여러 부모노드와 자식노드들을 가지는 구조로 되어있고, 그 구조는 아래와 같다.



하나의 HIKBTREE는 Header, Page list, Pages 그리고 Footer를 가진다. 하나의 HIKBTREE가 여러개의 Page들을 포함하고, 그 Page역시 여러개의 Data Block Entry들을 포함하는 구조를 보인다.

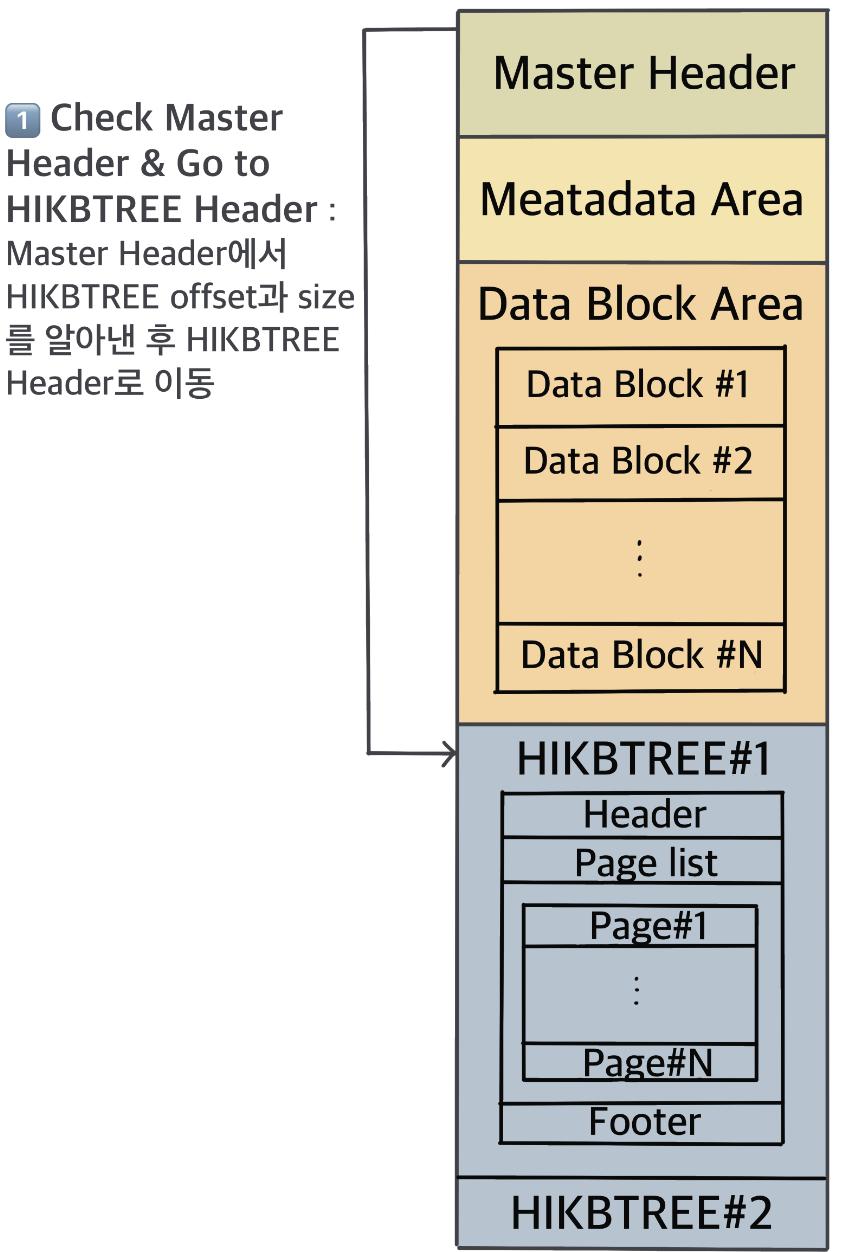
Header는 signature(HIKBTREE)와 생성 날짜, Page list, 첫번째 page, Footer 의 Offset을 포함한다. Page list는 총 page의 개수와 첫번째 page의 Offset을 포함한다. Footer는 last page offset을 가지므로, End of Page를 판단하기에 적합해보인다.

마지막으로 pages area는 Data Block entries를 포함하는 page들의 연속된 list로 구성되는데, page들은 4KB단위로 저장되어 있다. 각 page는 다음 page의 offset을 가지는 메타데이터부분(0x60 bytes 크기)과 여러개의 Data Block Entries(각 0x30 bytes 크기)로 구성되고, 각 Data Block Entry는 영상의 존재 여부와 Record Start/End time과 영상 채널번호, Data Block Offset을 가진다. 마지막 Block Entry는 영상이 존재하는지 여부가 0xFF 로 되어있음을 통해 알 수 있다.

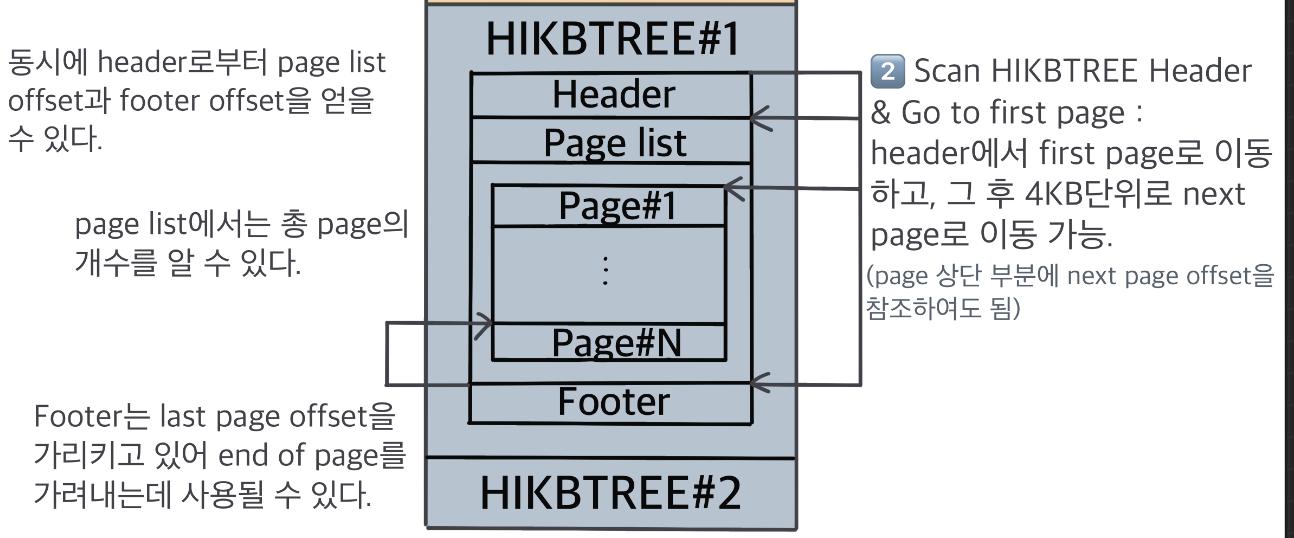


1. Video Data Recovery

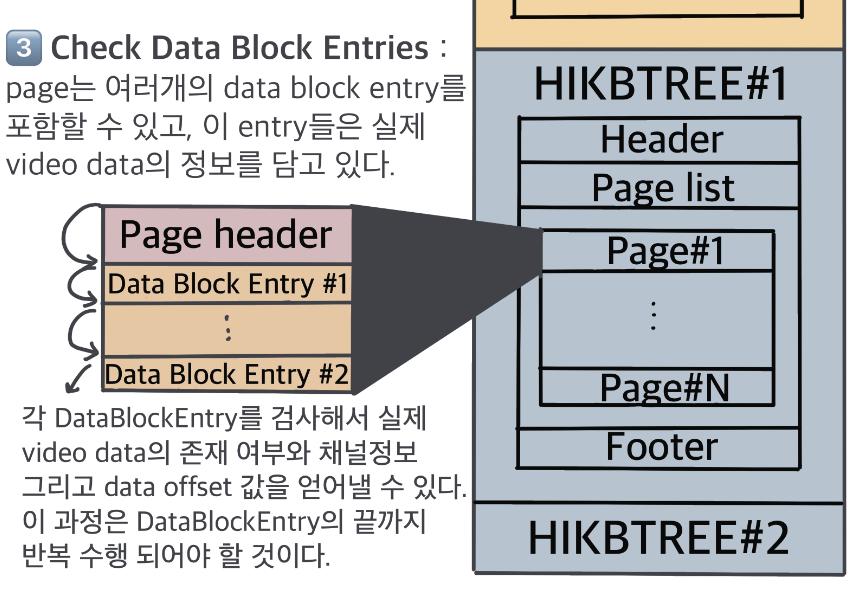
파일시스템의 구조를 통해 Video Data를 복구하는 과정은 다음과 같다.



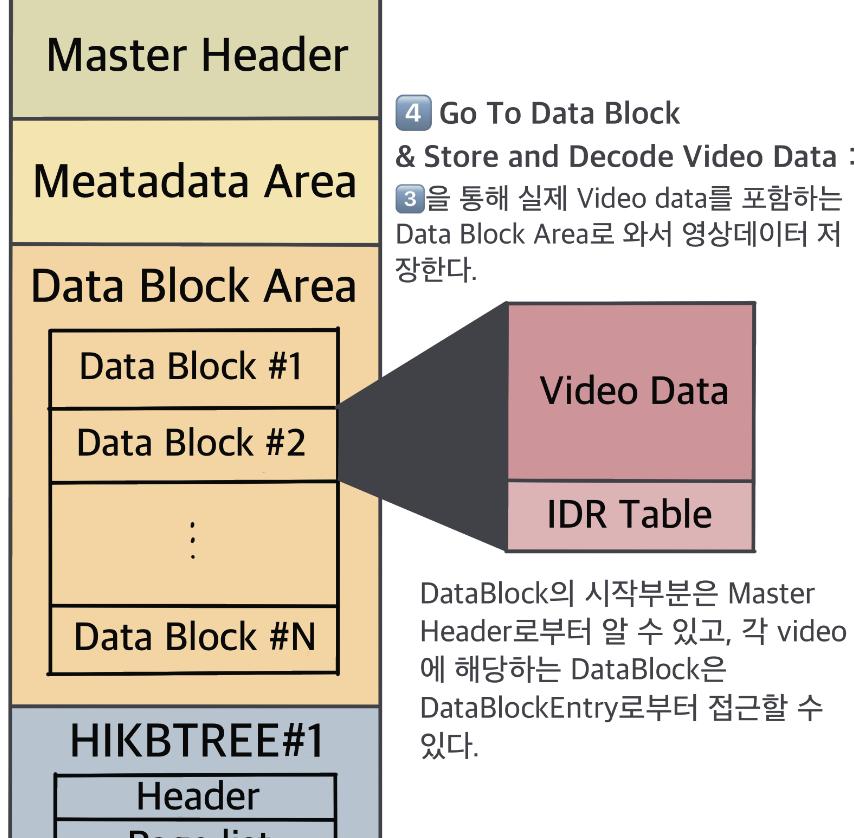
1️⃣ Master Header에서 필요한 정보를 수집한다. Master Header에서는 영상 복구에 필수적인 정보인 HIKBTREE offset과 size, 그리고 Data Block Area의 시작 offset과 Block size(보통의 경우 1GB로 고정), 그리고 Block의 개수를 알아낸다. 그 후 HIKBTREE로 이동하여 Video Data 복구에 필요한 추가 정보를 수집한다.



2️⃣HIKBTREE header로 이동하면, 바로 Data Block 영역정보가 담긴 entry가 나오지 않고, 또다른 단위인 page정보가 등장하므로, 그 page에 접근하기 위한 정보를 수집한다. HIKBTREE header에는 page list, page, footer offset이 저장되어 있다. 여기서 page list로 이동하면 총 page의 개수를 알 수 있고, page로 이동하여 Data Block entry에서 Data Block에 대한 정보를 수집하고 footer에서 그 수집을 끝낼 수 있게 된다.



3️⃣은 이러한 일련의 과정을 보여주는데, page 스캔을 통하여 Data Block entry에 접근하게 되면, 그 안에 포함된 정보인 “영상 존재 여부”를 파악한다. 이는 8byte길이로 0x00이 채워지면 영상이 존재, 0xff가 채워지면 영상이 존재하지 않음을 의미한다. 영상이 존재하면 해당 영상이 촬영된 채널과 그 영상 데이터가 저장된 offset을 알 수 있고, 영상이 기록된 시간에 대한 정보도 얻어낼 수 있다.



4️⃣Data Block Area에 접근하기 위한 영역 정보가 수집되면, Data Block으로 이동해서 영상 데이터를 저장해내면 된다. 이 영상은 H.264로 인코딩 되어있기 때문에 해당 디코더를 통해 볼 수 있을 것이다.

# **Conclusion**

**4. 추후 수정 방향**

|  |  |
| --- | --- |
| **소스파일** | **수정 방향** |
| TskData.java | TSK\_FS\_TYPE\_ENUM에  TSK\_FS\_TYPE\_HIKVISION, TSK\_FS\_TYPE\_HIKVISION\_DETECT 추가 |
| tsk\_fs.h | TSK\_FS\_TYPE\_ENUM에  TSK\_FS\_TYPE\_HIKVISION, TSK\_FS\_TYPE\_HIKVISION\_DETECT 추가 |
| tsk\_fs.h | HIKVISION 파일시스템인지 검사하는 TSK\_FS\_TYPE\_ISHIKVISION() 매크로 함수 추가 |
| fs\_types.c | fs\_type\_table 구조체 배열에 HIKVISION 항목 추가 |
| fs\_open.c | tsk\_fs\_open\_img() 함수에 HIKVISION 항목 추가 및 분기문 추가 |
| tsk\_fs\_i.h | extern TSK\_FS\_INFO \*HIKVISION\_open추가 |
| hikvision.c | HIKVISION\_open() 함수 개발  HIKVISION\_close() 함수 개발  HIKVISION\_dir\_open\_meta() 함수 개발 |
| tsk\_HIKVISION.h | HIKVISION.c에서 사용될 변수 및 함수를 포함한 헤더 추가 |

⓵tsk\_fs.h와 tsk\_types.c는 HIKVISION File System 식별을 위한 enum 상수와 구조체 오브젝트를 추가하여야 한다.

⓶ fs\_open.c 에는 HIKVISION가 식별될 경우 해당 파일을 여는 함수로 분기할 수 있도록 분기문을 추가하여야 할 것이다.

⓷tsk\_fs\_i.h 에서는 HIKVISION을 처리하 소스코드에서 정의된 함수들을 각각의 코드 들이 참조 수 있도록 extern으로 XFS 관련 함수를 모두 정의 하여야 한다.

⓸ tsk\_HIKVISION.h에는 HIKVISION file system 처리를 위한 구조체를 정의해 놓아야 한다.

⓹이후 실질적인 개발단계에서 HIKVISION의 File System을 잘 복구할 수 있도록 그에 맞는 함수

를 작성할 수 있도록 해야할 것이다.