International Journal of Internet, Broadcasting and Communication Vol.14 No.1 10-18 (2022)
http://dx.doi.org/10.7236/IJIBC.2022.14.1.10

IJIBC 22-1-2

Design and Implementation of APFS Object Identification Tool for Digital Forensics

Gyu-Sang Cho

Professor, Dept. of Computer&Software, Dongyang University, Korea
cho@dyu.ac.kr

Abstract

Since High Sierra, APFS has been used as the main file system. It is a well-established file system that has been used stably thus far. From the perspective of digital forensics, there are still many areas to be investigated. Apple File System Reference is 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. In this paper, we develop APFS object identification tool for digital forensics. The most basic and essential object identification and analysis of the APFS filesystem will be conducted with the tool. The analysis in this study serves as the background for an analysis of the checkpoint operation principle and structure, including the more complex B-tree structure of APFS. There are several options for the developed tool, but the results of two use cases will be shown here. Based on the implemented tool, it is hoped that more functions will be added to make APFS a useful tool for faster and more accurate analyses.

Keywords: *Digital Forensic, File Cluster, Object Header, APFS Filesystem, macOS.*

1. Introduction

The Apple File System is the default file format used on Apple platforms. This file system is the successor to HFS Plus, and some aspects of its design intentionally follow HFS Plus to enable data migration from HFS Plus to the Apple File System. Other aspects of its design address limitations of HFS Plus and enable features such as the cloning of files, snapshots, encryption, and the sharing of free space between volumes [1].

A representative study in the analysis of APFS's file system is the Reference [2], representing the first case of a detailed structural analysis of the APFS file system and serving as valuable data for structural analyses of the APFS file system. Essentially, Apple's Developer Reference provides information about APFS, but it does not provide enough data for an analysis [3]. The literature on the APFS file system to help with forensic analyses was conducted by the authors of Reference [4]. They also proposed different approaches to identify and recover deleted files on an APFS file system. They implemented their approaches as a proof of concept

Manuscript Received: October. 20, 2021 / Revised: October. 24, 2021 / Accepted: October. 27, 2021
Corresponding Author: cho@dyu.ac.kr
Tel: +82-54-630-1119, Fax: +82-54-630-1179
Professor, Dept. of Computer&Software, Dongyang University, Korea

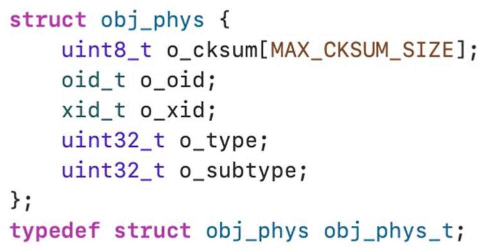
tool and presented AFRO as an open-source implementation of a forensic file recovery tool for APFS to implement all of the presented methodologies [5].

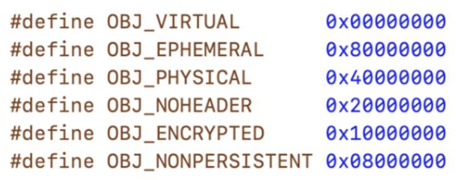
There are also a few open sources that are provided to help comprehend the structure of APFS. The "APFS FUSE Driver for Linux" project is a read-only FUSE driver for the new Apple File System and a collection of libraries to mount, dump and analyze APFS volumes and containers; it also supports software encrypted volumes and fusion drives [6]. The "libfsapfs" project is a library with which to access APFS read-only supported formats with the LGPLv3+ license. This library supports the APFS format features of ZLIB (DEFLATE) compression, LZVN compression, encryption, and extended attributes [7]. The "Apfsprogs" project is a suite of userland software for working with the Apple File System on Linux, but only mkfs and fsck tools are available currently [8]. There are also commercial software packages such as MacDrive that read and write Mac disks from Windows [9], as well as APFS for Windows by Paragon Software Paragon Technologie GmbH [10].

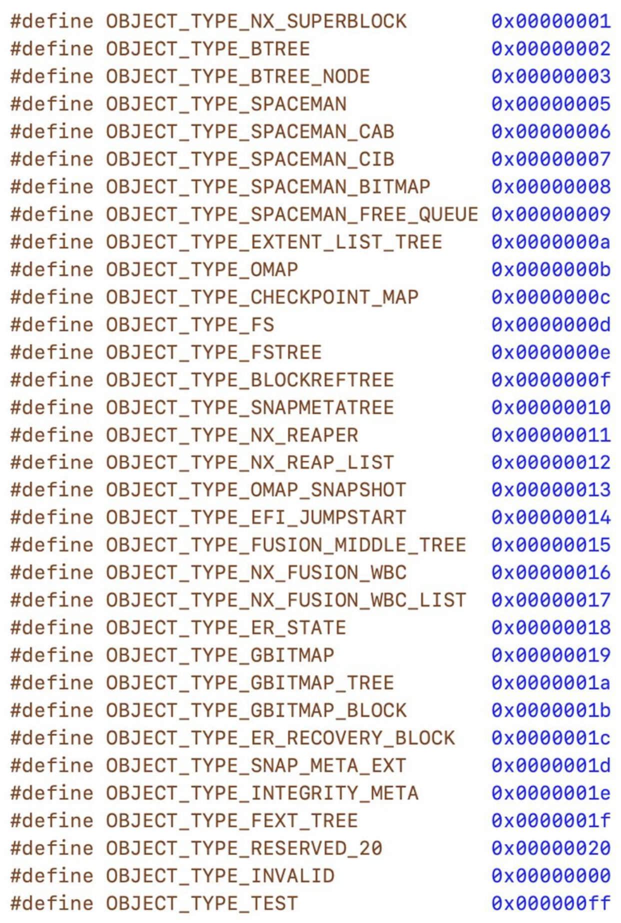
Researchers know more about the structure of APFS than before, but they have not yet fully analyzed its structure to a perfect level. Compared to the NTFS file system, the software provided is still limited. In this paper, the most basic and essential object identification, and analyses in the APFS filesystem will be conducted. The analyses here will become the background of an analysis of the checkpoint operation principle and structure, including the more complex B-tree structure of APFS.

2. Object's data type

Every header used at the beginning of all objects is composed of five members, i.e., o_cksum[], o_oid, o_xid, o_type, and o_subtype. cksum represents the Fletcher 64 checksum of the object, o_oid represents the object's identifier, o_xid represents the identifier of the most recent transaction in which this object was modified. o_type represents the object's type and flags; the object type is a 32-bit value for which the low 16 bits indicate the type and the high 16 bits are object type flag flags using the values listed in the Object Type Flags, i.e., OBJ_VIRTUAL (0x00000000), OBJ_EPHEMERAL (0x80000000), OBJ_PHYSICAL (0x40000000), OBJ_NOHEADER (0x20000000), OBJ_ENCRYPTED (0x10000000), and OBJ_NONPERSISTENT (0x08000000). o_subtype indicates the type of data stored in the data structure, such as a B-tree. For example, a node in a B-tree that contains volume records has the type OBJECT_TYPE_BTREE_NODE and subtype OBJECT_TYPE_FS.


```
struct obj_phys {
    uint8_t o_cksum[MAX_CKSUM_SIZE];
    oid_t o_oid;
    xid_t o_xid;
    uint32_t o_type;
    uint32_t o_subtype;
};
typedef struct obj_phys obj_phys_t;
```
(a) Structure of obj_phys_t


```
#define OBJ_VIRTUAL       0x00000000
#define OBJ_EPHEMERAL     0x80000000
#define OBJ_PHYSICAL      0x40000000
#define OBJ_NOHEADER      0x20000000
#define OBJ_ENCRYPTED     0x10000000
#define OBJ_NONPERSISTENT 0x08000000
```
(b) Object type flags



```
#define OBJECT_TYPE_NX_SUPERBLOCK       0x00000001
#define OBJECT_TYPE_BTREE               0x00000002
#define OBJECT_TYPE_BTREE_NODE          0x00000003
#define OBJECT_TYPE_SPACEMAN            0x00000005
#define OBJECT_TYPE_SPACEMAN_CAB        0x00000006
#define OBJECT_TYPE_SPACEMAN_CIB        0x00000007
#define OBJECT_TYPE_SPACEMAN_BITMAP     0x00000008
#define OBJECT_TYPE_SPACEMAN_FREE_QUEUE 0x00000009
#define OBJECT_TYPE_EXTENT_LIST_TREE    0x0000000a
#define OBJECT_TYPE_OMAP                0x0000000b
#define OBJECT_TYPE_CHECKPOINT_MAP      0x0000000c
#define OBJECT_TYPE_FS                  0x0000000d
#define OBJECT_TYPE_FSTREE              0x0000000e
#define OBJECT_TYPE_BLOCKREFTREE        0x0000000f
#define OBJECT_TYPE_SNAPMETATREE        0x00000010
#define OBJECT_TYPE_NX_REAPER           0x00000011
#define OBJECT_TYPE_NX_REAP_LIST        0x00000012
#define OBJECT_TYPE_OMAP_SNAPSHOT       0x00000013
#define OBJECT_TYPE_EFI_JUMPSTART       0x00000014
#define OBJECT_TYPE_FUSION_MIDDLE_TREE  0x00000015
#define OBJECT_TYPE_NX_FUSION_WBC       0x00000016
#define OBJECT_TYPE_NX_FUSION_WBC_LIST  0x00000017
#define OBJECT_TYPE_ER_STATE            0x00000018
#define OBJECT_TYPE_GBITMAP             0x00000019
#define OBJECT_TYPE_GBITMAP_TREE        0x0000001a
#define OBJECT_TYPE_GBITMAP_BLOCK       0x0000001b
#define OBJECT_TYPE_ER_RECOVERY_BLOCK   0x0000001c
#define OBJECT_TYPE_SNAP_META_EXT       0x0000001d
#define OBJECT_TYPE_INTEGRITY_META      0x0000001e
#define OBJECT_TYPE_FEXT_TREE           0x0000001f
#define OBJECT_TYPE_RESERVED_20         0x00000020
#define OBJECT_TYPE_INVALID             0x00000000
#define OBJECT_TYPE_TEST                0x000000ff
```

(c) Object flags

Figure 1. APFS object's header structure and object types & flags [4]

Every header used at the beginning of all objects is composed of five members, i.e., o_cksum[], o_oid, o_xid, o_type, and o_subtype. cksum represents the Fletcher 64 checksum of the object, o_oid represents the object's identifier, o_xid represents the identifier of the most recent transaction in which this object was modified. o_type represents the object's type and flags; the object type is a 32-bit value for which the low 16 bits indicate the type and the high 16 bits are object type flags using the values listed in the Object Type Flags. o_subtype indicates the type of data stored in the data structure. The same values used for o_type are used here as well.

2.1 Object Types

Values used as types and subtypes according to the object header's obj_phys_t structure (Figure 1 (a)) are as follows. OBJECT_TYPE_NX_SUPERBLOCK is used for a container superblock with the structure of nx_superblock_t. OBJECT_TYPE_BTREE and OBJECT_TYPE_BTREE_NODE are used for the B-tree root and node with the structure of tree_node_phys_t in both cases. OBJECT_TYPE_SPACEMAN is used as a space manager with the structure of spaceman_phys_t. OBJECT_TYPE_SPACEMAN_CAB is used for a chunk-info address block with the structure of cib_addr_block and OBJECT_TYPE_SPACEMAN_CIB is used for a chunk-info block with the structure of cib_addr_block. Both are used by the space manager. OBJECT_TYPE_SPACEMAN_BITMAP is used for a free-space and bitmap by the space manager.

OBJECT_TYPE_SPACEMAN_FREE_QUEUE is used for a free-space queue by the space manager; it is a mapping from spaceman_free_queue_key_t to spaceman_free_queue_t.

OBJECT_TYPE_EXTENT_LIST_TREE is used for an extents-list tree, which is a mapping from paddr_t to prange_t, in which the keys indicate an offset of the logical start of the extent and the value denotes the physical location of the stored data. OBJECT_TYPE_OMAP is used for an object map as a type with the structure of omap_phys_t and a tree that stores the records of an object map mapping from omap_key_t to omap_val_t as a subtype. OBJECT_TYPE_CHECKPOINT_MAP is used for a checkpoint map with the structure of checkpoint_map_phys_t. OBJECT_TYPE_FS is used for a volume with the structure of apfs_superblock_t. OBJECT_TYPE_FSTREE is used for a tree containing file-system records, in which the keys and values stored in the tree vary. Each key begins with a type of j_key_t which indicates the type of that key and value.

OBJECT_TYPE_BLOCKREFTREE is for a tree containing extent references, mapping from j_phys_ext_key_t to j_phys_ext_val_t. OBJECT_TYPE_SNAPMETATREE is for a tree containing snapshot metadata for a volume. OBJECT_TYPE_NX_REAPER is for a reaper with the structure of nx_reaper_phys_t. OBJECT_TYPE_NX_REAP_LIST is for a reaper list with the structure of nx_reap_list_phys_t. OBJECT_TYPE_OMAP_SNAPSHOT is for a tree containing information about snapshots of an object map, with mapping from xid_t to omap_snapshot_t. OBJECT_TYPE_EFI_JUMPSTART is for the EFI information used for booting. OBJECT_TYPE_FUSION_MIDDLE_TREE, and OBJECT_TYPE_NX_FUSION_WBC are used for a tree used with a fusion device.

OBJECT_TYPE_ER_STATE is for an encryption-rolling state. OBJECT_TYPE_GBITMAP, OBJECT_TYPE_GBITMAP_TREE, and OBJECT_TYPE_GBITMAP_BLOCK are for a general-purpose bitmap with the structure of gbitmap_phys_t. OBJECT_TYPE_ER_RECOVERY_BLOCK is used to recover from a system crash. OBJECT_TYPE_SNAP_META_EXT is for additional metadata about snapshots. OBJECT_TYPE_INTEGRITY_META is for an integrity metadata object. OBJECT_TYPE_FEXT_TREE is for a B-tree of file extents. OBJECT_TYPE_RESERVED_20, OBJECT_TYPE_INVALID, and OBJECT_TYPE_TEST are not currently used [3].

2.2 Object Flags

Object flags (Figure 1 (b)) are used to provide additional information in the form of an o_type of the object header representing the object's type and flags, specifically the high 16 bits of the 32-bit object type. The value is extracted from the object type by OBJECT_TYPE_FLAGS_MASK(0xffff0000).

There are three methods of storing objects at the container level depending on the object, i.e., ephemeral objects, physical objects, and virtual objects. Ephemeral objects are stored in memory, physical objects are stored at a specific block location on the disk, and virtual objects are stored on the disk at a block location looked up using an object map. OBJ_VIRTUAL is for a virtual object, OBJ_EPHEMERAL is for an ephemeral object, OBJ_PHYSICAL is for a physical object, OBJ_NOHEADER is for an object stored without an obj_phys_t header, OBJ_ENCRYPTED is for an encrypted object, and OBJ_NONPERSISTENT is for an ephemeral object that does not persist across the unmounting process [3].

2.3 Object subtype

The object's subtype uses the uint32_t o_subtype of the obj_phys_t structure. The subtype indicates the type of data stored in the data structure. For a B-tree, a node of a B-tree that contains volume records is of the

OBJECT_TYPE_BTREE_NODE type and the OBJECT_TYPE_FS subtype. The values of the subtype used are identical to those used with "Object Types."

With regard to "Object Types," some are used only for a subtype. Examples include OBJECT_TYPE_SPACEMAN_FREE QUEUE, OBJECT_TYPE_EXTENT_LIST_TREE, OBJECT_TYPE_FSTREE, OBJECT_TYPE_BLOCK REFTREE, OBJECT_TYPE_SNAPMETATREE, OBJECT_TYPE_FUSION_MIDDLE_TREE, OBJECT_TYPE_GBITMAP_TREE, and OBJECT_TYPE_FEXT_TREE [3].

2.4 APFS Overall Configuration

Figure 2 shows the overall diagram of the APFS composition. When APFS parses the filesystem, the first central structure is the "Container Superblock," which contains pointers to the "Checkpoint Descriptor" and "Checkpoint Data." The "Checkpoint Descriptor" contains the "Checkpoint" itself and copies of older "Container Superblocks." The "Checkpoint Data" contains the "Space Manager," "History Blocks," and "Reaper," which show the data of the entire container at a given point. The "Space Manager" points to the "Space Manager Internal Pool," which points to the "Bitmap" in turn. The first "Container Superblock" stores a reference to an "Object Map (OMAP)" B-tree that points to its "OMAP Root Node." This B-tree contains 'OMAP Entries," which link object IDs to block offsets. The "Container Superblock" stores pointers to the "Volume Superblocks" of all volumes in the filesystem. Each "Volume Superblock" points to the "Extentref Tree," its own 'OMAP," and a "Root Directory Node" for the particular volume [4].

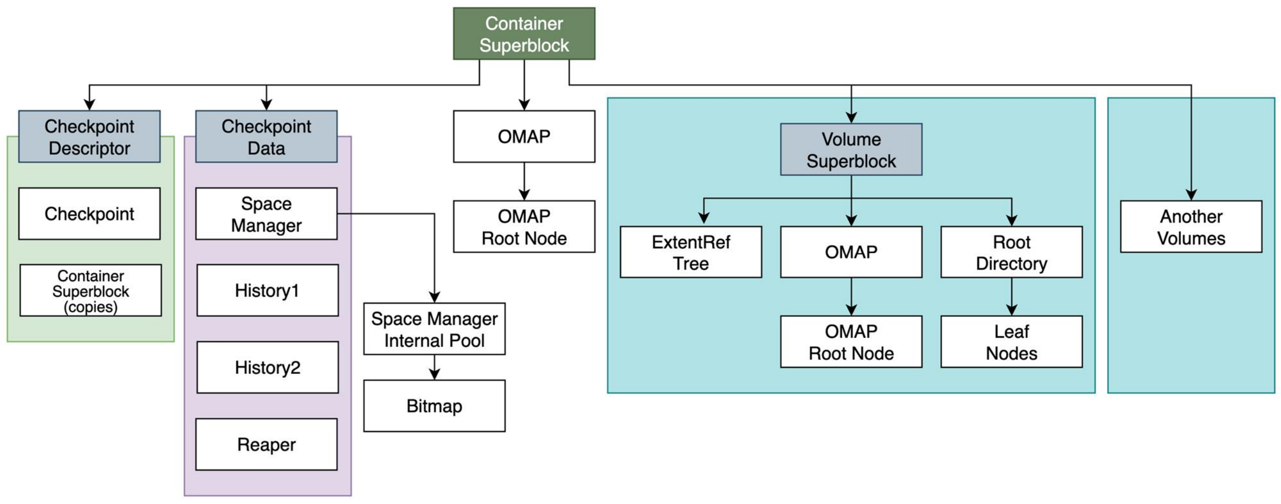


Figure 2. APFS Overall configuration

3. Configuration of the APFS Object Identification Tool

3.1 Overview

The APFS Object Identification tool has four features, as depicted in Figure 3. "Object Header Parsing" is a fundamental feature which parses the object header's obj_phys_t structure as attributes, i.e., o_cksum[], o_oid, o_xid, o_type, and o_subtype, as explained in the previous section. “Searching Object Identifier” is feature searching a target object identifier with the options of an object’s cluster number, the object’s type, and the object’s subtype. The findings are listing in a command line (terminal) window as a view. The “Entire Disk Searching” feature denotes a run-through of an entire disk cluster from cluster #0 to cluster #MAX_NO, which is implemented when searching for an object’s clusters that have been erased, hidden, or corrupted. “Object Tree Listing” is feature that lists selected items from the results, e.g., listing only an object’s id number; listing an object’s id number and a cluster number; listing an object’s type and subtype, the object’s statistics, and the object’s header and cluster contents, for instance.

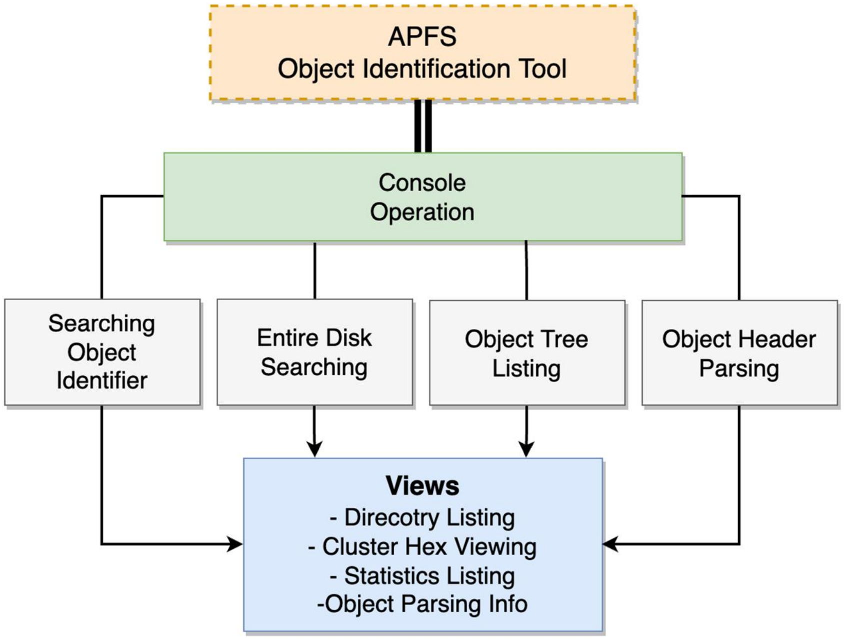


Figure 3. Features of the APFS Object Identification Tool

3.2 Development Environments

- OS: macOS Big Sur version 11.6.1
- Application Type: macOS command line (terminal) program
- Disk Format: APFS
- Target Storage Device: USB memory(16GB)
- Disk Allocation Cluster Size: 4,096 bytes
- Programming Language: C/C++
- Programming Tools: Xcode v13.0(13A233)

The APFS Object Identification tool is designed to run in the command line(terminal) mode in macOS with the C/C++ language using Apple’s XCode environment.

3.3 Command Line Options

There are three categories of command line options for the tool. These categories can be used alone or in combination.

- Options for object's attribute

"-oid"	: to set the Object ID,
"-xid"	: to set the Transaction ID
"-type"	: to set the low 16 bits indicating the type using the values listed in Object Types
"-typeFlag"	: to set the high 16 bits that are flags using the values listed in Object Type Flags
"-stype"	: to set the subtype indicating the type using the values listed in Object Types

- Options for cluster numbers

"-all"	: to set entire cluster searching
"-cls"	: to set a cluster number

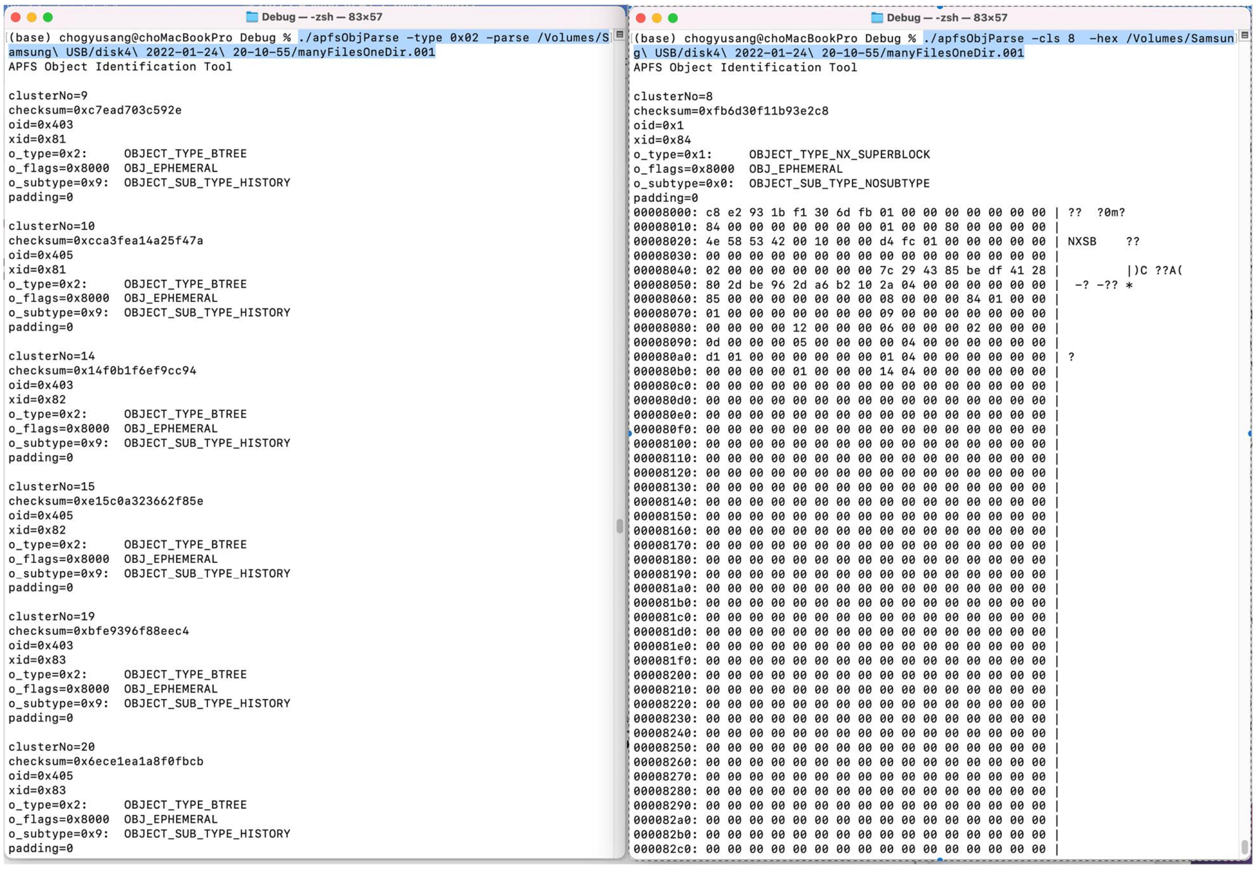
- Options for views

"-dir"	: an option to list directory type
"-hex"	: an option to list hexa-data of a cluster
"-stat"	: statistics of an object's option attributes
"-parse"	: to parse of an object's attributes using the cluster number(default).

3.4 Experimental Results

Only two experiments were performed selectively among various possible experiments. Experiment #1 assigned the command line option as "-type 0x02 -parse" and the directory name and file name. "-type 0x02" is set to find the B-tree root node (OBJECT_TYPE_BTREE, 0x02). For the 0x02 option, numerous objects are found when searching for objects, as shown in Figure 4(a). Given the many purposes of B-trees used in APFS, copies left along with many objects were also found.

Experiment #2 assigned the command line option as "-cls 8 -hex" and the directory name and file name. "-cls" is used for setting the cluster number, and "-hex" is used to list the contents of a cluster in the form of hexa-data. In this case, "-cls 8" represents a container superblock. This block has the signature "NXSB" at offset 0x20, as shown in Figure 4 (b).



```
(base) chogyusang@choMacBookPro Debug % ./apfsObjParse -type 0x02 -parse /Volumes/S
amsung\ USB/disk4\ 2022-01-24\ 20-10-55/manyFilesOneDir.001
APFS Object Identification Tool

clusterNo=9
checksum=0xc7ead703c592e
oid=0x403
xid=0x81
o_type=0x2:      OBJECT_TYPE_BTREE
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x9:  OBJECT_SUB_TYPE_HISTORY
padding=0

clusterNo=10
checksum=0xcca3fea14a25f47a
oid=0x405
xid=0x81
o_type=0x2:      OBJECT_TYPE_BTREE
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x9:  OBJECT_SUB_TYPE_HISTORY
padding=0

clusterNo=14
checksum=0x14f0b1f6ef9cc94
oid=0x403
xid=0x82
o_type=0x2:      OBJECT_TYPE_BTREE
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x9:  OBJECT_SUB_TYPE_HISTORY
padding=0

clusterNo=15
checksum=0xe15c0a323662f85e
oid=0x405
xid=0x82
o_type=0x2:      OBJECT_TYPE_BTREE
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x9:  OBJECT_SUB_TYPE_HISTORY
padding=0

clusterNo=19
checksum=0xbfe9396f88eec4
oid=0x403
xid=0x83
o_type=0x2:      OBJECT_TYPE_BTREE
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x9:  OBJECT_SUB_TYPE_HISTORY
padding=0

clusterNo=20
checksum=0x6ece1ea1a8f0fbcb
oid=0x405
xid=0x83
o_type=0x2:      OBJECT_TYPE_BTREE
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x9:  OBJECT_SUB_TYPE_HISTORY
padding=0
```

(a) Experiment #1

```
(base) chogyusang@choMacBookPro Debug % ./apfsObjParse -cls 8  -hex /Volumes/Samsun
g\ USB/disk4\ 2022-01-24\ 20-10-55/manyFilesOneDir.001
APFS Object Identification Tool

clusterNo=8
checksum=0xfb6d30f11b93e2c8
oid=0x1
xid=0x84
o_type=0x1:      OBJECT_TYPE_NX_SUPERBLOCK
o_flags=0x8000  OBJ_EPHEMERAL
o_subtype=0x0:  OBJECT_SUB_TYPE_NOSUBTYPE
padding=0
00008000: c8 e2 93 1b f1 30 6d fb 01 00 00 00 00 00 00 00 | ??  ?0m?
00008010: 84 00 00 00 00 00 00 00 01 00 00 80 00 00 00 00 |
00008020: 4e 58 53 42 00 10 00 00 d4 fc 01 00 00 00 00 00 | NXSB    ??
00008030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008040: 02 00 00 00 00 00 00 00 7c 29 43 85 be df 41 28 |         |)C ??A(
00008050: 80 2d be 96 2d a6 b2 10 2a 04 00 00 00 00 00 00 |  -? -?? *
00008060: 85 00 00 00 00 00 00 00 08 00 00 00 84 01 00 00 |
00008070: 01 00 00 00 00 00 00 00 09 00 00 00 00 00 00 00 |
00008080: 00 00 00 00 12 00 00 00 06 00 00 00 02 00 00 00 |
00008090: 0d 00 00 00 05 00 00 00 00 04 00 00 00 00 00 00 |
000080a0: d1 01 00 00 00 00 00 00 01 04 00 00 00 00 00 00 | ?
000080b0: 00 00 00 00 01 00 00 00 14 04 00 00 00 00 00 00 |
000080c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000080d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000080e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000080f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000081a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000081b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000081c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000081d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000081e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000081f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00008290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000082a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000082b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
000082c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
```

(b) Experiment #2

Figure 4. Experiments with the APFS Object Identification Tool

5. Conclusions

In this paper, we designed and implemented an APFS Object Identification tool for a forensics analysis and for education. Apple File System Reference is provided to the Apple developer site [1, 11], but it is not satisfactory for use by developers. Nevertheless, several developers released the results of analyses of parts that were not disclosed as documents as part of the development of the program [2-8]. These sources are very helpful for a digital forensic analysis. In this paper, a tiny tool for analyzing APFS objects is developed. Despite the fact that it has minor functions, it is the most basic and essential object identification and analysis tool developed thus far. There are many options for the developed tool, but the results of two use cases were shown here. Based on the implemented tool, it is hoped that more functions will be added to make APFS a useful tool for rapid and accurate analyses.

Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF- 2019R1F1A1058902)

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] A. Dewald and J. Plum. APFS INTERNALS FOR FORENSIC ANALYSIS, 2018.
https://static.ernw.de/whitepaper/ERNW_Whitepaper65_APFS-forensics_signed.pdf

[5] Jonas Plum and Andreas Dewald. Forensic apfs file recovery. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–10, 2018

[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/*

[11] Apple File System Reference, *https://developer.apple.com/support/downloads/Apple-File-System-Reference.pdf*