International Journal of Internet, Broadcasting and Communication Vol.14 No.3 8-16 (2022)
http://dx.doi.org/10.7236/IJIBC.2022.14.3.8

IJIBC 22-3-2

A File/Directory Reconstruction Method of APFS Filesystem for Digital Forensics

Gyu-Sang Cho, Sooyeon Lim*

Professor, Dept. of Computer&Software, Dongyang University, Korea
**Professor, Dept. of Fine Arts, Kyungpook National University, Korea*
cho@dyu.ac.kr, sylim@knu.ac.kr

Abstract

In this paper, we propose a method of reconstructing the file system to obtain digital forensics information from the APFS file system when meta information that can know the structure of the file system is deleted due to partial damage to the disk. This method is to reconstruct the tree structure of the file system by only retrieving the B-tree node where file/directory information is stored. This method is not a method of constructing nodes based on structural information such as Container Superblock (NXSB) and Volume Checkpoint Superblock (APSB), and B-tree root and leaf node information. The entire disk cluster is traversed to find scattered B-tree leaf nodes and to gather all the information in the file system to build information. It is a method of reconstructing a tree structure of a file/directory based on refined essential data by removing duplicate data. We demonstrate that the proposed method is valid through the results of applying the proposed method by generating numbers of user files and directories.

Keywords: *Digital Forensics, File/Directory Tree Reconstruction, B-tree, Object Type Record, APFS Filesystem, macOS*

1. Introduction

APFS (Apple Filesystem) replaces HFS+ (Hierarchical File system Plus) as the default file system for iOS 10.3 and macOS High Sierra and later. APFS offers improved file system fundamentals as well as several new features, including cloning, snapshots, space sharing, fast directory sizing, atomic safe-save, and sparse files [1].

K. H. Hansen and F. Toolan published a representative study of APFS's file system analysis [2], and it is the first and very important study, providing very crucial data that can analyze the basic structure of the APFS file system. Another important source is provided by the Apple site that is Apple's Developer Reference provides fundamental information and data structures about APFS [3]. A. Dewald and J. Plum implemented approaches as a proof of concept tool and evaluate those against each other and against file carving, and the tool presented as an open-source implementation of a forensic file recovery tool for APFS [4]. From the perspective of digital forensics, there are still many areas to be investigated. Apple File System Reference is

Manuscript Received: May. 10, 2022 / Revised: May. 12, 2022 / Accepted: May. 15, 2022
Corresponding Author: sylim@knu.ac.kr
Tel: +82-53-950-5684, Fax: +82-53-950-5698
Professor, Dept. of Fine Arts, Kyungpook National University, Korea

provided to the apple developer site, but it is not satisfactory to fully analyze APFS. Researchers know more about the structure of APFS than before, but they have not yet fully analyzed its structure to a perfect level about it. A study on the development of tools for analyzing APFS file systems has been published. In the study, an "Object Identifier" obtained by parsing the object header of the cluster is used as the basis for "Searching Object Identifier", "Entire Disk Searching", "Object Tree Listing", and "Object Header Parsing" were implemented [5].

There are cases in which APFS file system tools have been developed and provided as open source. The APFS FUSE Driver for Linux project [6], "libfsapfs" project[7], "Apfsprogs" project[8] are used for expanding APFS filesystem to Linux, and a commercial software packages such as MacDrive and Paragon Software are used for Windows [9, 10].

In this paper, we propose a method to reconstruct a file/directory tree without information about the file system structure or the b-tree structure by exploring the B-tree cluster in case the APFS filesystem is partially corrupted or partially cluster deleted. This method is used as a countermeasure of an anti-forensics method for performing digital forensics on partially damaged disks.

2. Structures of APFS Filesystem Objects

2.1 APFS Overall Configuration

Figure 1 shows a diagram of the APFS overall configuration. The APFS starts with the first structure of the "Container Superblock(Magic Number: NXSB)". It contains pointers to the "Checkpoint Superblock Descriptor". The "Checkpoint Descriptor" has a multiple "Checkpoint" structure, which is copies of older "Container Superblocks." The "Checkpoint Superblock Data" contains the "Space Manager" shows the data of the entire container at a given point. The "Space Manager" points to the "Bitmap". The first "Container Superblock" stores a reference to an "Object Map (OMAP)" that points to its "OMAP Root Node." The "Checkpoint Superblock" points to the "Volume Checkpoint Superblocks(Magic Number: APSB)" of volumes in the filesystem. Each "Volume Superblock" points to a "Snapshots Block," a "Extents B-tree," and a "Root Directory Node" for the particular volume [4]. Our attention is focused only on the "File/Folder B-tree" leaf node. All the data stored on the disk used in this study is used to reconstruct the file/directory tree structure only with information stored on leaf nodes in the B-tree structure.

2.2 Record Type Related File and Directory Information

An APFS filesystem object stores information about a directory and a file on disk. The objects are stored as one or more records. The records are stored as key and value pairs in a B-tree, in which the key contains the object identifier and the record type used to look up a record, and the records are sorted to keep them ordered in a B-tree node. For a directory, a record of type APFS_TYPE_INODE for the inode, and a record of type APFS_TYPE_DIR_REC for the directory entry are required basically, and more types, e.g., APFS_TYPE_CRYPTO_STATE, APFS_TYPE_DIR_STATS and APFS_TYPE_XATTR, are required

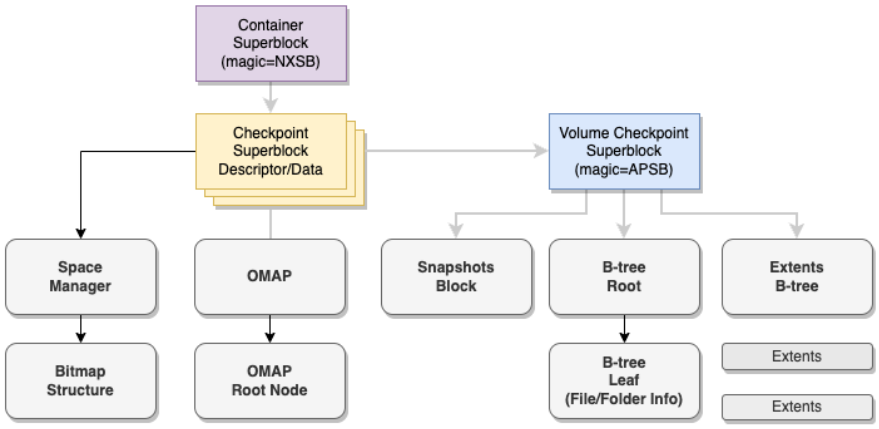
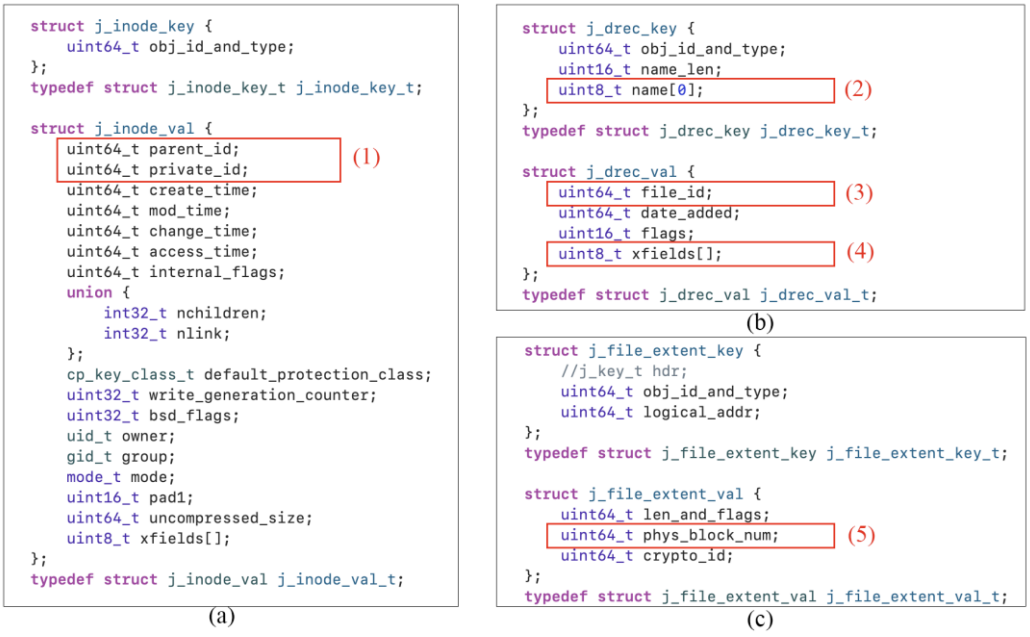


Figure 1. APFS overall configuration

additionally. For a file, two record of type APFS_TYPE_INODE and APFS_TYPE_FILE_EXTENT are required basically, and more types, e.g., APFS_TYPE_CRYPTO_STATE, APFS_TYPE_DSTREAM_ID, APFS_TYPE_EXTENT, APFS_TYPE_SIBLING_LINK and APFS_TYPE_XATTR, are required occasionally [3].



```
struct j_inode_key {
    uint64_t obj_id_and_type;
};
typedef struct j_inode_key_t j_inode_key_t;

struct j_inode_val {
    uint64_t parent_id;
    uint64_t private_id;
    uint64_t create_time;
    uint64_t mod_time;
    uint64_t change_time;
    uint64_t access_time;
    uint64_t internal_flags;
    union {
        int32_t nchildren;
        int32_t nlink;
    };
    cp_key_class_t default_protection_class;
    uint32_t write_generation_counter;
    uint32_t bsd_flags;
    uid_t owner;
    gid_t group;
    mode_t mode;
    uint16_t pad1;
    uint64_t uncompressed_size;
    uint8_t xfields[];
};
typedef struct j_inode_val j_inode_val_t;
```

(1)

(a)

```
struct j_drec_key {
    uint64_t obj_id_and_type;
    uint16_t name_len;
    uint8_t name[0];
};
typedef struct j_drec_key j_drec_key_t;

struct j_drec_val {
    uint64_t file_id;
    uint64_t date_added;
    uint16_t flags;
    uint8_t xfields[];
};
typedef struct j_drec_val j_drec_val_t;
```

(2) (3) (4)

(b)

```
struct j_file_extent_key {
    //j_key_t hdr;
    uint64_t obj_id_and_type;
    uint64_t logical_addr;
};
typedef struct j_file_extent_key j_file_extent_key_t;

struct j_file_extent_val {
    uint64_t len_and_flags;
    uint64_t phys_block_num;
    uint64_t crypto_id;
};
typedef struct j_file_extent_val j_file_extent_val_t;
```

(5)

(c)

Figure 2. Structures of record type (a) inode_key/value (b) drec_key/value (c) file_extent_key/value

Among the various types of records, the crucial data structure is shown in Figure 2. Figure 2(a) shows j_inode_key/value, Figure 2(b) shows (b) j_drec_key/value, and Figure 2(c) shows j_file_extent_key/value. In the red box (1) of Figure 2(a), the private_id represents the unique number of the file/directory, and the parent_id represents the unique number of the parent of the directory to which it belongs. The name in the red box (2) of Figure 2(b) contains the name information of the related file/directory. The unique file number corresponding to the private_id of the red box(2) can be obtained in the red box (3) also. To obtain the number of the cluster where the file is stored, look up the value of the red square (5) in Figure 2 (c). In the extended

structure of the red box (4) in Figure 2, information on the byte size of the data stored by the corresponding file and the related name can be additionally obtained from the structures of xf_blob_t and x_field_t.

3. A File/Directory Reconstruction Method of APFS Filesystem

3.1 Overview

The procedure of a file/directory reconstruction method of APFS filesystem consists of four steps. The overall configuration diagram is depicted in Figure 3. The first step is called "B-tree Leaf Looking-Up". This step is the process of finding a cluster with an object identifier of 0x03(OBJECT_TYPE_BTREE_NODE) and B-tree node's flag of 0x02(BTNODE_LEAF) from the entire disk cluster. Values used as types and subtypes by the obj_phys_t structure. The second step is called "B-tree Leaf Parsing". In the step, B-tree key/value information in the cluster is decomposed to analyze the needed data. The main structures are btree_node_phys_t, btn_index_node_val_t, nloc_t, kvloc_t, and kvoff_t. The third step is called "Data Refining". In this step duplicate information obtained by decomposing key/value pairs contained in the cluster is removed and data is refined using the associated information. The fourth step is called "Reporting". The output of this step results from the result of the previous step, which includes "File/Directory Info Display" and "Reconstructed Tree Structure Display".

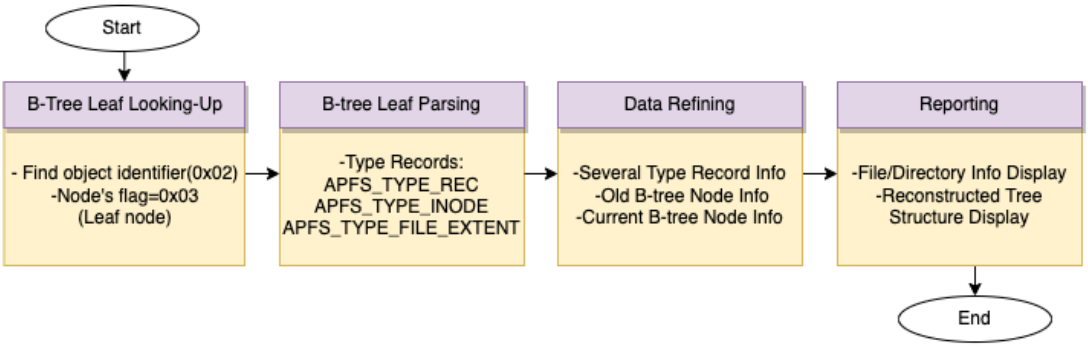
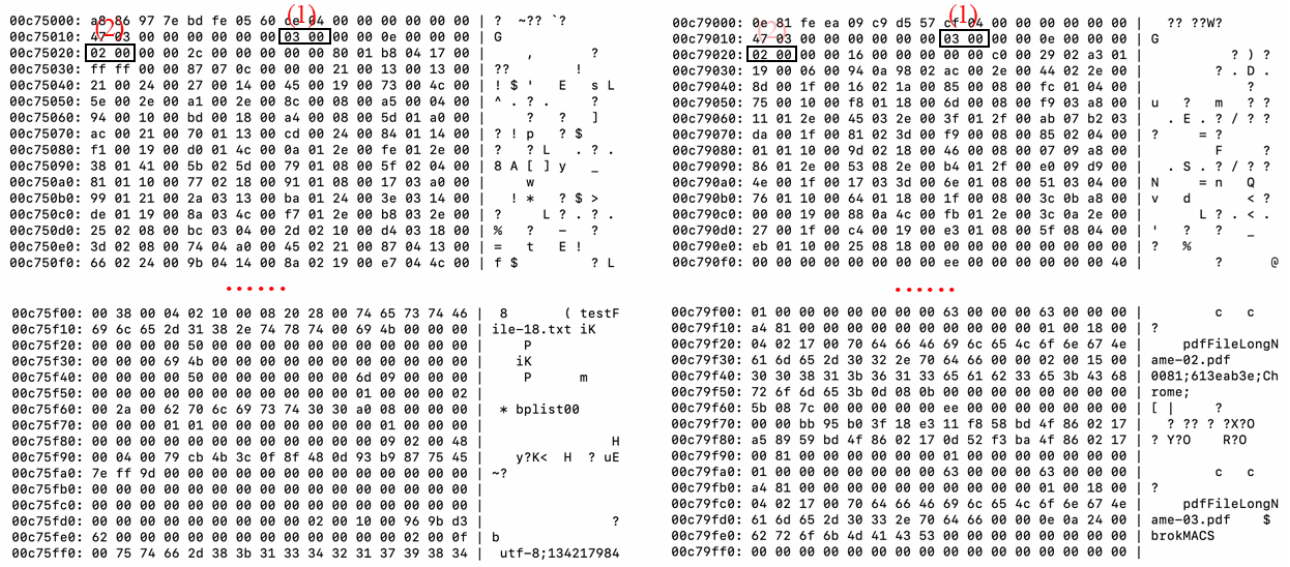


Figure 3. Procedure of file/directory tree reconstruction

3.2 Step 1: B-Tree Leaf Looking-Up

This step begins with the search for the 0x03(OBJECT_TYPE_BTREE_NODE) object identifier and B-tree node's flag of 0x02(BTNODE_LEAF) while traversing entire clusters. The object identifier can be found in the obj_phys_t structure includes five members, i.e., o_cksum[], o_oid, o_xid, o_type, and o_subtype. The o_oid contains object identifier number. Figure 4 shows cluster two screen shots of B-tree node. Figure 4(a) shows the cluster node number 3,189 of the b-tree leaf node, and Figure 4(b) shows the cluster node number of 3,193. The "03 00" marked as "(1)" shows the object type, and "02 00" marked as "(2)" shows the node's flag. These figures show only 256 bytes at the beginning and 256 bytes at the end of a 4K-sized cluster.



(a) Cluster number=3,189 (b) Cluster number=3,193

Figure 4. Screenshots of B-tree leaf cluster

(a) TOC of a sample cluster=3,189 (b) Inode attributes of "pdfFileLongName-01.pdf"

Figure 5. Screenshots of B-tree node parsing

3.3 Step 2: B-Tree Leaf Parsing

In this step, B-tree key/value information in the cluster is decomposed to analyze the needed data. The structures used to analyze are btree_node_phys_t, btn_index_node_val_t, nloc_t, kvloc_t, and kvoff_t. The B-trees used in APFS are implemented using the btree_node_phys_t structure to represent a node. The same structure is used for all nodes in a tree. Within a node, storage is divided into several areas [3], as followings:

1) information about the node 2) the table of contents, which lists the location of keys and values, 3) storage for the keys and values, and 4) information about the entire tree.

A file-system object stores information about a directory or a file on disk. These objects are stored as several records. The relationship between file-system objects and the records they are made up from is as follows:

For Files:

- APFS_TYPE_INODE
- APFS_TYPE_CRYPTO_STATE
- APFS_TYPE_DSTREAM_ID
- APFS_TYPE_EXTENT
- APFS_TYPE_FILE_EXTENT
- APFS_TYPE_SIBLING_LINK
- APFS_TYPE_XATTR

For Directories

- APFS_TYPE_INODE
- APFS_TYPE_CRYPTO_STATE
- APFS_TYPE_DIR_REC
- APFS_TYPE_DIR_STATS
- APFS_TYPE_XATTR

Among these, the crucial records used in this study are APFS_TYPE_INODE(j_inode_key_t and j_inode_val_t), APFS_TYPE_FILE_EXTENT (j_inode_key_t and j_inode_val_t), APFS_TYPE_XATTR (j_xattr_key_t and j_xattr_val_t.) for files, APFS_TYPE_INODE(j_inode_key_t and j_inode_val_t) and APFS_TYPE_DIREC(j_drec_key_t and j_drec_val_t**)** for directories.

Figure 5(a) shows a list of a TOC(Table of Content) after parsing pairs of key/value in the B-tree node. It can be found entry keys using the TOC information firstly, and then can be found the location containing the entry values. In the case of Figure 5, the entry value of the number 41 of TOC in cluster 3,189 contains information about the inode of the file "pdfFileLongName-01.pdf" as in Figure 5(b).

3.4 Step 3: Data Refining

In this step, duplicate information obtained by decomposing key/value pairs contained in the cluster is removed and data is refined using the associated information. This step eliminates the duplication of data obtained from the record the record type operation of APFS_TYPE_INODE, APFS_TYPE_EXTENT, APFS_TYPE_INODE, and APFS_TYPE_DIREC etc. Due to the operating principle of the APFS file system, many duplicate clusters occur as file operations. A redundant element is obtained in a process of randomly obtaining B-tree information process. Refined data is re-collected with the minimum information necessary for file/directory reconstruction. Figure 6 shows screen shot of collected duplicated data. As the data is obtained from multiple duplicate clusters, i.e., clsNo=0xc75(3,189) and clsNo=0xc79(3,193), it can be seen that the information is distributed and the same information appears repeatedly several times.

3.5 Step 4: Reporting

In this step, a directory tree form is reconstructed based on the refined data obtained from the previous steps. The output includes “File/Directory Info Display” and “Reconstructed Tree Structure Display”. Figure 7(a) shows an example “File/Directory Info Display” collected data reporting from record type operation type 0x09(APFS_TYPE_DIREC) and type 0x03(APFS_TYPE_INODE) of a directory named “testFolder2”, and Figure 7(b) shows collected data from record type operation type 0x08(APFS_TYPE_EXTENT), type 0x09(APFS_TYPE_DIREC) and type 0x03(APFS_TYPE_INODE) of a file named “pdfFileLongName-01.pdf”. And Figure 8(b) shows a typical example of “Reconstructed Tree Structure Display”.

```
dTree[13655]: parentId=0x0e9(233)       privateId=0xee(238)     objID=0x0ee     objType=8       dir/file=8
              physBlockNum=0xaf8        clsNo=0xc79(3193)       size=0x0        nchildren=0     name=pdfFileLongName-03.pdf
dTree[13656]: parentId=0x0e9(233)       privateId=0xeb(235)     objID=0x0eb     objType=8       dir/file=8
              physBlockNum=0x9ca        clsNo=0xc79(3193)       size=0x0        nchildren=0     name=
dTree[13657]: parentId=0x0e9(233)       privateId=0xeb(235)     objID=0x0e9     objType=9       dir/file=8
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0x0        nchildren=0     name=pdfFileLongName-01.pdf
dTree[13658]: parentId=0x0e9(233)       privateId=0xec(236)     objID=0x0ec     objType=8       dir/file=8
              physBlockNum=0xa78        clsNo=0xc79(3193)       size=0x0        nchildren=0     name=pdfFileLongName-04.pdf
dTree[13659]: parentId=0x0e9(233)       privateId=0xec(236)     objID=0x0ec     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc89(3209)       size=0x18ce000  nchildren=1     name=
dTree[13660]: parentId=0x0e9(233)       privateId=0xec(236)     objID=0x0ec     objType=8       dir/file=8
              physBlockNum=0xa78        clsNo=0xc89(3209)       size=0x0        nchildren=0     name=pdfFileLongName-04.pdf
dTree[13661]: parentId=0x0e9(233)       privateId=0xec(236)     objID=0x0e9     objType=9       dir/file=8
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0x0        nchildren=0     name=pdfFileLongName-04.pdf
dTree[13662]: parentId=0x0e9(233)       privateId=0xec(236)     objID=0x0ec     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc79(3193)       size=0x18ce000  nchildren=1     name=
dTree[13663]: parentId=0x0e9(233)       privateId=0xea(234)     objID=0x0ea     objType=8       dir/file=8
              physBlockNum=0x9a3        clsNo=0xc75(3189)       size=0x0        nchildren=0     name=pdfFileLongName-05.pdf
dTree[13664]: parentId=0x0e9(233)       privateId=0xed(237)     objID=0x0e9     objType=9       dir/file=8
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0x0        nchildren=0     name=pdfFileLongName-02.pdf
dTree[13665]: parentId=0x0e9(233)       privateId=0xea(234)     objID=0x0e9     objType=9       dir/file=8
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0x0        nchildren=0     name=pdfFileLongName-05.pdf
dTree[13666]: parentId=0x0e9(233)       privateId=0xed(237)     objID=0x0ed     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc79(3193)       size=0x6612900  nchildren=1     name=
dTree[13667]: parentId=0x0e9(233)       privateId=0xed(237)     objID=0x0ed     objType=8       dir/file=8
              physBlockNum=0xa91        clsNo=0xc79(3193)       size=0x0        nchildren=0     name=pdfFileLongName-02.pdf
dTree[13668]: parentId=0x0e9(233)       privateId=0xed(237)     objID=0x0ed     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc89(3209)       size=0x6612900  nchildren=1     name=
dTree[13669]: parentId=0x0e9(233)       privateId=0xed(237)     objID=0x0ed     objType=8       dir/file=8
              physBlockNum=0xa91        clsNo=0xc89(3209)       size=0x0        nchildren=0     name=pdfFileLongName-02.pdf
dTree[13670]: parentId=0x0e9(233)       privateId=0xeb(235)     objID=0x0eb     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0xad47f00  nchildren=1     name=
dTree[13671]: parentId=0x0e9(233)       privateId=0xea(234)     objID=0x0ea     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0x25a2700  nchildren=1     name=
dTree[13672]: parentId=0x0e9(233)       privateId=0xee(238)     objID=0x0e9     objType=9       dir/file=8
              physBlockNum=0x0  clsNo=0xc75(3189)       size=0x0        nchildren=0     name=pdfFileLongName-03.pdf
dTree[13673]: parentId=0x0e9(233)       privateId=0xee(238)     objID=0x0ee     objType=3       dir/file=0
              physBlockNum=0x0  clsNo=0xc89(3209)       size=0x17a5a100 nchildren=1     name=
```

Figure 6. Screen shot of procedure of file/directory tree reconstruction

```
<<Directory Type Record Info: testFolder2>>
dTree[1053]:
        parentId=  //obj_ID가 parentID역할
        privateId=0xe9(233)
        objID=0x002
        objType=9(APFS_TYPE_DIR_REC)
        dir/file=4(APFS_DT_DIR, dir)
        physBlockNum=0x0
        clsNo=0xcad(3245)
        size=0x0
        nchildren=0
        name=testFolder2
dTree[1054]:
        parentId=0x002(002)
        privateId=0xe9(233)
        objID=0x0e9
        objType=3(APFS_TYPE_INODE)
        dir/file=0
        physBlockNum=0x0
        clsNo=0xc75(3189)
        size=0x0
        nchildren=5
        name=testFolder2
```

(a) Directory: testFolder2

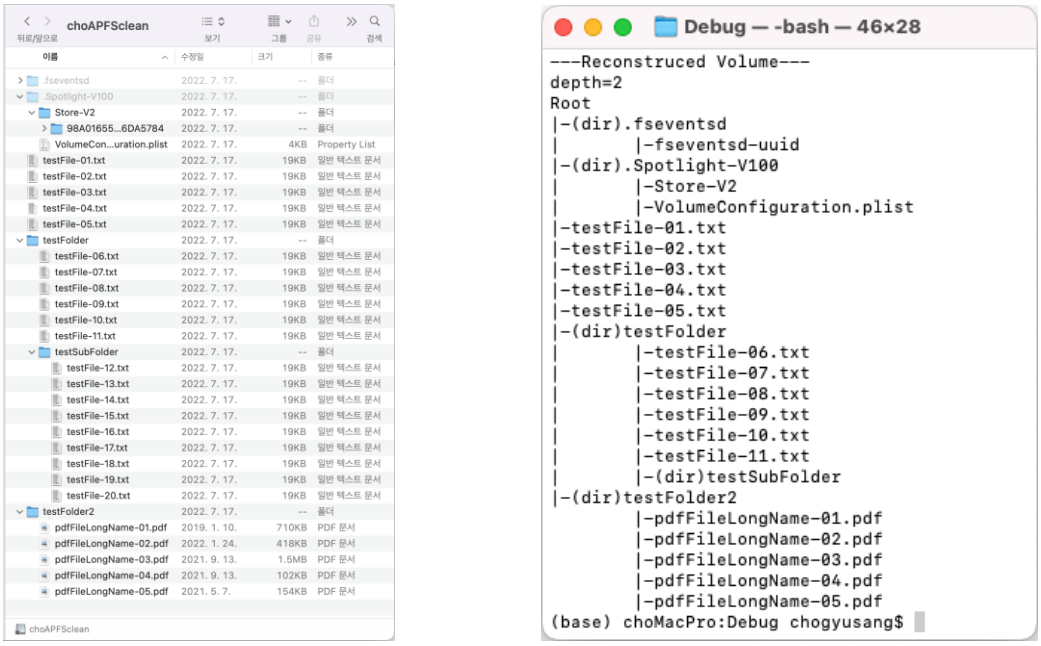
```
<<File Type Record Info: pdfFileLongName-01.pdf>>
dTree[13656]:
        parentId=0x0e9(233)
        privateId=0xeb(235)
        objID=0x0eb
        objType=8(APFS_TYPE_FILE_EXTENT)
        dir/file=8(APFS_DT_REG)
        physBlockNum=0x9ca
        clsNo=0xc79(3193)
        size=0x0
        nchildren=0
        name=

dTree[13657]:
        parentId=   //obj_ID가 parentID역할.
        privateId=0xeb(235)
        objID=0x0e9
        objType=9(APFS_TYPE_DIR_REC)
        dir/file=8(APFS_DT_REG)
        physBlockNum=0x0
        clsNo=0xc75(3189)
        size=0x0
        nchildren=0
        name=pdfFileLongName-01.pdf

dTree[13670]:
        parentId=0x0e9(233)
        privateId=0xeb(235)
        objID=0x0eb
        objType=3(APFS_TYPE_INODE)
        dir/file=0
        physBlockNum=0x0
        clsNo=0xc75(3189)
        size=0xad47f00
        nchildren=0
        name=
```

(b) File: pdfFileLongName-01.pdf

Figure 7. File/directory type record information



(a) Directory View(Finder View) (b) Reconstruction Result

Figure 8. Screen shots of experiment result

4. Experimental Results

4.1 Development Environments

- OS: macOS Monterey version 12.4
- Application Type: macOS command line (terminal) program
- Disk Format: APFS
- Target Storage Device: USB memory stick(532.6MB)
- File Imaging: Raw Format
- Disk Allocation Cluster Size: 4,096 bytes
- Programming Language: C/C++
- Programming Tools: Xcode v13.4.1 (13F100)

The APFS file/directory reconstruction program is designed to run in the command line(terminal) mode in macOS with the C/C++ language using Apple's XCode development tool.

4.2 Experiment of APFS File/Directory Reconstruction

The USB memory stick formatted with the APFS file system was imaged in raw format and exported as a file. User files and directories were created for the experiment; the directory created testFolder, testFolder2, testSubFolder. Five files in the root directory (testFile-01.txt~testFile-05.txt), six files in the testFolder (testFile-06.txt~testFile-11.txt), five files in the testFolder2(testFile-12.txt~testFile-20.txt), and five files in the testFolder2 (pdfFileLongName-01.pdf ~ pdfFileLongName-05.pdf) were created. The file system generated 19 directories and 81 files, including ".fseventsd" and "Store-V2" hidden files, which are automatically generated for a spotlight and a journaling.

Figure 8(a) and Figure 8(b) are displayed to compare the results reconstructed by the proposed method with the case through a general file viewer(Finder Viewer), which Figure 8(a) shows a screenshot of the experiment

disk with the Finder Viewer of macOS, and Figure 8(b) shows a screenshot of the reconstructed file/directory by a proposed method. In both cases, the depth level of the directory was only displayed up to level=2.

5. Conclusions

In this paper, we proposed a method for obtaining digital forensics information from partially damaged disks. This method searches for the B-tree leaf on the entire disk and then acquires the type record information contained therein. Among them, a number of duplicated information, such as the file related type records (APFS_TYPE_INODE, APFS_TYPE_FILE_EXTENT, APFS_TYPE_XATTR etc.), and the directory type records(APFS_TYPE_INODE, APFS_TYPE_DIREC etc.) are refined for the file/directory tree reconstruction. This means that the data necessary for file/directory reconstruction (i.e., parent_id, private_id, name, file_id, xfields, phys_block_num, etc.) is stored in duplicate, so the data is needed to refine by removing redundant elements. In the case of a partially damaged disk, it is important that probability of the reconstruction can be increased due to duplicated redundant information. So, further studies need to find a theoretic clear way to increase the possibility of of file/directory tree reconstruction in cases of partial damage using duplicated redundant data.

References

[1] Apple Developer, “About Apple File System,” *https://developer.apple.com/documentation/foundation/file_system/about_apple_file_system.*
[2] Kurt H. Hansen and Fergus Toolan, “Decoding the apfs file system,” *Digital Investigation*, No. 22, pp. 107–132, 2017.
https://doi.org/10.1016/j.diin.2017.07.003
[3] Apple File System Reference, *https://developer.apple.com/support/downloads/Apple-File-System-Reference.pdf.*
[4] Jonas Plum and Andreas Dewald. “Forensic apfs file recovery,” *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–10, 2018.
[5] G. -S. Cho, “Design and Implementation of APFS Object Identification Tool for Digital Forensics,” International Journal of Internet, Broadcasting and Communication(IJIBC), Vol.14, No.1, 2022.
http://dx.doi.org/10.7236/IJIBC.2022.14.1.x
[6] Simon Gander, APFS FUSE Driver for Linux, *https://github.com/sgan81/apfs-fuse.*
[7] Joachim Metz, libfsapfs, *https://github.com/libyal/libfsapfs*.
[8] Ernesto Fernández, APFS for Linux, *https://github.com/linux-apfs/apfsprogs.*
[9] MacDrive, *https://www.macdrive.com/.*
[10] ParagonTechnologie GmbH, APFS for Windows by Paragon Software. *https://www.paragon-software.com/home/apfs-windows/.*