International Journal of Internet, Broadcasting and Communication Vol.13 No.3 178-192 (2021)
http://dx.doi.org/10.7236/IJIBC.2021.13.3.178

IJIBC 21-3-23

Development of a Forensic Analyzing Tool based on Cluster Information of HFS+ filesystem

Gyu-Sang Cho*

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

Abstract

File system forensics typically focus on the contents or timestamps of a file, and it is common to work around file/directory centers. But to recover a deleted file on the disk or use a carving technique to find and connect partial missing content, the evidence must be analyzed using cluster-centered analysis. Forensics tools such as EnCase, TSK, and X-ways, provide a basic ability to get information about disk clusters, but these are not the core functions of the tools. Alternatively, Sysinternals' DiskView tool provides a more intuitive visualization function, which makes it easier to obtain information around disk clusters. In addition, most current tools are for Windows. There are very few forensic analysis tools for MacOS, and furthermore, cluster analysis tools are very rare.

In this paper, we developed a tool named FACT (Forensic Analyzer based Cluster Information Tool) for analyzing the state of clusters in a HFS+ file system, for digital forensics. The FACT consists of three features, a Cluster based analysis, B-tree based analysis, and Directory based analysis. The *Cluster based analysis is the main feature, and was basically developed for cluster analysis. The FACT tool's cluster visualization feature plays a central role. The FACT tool was programmed in two programming languages, C/C++ and Python. The core part for analyzing the HFS+ filesystem was programmed in C/C++ and the visualization part is implemented using the Python Tkinter library. The features in this study will evolve into key forensics tools for use in MacOS, and by providing additional GUI capabilities can be very important for cluster-centric forensics analysis.*

Keywords: *HFS+ filesystem, Cluster Analysis, Forensic Tool, Digital Forensics, B-tree structure.*

1. Introduction

A cluster is the smallest logical amount of disk space that can be allocated to hold a file or a directory. To reduce the overhead of managing on-disk data structures, the filesystem allocates data in contiguous groups of sectors, called clusters. Usually, a cluster has 8 contiguous sectors, i.e. 4K(4,096 bytes) [1].

If you directly access the raw disk without using the operating system to obtain the information contained in the cluster, all of the information for the forensic investigation will be interpreted based on the cluster information. Many forensic tools provide a function to display disk cluster information, but do not provide a satisfactory function to utilize it. Alternatively, DiskView provides a much more intuitive and visual management function. This tool is generally used as a system management tool, but is also very useful as a

Manuscript Received: August. 2, 2021 / Revised: August. 5, 2021 / Accepted: August. 9, 2021
Corresponding Author: cho@dyu.ac.kr
Tel: +82-54-630-1119, Fax: +82-54-630-1179
Professor, Department of Computer Software, Dongyang University, Korea

forensic analysis tool because it allows you to visually understand information about files/directories on the disk by displaying the fragmentation status of the hard disk or the location of the specified file [2].

Sysinternals' DiskView displays a graphical map of your disk, allowing you to determine where a file is located or, by clicking on a cluster, seeing which file occupies it. Double-clicking gets more information about a file to which a cluster is allocated [2]. However, DiskView is designed for disk management, not digital forensics, so it has practical limitations when it comes to obtaining complexly connected digital forensic-related information. In addition, it was developed only for the NTFS file system, which makes it necessary to develop a tool with the same function for HFS+, APFS, and Ext3/4. Even though similar features are provided in existing forensic tools, it is difficult to find tools with specialized functions for cluster analysis.

There have been several studies that have performed analysis of disk clusters. Hargreaves provides a discussion of the development of a prototype visualization tool that could be used for examining application or operating system files that themselves contain allocated and unallocated blocks, and demonstrates how a visualization could assist in identifying areas that are unallocated and therefore may contain deleted data of interest [3].

M. Karresand et. al. insisted on an approach that builds on the principle of searching, where it is more probable to find what you are looking for. So, they studied the behavior of the cluster allocation algorithm in the NTFS filesystem to see where new data is actually placed on the disk [1]. They showed that data were more frequently allocated closer to the middle of the disk, so that area should be getting higher attention during a digital forensic investigation of a NTFS formatted hard disk [4].

Burghardt and Feldman described a method for identifying and extracting the residual contents of deleted files in an HFS+ file system. Their research were based on the premise that records of file I/O operations is maintained in a journal on HFS+ file systems, and the record could be used to reconstruct recent deletions of files. The method is effective even if the allocation blocks are separated into multiple fragments, but, subsequent studies revealed that there were disadvantages to their studies and presented complementary techniques [5].

Bang et.al. argued that existing Burghardt and Feldman's research and analysis result [5] has a drawback. The research recovers the deleted file by metadata that is maintained in a journal on HFS+ file system, however the technique excludes specific files, so the problem needs to be reformed. They suggested an algorithm that analyzed a journal in the HFS+ file system in detail. And they demonstrated that the deleted file could be recovered from the extracted metadata using this algorithm without the excluded file [6].

Cho proposed a method to manipulate the fragmentation of disks by arbitrarily allocating and releasing the status of a disk cluster in the NTFS file system. The method allowed experiments to be performed in several studies related to fragmentation problems on disk clusters. The author indicated the importance of having a cluster analysis tool to show and analyze the results of experiments, including a test on the performance of disk defragmentation tools according to the state of fragmentation, the experimental environment for fragmented file carving methods for digital forensics, the setup of cluster fragmentation for testing the robustness of data hiding methods within directory indexes, and a test for the file system's disk allocation methods for the various versions of Windows [7].

In this study, a tool named FACT (**F**orensic **A**nalyzer based **C**luster Information **T**ool) was developed for analyzing the state of clusters in the HFS+ file system, and for use in digital forensics. The tool displays comprehensive information about files and directories. That is, a function is provided to display a unique number for a file/directory, the total number of clusters, the start location of the cluster, timestamp information of the file/directory, etc. In addition, information related to the structure of the directory and the cluster is provided with a visualization. This tool employs the HFS+ file system analysis in C/C++, and the user interface

is graphically provided using the Tkinter library written in Python language. These programs were developed to work in the macOS environment.

Section 2 of this paper describes the basic structure of the HFS+ filesystem and the structure of the catalog B-tree. Section 3 describes the main features of FACT, and Section 4 describes the purposes and functions of the library produced to implement FACT, and the paper concludes in Section 5.

2. Retrieval of Cluster Information in the HFS+ filesystem

2.1 HFS+ Filesystem Basic

HFS+ uses a number of structures to manage the data organization on the volume. These structures include the volume header, the catalog file, the extents overflow file, the attributes file, the allocation file (bitmap), and the startup file. The volume header, which every HFS Plus volume must have, contains file system attributes, such as the version and the allocation block size, and information to locate the metadata files. And the volume header contains information about the date and time of the volume's creation and the number of files on the volume, as well as the location of the other key structures on the volume [8].

The volume header is always located at 1024 bytes from the start of the volume. A copy of the volume header, i.e., the alternate volume header, is stored starting at 1024 bytes before the end of the volume. The first 1024 bytes space in the volume is a reserved area, and the last 512 bytes of the volume after the alternate volume header are also reserved. All of the allocation blocks containing the volume header, alternate volume header, the reserved areas before the volume header, and after the alternate volume header, are marked as used in the allocation file [9].

HFS Plus has five special files, which store the file system structures required to access folders, user files, and attributes. The special files are the catalog file, the extents overflow file, the allocation file, the attributes file and the startup file. Special files only have a data fork, and the extents of that fork are described in the volume header [8].

The allocation file has bitmap information which tracks the allocation status of each block in the volume. The catalog file contains metadata for each file and a directory on the volume. The extents overflow file has cluster location, and run-length information for forks that have more than eight extents allocated to them. The attributes file is a special file which contains additional data for a file or a directory. The startup file contains information used for booting the system [10]. Figure 1 shows the organization of the HFS volumes, where the five special files depict the order and a location in a volume. Among these special files, the important files with cluster-related information are the bitmap file and the catalog file. Above all, it is very important to analyze catalog files because the catalog files contain a lot of meta-information about the files/directories and the clusters in which they are stored.

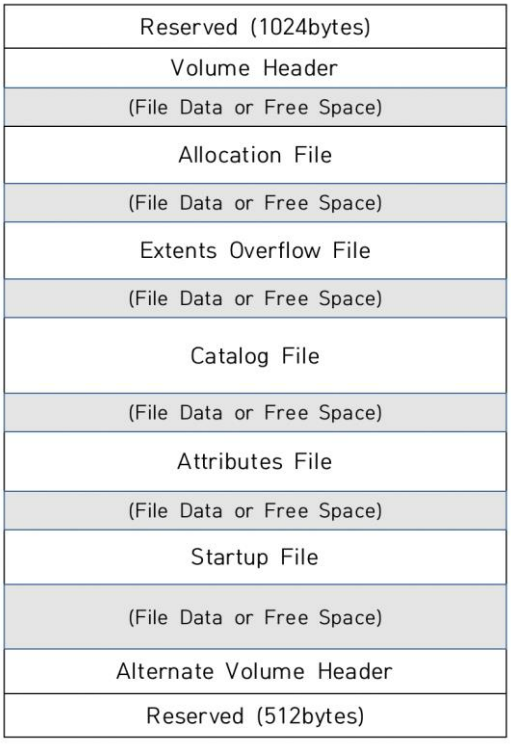

Figure 1. Organization of HFS Plus Volumes[9]

2.2 Catalog File

The location of the first block of the catalog file, i.e., the head node of the file, is stored in the volume header file. The catalog file contains the metadata of all the files and folders in a volume, including creation, modification and access date, permissions, file identifier, and information about the user who created the file [9].

HFS+ maintains a list of the fork's extents that tracks allocation blocks belonging to a fork. An extent is a contiguous range of allocation blocks allocated to some fork by a pair of numbers, in which the first allocation block number and the number of allocation blocks are as shown in Figure 3. The first eight extents of each fork are stored in a user catalog file, and any additional extents are stored in the extents overflow file [11].

The catalog file is used to maintain information about the hierarchy of files and folders on HFS+. A catalog file is organized as a B-Tree and hence consists of a head node, index nodes, leaf nodes, and map nodes. The first location of the extent in the catalog file is stored in the volume header. The node number of the root node of the B-tree can be obtained from the catalog file's header node. The node size of the catalog file must be at least 4 KB, but the actual implemented size of one HFS+ catalog record is 8K bytes [9].

Each file or folder in the catalog file is assigned a unique CNID (Catalog Node ID). The CNID is the folder ID for directory and the file ID for file, respectively. A parent ID is assigned to the CNID of the folder containing the file or folder for any given file or folder.

The catalog file key consists of the parent folder's CNID and the name of the file or folder using the HFSPlusCatalogKey structure type, as in Figure 2(d). A catalog file leaf node can contain four different types of data records, i.e., a folder record, a file record, a folder thread record, and a file thread record. The folder and the file thread record are used to map the file or folder ID to the actual parent directory ID and name. The

catalog folder record is used to hold information about a folder on the volume. The HFSPlusCatalogFolder structure type, as shown in Figure 2(b), is used to hold information about a directory on the volume. The HFSPlusCatalogFile structure type in Figure 2(a) is used to store file data. The HFSPlusCatalogThread structure type in Figure 2(c) for the catalog thread record is used to link a CNID to a file or directory record using that CNID [9].

The B-trees structure is used for the catalog, extents overflow, and attributes files. A B-tree is stored in a file data fork. Each B-tree has a HFSPlusForkData structure, as in Figure 3, in the volume header that describes the size and initial extents of that data fork.

There are four kinds of B-tree in the HFS+, and each B-tree contains a single header node. The header node is always the first node in the B-tree. It contains the information needed to find other any other node in the tree. Map nodes contain map records, which hold any allocation bitmap data that describes the free nodes in the B-tree, that overflow the map record in the header node. The index nodes have a pointer to link sub-nodes that determine the structure of the B-tree. Leaf nodes contain several data records that represent the data associated with a given key, and the key for each data record is unique [11].

```
struct HFSPlusCatalogFile {
    int16_t          recordType;
    uint16_t         flags;
    uint32_t         reserved1;
    uint32_t         fileID;
    uint32_t         createDate;
    uint32_t         contentModDate;
    uint32_t         attributeModDate;
    uint32_t         accessDate;
    uint32_t         backupDate;
    HFSPlusBSDInfo   bsdInfo;
    FndrFileInfo     userInfo;
    FndrOpaqueInfo   finderInfo;
    uint32_t         textEncoding;
    uint32_t         foundForkFlag;
    HFSPlusForkData  dataFork;
    HFSPlusForkData  resourceFork;
};
```

(a)

```
struct HFSPlusCatalogFolder {
    int16_t         recordType
    uint16_t        flags;
    uint32_t        valence;
    uint32_t        folderID;
    uint32_t        createDate;
    uint32_t        contentModDate;
    uint32_t        attributeModDate;
    uint32_t        accessDate;
    uint32_t        backupDate;
    HFSPlusBSDInfo  bsdInfo;
    FndrDirInfo     userInfo;
    FndrOpaqueInfo  finderInfo;
    uint32_t        textEncoding;
    uint32_t        folderCount;
};
```

(b)

```
struct HFSPlusCatalogThread {
    int16_t        recordType;
    int16_t        reserved;
    uint32_t       parentID;
    HFSUniStr255   nodeName;
};
```

(c)

```
struct HFSPlusCatalogKey {
    uint16_t             keyLength;
    HFSCatalogNodeID     parentID;
    HFSUniStr255         nodeName;
};
```

(d)

Figure 2. Structure of a HFS+ Catalog Related Structure [9]
(a) HFSPlusCataloffFile (b) HFSPlusCataloffFolder
(c) HFSPlusCatalofThread (d) HFSPlusCatlogKey

```
struct HFSPlusForkData {
    uint64_t     logicalSize
    uint32_t     clumpSize;
    uint32_t     totalBlocks
    HFSPlusExtentRecord extents[8];
};

struct HFSPlusExtentRecord {
    uint32_t    startBlock;
    uint32_t    blockCount;
};
```

Figure 3. Structure of a HFS+ Fork Data and Extent Record [9]

2.3 Volume Header

The HFS+ volume contains a volume header 1024 bytes offset from the start of the volume. The volume header contains information about the whole partition, which makes it an HFS+ volume. Given the importance of the volume header information, an identical copy is kept at the end of the partition, which is known as the alternate volume header. It is stored starting 1024 bytes before the end of the volume. The alternate volume header is intended solely for use in disk repair. The volume header is updated when the length or location is changed in the special files [12].

Figure 4 shows the HFS+ volume header structure of the HFSPlusVolumeHeader type, which includes a volume header signature, attributes for the entire volume, the location of a journal file, its date of creation, modificatio, backup, checked and so on. There is also some summary information for the volume, such as the total number of files and folders on it, and the number of unused allocation blocks. There is also location and size information about the special files containing file system information, such as allocation file, extents file, catalog file, attribute file, startup file [11].

```
struct HFSPlusVolumeHeader {
   uint16_t    signature
   uint16_t    version;
   uint32_t    attributes;
   uint32_t    lastMountedVersion;
   uint32_t    journalInfoBlock;

   uint32_t    createDate;
   uint32_t    modifyDate;
   uint32_t    backupDate;
   uint32_t    checkedDate;

   uint32_t    fileCount;
   uint32_t    folderCount;

   uint32_t    blockSize;
   uint32_t    totalBlocks;
   uint32_t    freeBlocks;
   uint32_t    nextAllocation;
   uint32_t    rsrcClumpSize;
   uint32_t    dataClumpSize;
   uint32_t    nextCatalogID;
   uint32_t    writeCount;
   uint64_t    encodingsBitmap;

   uint8_t    finderInfo[32];

   HFSPlusForkData  allocationFile;
   HFSPlusForkData  extentsFile;
   HFSPlusForkData  catalogFile;
   HFSPlusForkData  attributesFile;
   HFSPlusForkData  startupFile;
};
```

Figure 4. Volume Header Structure[9]

2.4 Allocation File

Information about the current allocated cluster is stored in an allocation file. HFS+ uses the allocation file to keep track of whether each allocation cluster is occupied or not. The contents of the allocation file is a bitmap data. The bitmap data contains one bit for each corresponding allocation cluster. If the corresponding allocation cluster is currently being occupied by some content, the bit is set, otherwise if the corresponding allocation cluster is empty, the bit is cleared [11].

Each byte has the status of eight allocation clusters. A byte b has eight bits as (b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8). Within the byte b, the most significant bit b_1 holds information about the allocation cluster with the lowest cluster number, the least significant bit b_8 holds information about the allocation cluster with the highest cluster number. The size of the allocation file is directly related to the number of allocation clusters in the volume, which depends both on the disk size and on the numbers of allocation clusters [9].

3. Features of FACT(**F**orensic **A**nalyzer based **C**luster Information **T**ool)

3.1 Overview

FACT consists of three features: Cluster based analysis, B-tree based analysis, and Directory based analysis. Basically, it was developed for cluster analysis, and in this tool the feature of cluster visualization plays a central role. This allows users to distinguish between system files and regular files using a visualization of the metadata and file/directory information. A B+tree structure is used to manage the metadata, as a catalog of HFS+. The B-tree based analysis analyzes the catalog data contained in the B-tree node and extracts it as data for meaningful forensic analysis. The directory based analysis is implemented using a general file and directory hierarchic structure from the leaf nodes of the B-tree, with cluster information. This is fundamentally different from obtaining directory information through the file system.

3.2 Cluster based Analysis

FACT allows system files containing metadata to be distinguished from a user's regular files. Visualization of the metadata and file/directory information is performed using a coloring method. When a certain cluster is allocated, a color is displayed indicating the type of the file/directory. When a cluster is unallocated, it is displayed in gray. In this way the assigned status can be easily and intuitively recognized.

When a cluster cell is clicked, the color of the contents corresponding to the cell changes to "red" to indicate that it has been selected. If you double-click on it, the contents of the corresponding file/directory are displayed. In Sysinternals' DiskView, only the file name, path, and basic property information of the selected file/directory are displayed, but in the FACT tool, a compound linkage feature is implemented so that you can see the meta information of all files and the contents in the file.

- Cluster coloring
- File system metadata display
- File/Directory information display
- Cluster allocation status display

3.3 B-tree based Analysis

One of the major features of the FACT tool is that it correlates information in clusters and B-tree nodes. The tool is designed so that the B-tree nodes are displayed by double-clicking a cluster located with catalog B-tree data. This function is a unique feature of FACT and is not provided by other forensic tools. There is a window which displays basic B-tree information along with various B-tree catalog information. Starting from the root node of the B-tree, through the index node to the leaf node, the B-tree nodes are displayed in the form of a tree diagram. The B-tree used here, to be precise, a B+ tree, has a structure in which leaf nodes are sequentially connected. All of the file/directory information is contained in this leaf node, so the tool has a feature that shows only leaf nodes. The tree diagram of the B-tree structure can be selectively displayed either in a simplified form, or with the entire form of the B-tree.

In general, when conducting digital forensic investigations, the data stored on a storage device is analyzed in a static form. In special cases, such as when analyzing anti-forensic cases, there may be situations in which forensic analysis must be performed using a special method. If an experimental study on dynamic changes of

the B-tree is necessary, a feature implemented in the FACT tool can be used. This is a very special feature that can conduct forensic analysis of temporal changes in files and directories.

To implement the feature, a B-tree snapshot function for the HFS+ filesystem is provided. It provides a planar function and a temporal function for cluster information. This feature was created to help experimentally determine how disk clusters and B-trees are stored on the disk when file/directory creation and deletion occur. The core feature is that by comparing snapshots before and after file/directory change, the internal changes to the b-tree node stored in the cluster can be compared and contrasted.

There are two ways to display the B-tree mode. One is text mode and the other is graphic mode. Both display the same content but in different ways. In text mode, all of the B-tree contents are displayed in the console window or terminal window. Users can capture text and save it as a file, which can be edited and utilized. The graphic mode is a feature provided for visualization. The advantage is that the structure of the b-tree can be easily understood intuitively, and with the added user interface capabilities, much more functionality can be displayed interactively.

- Text mode display
- Graphic mode display
- Leaf node only display
- B-tree snap shot

3.4 Directory based Analysis

The FACT tool provides a directory based analysis method. This method provides two functions: a directory tree display and a directory reconstruction method with only leaf nodes. The directory display method has a general directory tree configuration, which uses the same concept as the system used for file management in the operating system. When you click on a directory list or a file list, the corresponding contents and information are displayed.

In the FACT tool, a method of configuring a directory by reconstructing the metadata in the HFS+ file system is implemented, that is, the information in the catalog b-tree. This directory reorganization method uses only leaf node information. Reconstructing the directory with only the leaf node information means that the directory can be built robustly when the index node information is lost. This method can be used when it is difficult to obtain accurate file system information due to partial loss of disk information. So far, this method has been implemented for a case in which all of the leaf nodes were normally configured. In a subsequent study it will be applied to cases where the connection of the leaf node is partially lost, or the contents of the b-tree node clusters are damaged.

- Directory tree display
- Directory reconstruction based only leaf node

Figure 3 shows the overall configuration of FACT. The three major analysis components constituting this tool are indicated: b-tree analysis, cluster based analysis, and directory based analysis.

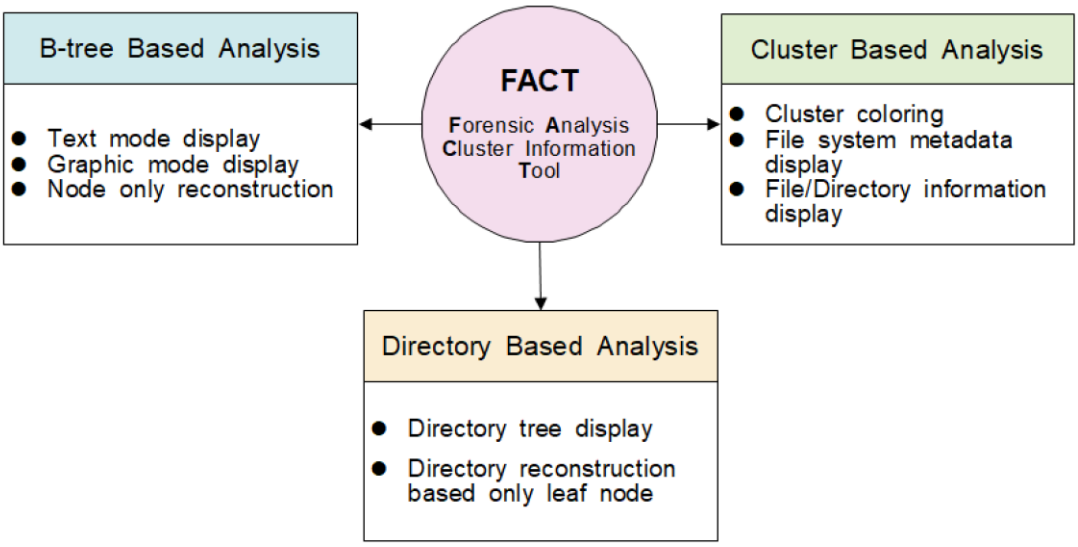

Figure 3. Feature Configuration of FACT

4. Functions for Implementation of FACT

4.1 Overview

The FACT tool provides a directory based analysis method. This method provides three analysis features, a cluster based analysis, B-tree based analysis, and Directory based analysis. The cluster based analysis is the main feature, and was basically we developed for cluster analysis. The cluster visualization feature plays a central role in the FACT tool. The FACT tool is programmed with two programming languages, C/C++ and Python. The core part for analyzing the HFS+ filesystem was programmed in C/C++ and the visualization part is implemented by Python Tkinter library. The following section shows how the FACT tool is implemented, and the executed result is shown.

4.2 Development Environments

- OS: macOS Big Sur version 11.2.2
- Application Type : macOS console program, Python execution program
- Disk Format : HFS+
- Target Storage Device: USB memory(16GB)
- Disk Allocation Cluster Size : 4,096 bytes
- Programming Language: C/C++, Python 3.9.0
- Programming Tools: Xcode v11.6, Python IDLE 3.9.0, Spyder 5.0.3

4.3 Implemented functions for FACT core features

The FACT are composed of three analysis features, the cluster based analysis, B-tree analysis, and Directory analysis based on core features. Several functions have been produced for common use by these analysis features. The FACT core features were developed using the C/C++ language, and features HFS+

filesystem storage open and close, HFS+ Volume Information Parsing, HFS+ Allocation Status Information, Header record analysis, B-tree catalog node parsing, B-tree catalog index record parsing, B-tree catalog key parsing, B-tree catalog file/folder/thread parsing, and B-tree catalog file block information parsing functions. The prototypes of the functions implemented in each function are as follows.

- HFS+ filesystem storage open and close function:
  unsigned int openHFSPlusDisk(FILE* f)
  unsigned int closeHFSPlusDisk(FILE* f)

- HFS+ Volume Information Parsing:
  getVolumeHeaderInfo(FILE* f, HFSPlusVolumeHeader* hfsplus_header)
  printVolumeHeaderInfo(HFSPlusVolumeHeader* hfsplus_header)
  int proofCatalogBySize()

- HFS+ Allocation Status Information
  unsigned int checkAllocation(FILE* f)
  BOOL isAllocated(**int** clusterNo, FILE* f)
  void dispAllocation(**int** clusterNo, FILE* f)

- Header record analysis function:
  int getHeaderRecordInfo(BTHeaderRec* header_rec, **const unsigned char*** p)
  int printHeaderRecordInfo(BTHeaderRec* header_rec)
  int printHeaderRecordUserdata(**const unsigned char*** p)
  int getHeaderRecordBitmap(**const unsigned char*** p, **unsigned char** *bitmap)

- B-tree catalog node parsing functions:
  static uint32_t getNextNode(**unsigned char*** catalog, uint32_t node)
  int getNodeDescriptor(BTNodeDescriptor* node_desc, **const unsigned char*** p)
  int printNodeDescriptor(BTNodeDescriptor* node_desc)
  unsigned char* nextNodedescOffset(**const unsigned char*** p)

- B-tree catalog index record parsing functions:
  unsigned char* nextIndexOffset(**unsigned char*** qq, uint16_t length)
  int getIndexRecordItem(**const unsigned char*** q, HFSPlusCatalogIndex* catalogIndex, **int** rec)
  void printIndexRecordItem(HFSPlusCatalogIndex* catalogIndex)

- B-tree catalog key parsing functions:
  int getCatalogKey(**const unsigned char*** q, HFSPlusCatalogKey* cKey)
  void printCatalogKey(HFSPlusCatalogKey* cKey)

  int printHeaderRecordBitmap(**const unsigned char*** p)
  int drawBTreeHeaderRecordBitmap(FILE* f)

- B-tree catalog file/folder/thread parsing functions:
  int getCatalogFolderItem(**const unsigned char*** q, HFSPlusCatalogFolder* cFolder)

void printCatalogFolderItem(HFSPlusCatalogFolder* cFolder)
int getCatalogFileItem(**const unsigned char*** q, HFSPlusCatalogFile* cFile)
void printCatalogFileItem(HFSPlusCatalogFile* cFile)
void drawCatalogFileContents(FILE *fp, HFSPlusCatalogFile* cFile)
int getCatalogThreadItem(**const unsigned char*** q, HFSPlusCatalogThread* cThread)
void printCatalogThreadItem(HFSPlusCatalogThread* cThread)

- B-tree catalog file block information parsing functions:
uint32_t getFileBlockInfo(HFSPlusCatalogFile* cFile, uint32_t* fLocation)
void printFileBlockInfo(HFSPlusCatalogFile* cFile)

4.4 Implemented functions for Cluster Based Analysis

The functions created to perform the Cluster Based Analysis are as follows. These were mainly designed to display information pertaining to clusters.

- Cluster based analysis related functions:
bTreeNode* getNodeLink(**unsigned char*** catalog, **int** nodeNum)
int getCatalogBTreeInfoByIndex(FILE* f)
void drawCluster(FILE* f,**int** clusterNo)

4.5 Implemented functions for B-tree Based Analysis

The functions produced to implement the B-tree based analysis are as follows. These were mainly composed with functions for extracting the necessary elements from the b-tree data structure used to operate the catalog data structure, and to display the corresponding information.

- B-tree based analysis related functions:
void drawDirTreeByLeafInfo(DirTree* t, **int** No, **int** depth)
int getDirTreeByLeafInfo(FILE* f)
int drawCatalogClustersBySequential(FILE* f)
int drawBTreeByIndex(bTreeNode* link)

4.6 Implemented Functions for Directory Based Analysis

The functions produced to implement the Directory based analysis are as follows. These functions typically use the same directory approach used by operating systems. However, the directory function is not designed to operate via the OS, but implements the directory by reconstructing the catalog information of the HFS + file system. It is designed for users familiar with directory structures.

- Directory based analysis related functions:
int getDirTreeFromNode(**unsigned char*** catalog, uint32_t node)
void drawTree(DirTree* t, **int** No, **int** depth)
void printDirTreeItemListAll(DirTree* t,**int** itemNum)

int getFileItemFromNodeByName(**unsigned char*** catalog, uint32_t node,HFSUniStr255 fName, HFSPlusCatalogFile* cFileCB)

4.7 Implemented functions for Display

Python's Tkinter library is used to implement the graphical user interface. The GUI is commonly used for three display functions, that is, the Cluster display, B-tree display, and Directory display. Each function consists of functions under the following functions.

- Display drawing entry-point function:
  def mainDraw() :
  def win_exit() :
- Cell and tree drawing function
  def lineDraw(num) :
  def drawCluster(fname, clusterNo) :
  def hexdump(f, pos, cluster_size, y) :
- Canvas zooming function
  def ZoomIn() :
  def ZoomOut() :
- GUI I/O(mouse, keyboard I/O) function
  def input_value(sizeValue) :
  def tConfigure(event) :
  def sConfigure(event) :
  def tClick(event) :
  def sClick(event) :
- Popup window function
  def delete(dummy_event):
  def newWindow(dummy_event) :
  def pop_window(dummy_event) :
- Property display function
  def Vproperties() :
  def Legend() :

Figure 4 shows the configuration for FACT, which is composed of three analysis features, the cluster-based analysis, B-tree analysis, and Directory analysis based on core features. The FACT core features have many functions which were developed using the C/C++ language, and the Python based Tkinter drawing library was used to develop the FACT display function. Figure 5 shows a screenshot of the cluster-based analysis window, the “red” cells show the selected file “testData.pdf”, and the hexadump window shows the content of the selected file.

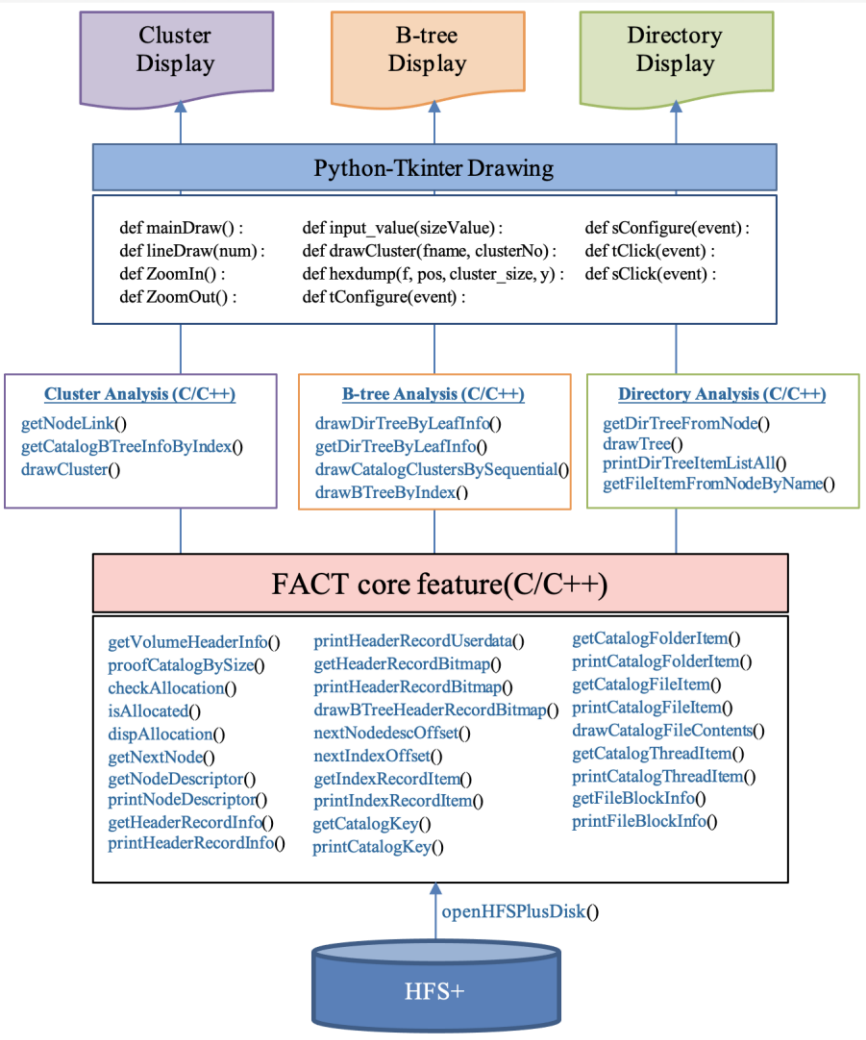

Figure 4. Classification of Implemented functions for FACT

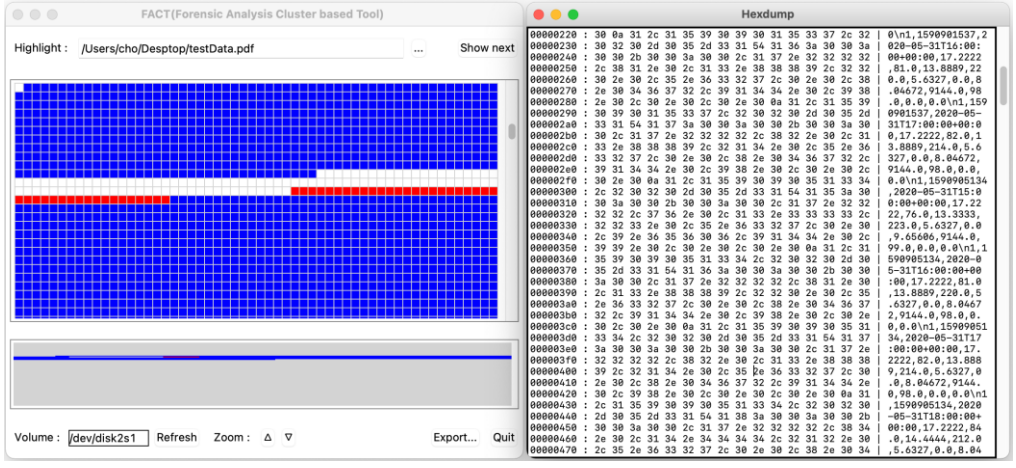

Figure 5. Screenshot of FACT

5. Conclusions

This study developed a forensic tool called FACT for cluster-centered work, and consists of three functions, a Cluster based analysis, B-tree based analysis, and Directory based analysis. It can be used to recover deleted files on a disk, or as a carving technique when partial missing content needs to be found and connected. The evidence is analyzed with cluster-centered analysis. Forensics analyzing tools, such as EnCase, TSK, and X-ways, are basically capable of getting information about disk clusters, but these are not the core functions of the tool. Rather, like Sysinternals' DiksView tool for Windows, the tool provides more intuitive features. The tools are also provided for macOS, and cluster analysis tools are very rare.

It must be useful if the forensics functionality of the FACT is extended to tools such as Sysinternals' DiskView. Many tools are designed for Windows, so it is important to develop tools that can also be used with MacOS. For these two reasons, the FACT tool developed in this study can be a valuable asset for cluster-centered analysis and B-tree analysis. The developed FACT tool was programmed with two programming languages, C/C++ and Python. The core part for analyzing the HFS+ filesystem was programmed in C/C++ and the visualization part was implemented using the Python Tkinter library. The features implemented in this study will evolve into key forensics tools for use in MacOS, and by providing additional GUI capabilities, can be very important for cluster-centric forensics analysis.

Acknowledgement

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

References

[1] Data Cluster, Wikipedia, *https://en.wikipedia.org/wiki/Data_cluster*.
[2] DiskView v2.41, Microsoft Docs, *https://docs.microsoft.com/en-us/sysinternals/ downloads/ diskview*.
[3] Christopher J. Hargreaves, “Visualisation of allocated and unallocated data blocks in digital forensics,” *8th International Annual Workshop on Digital Forensics & Incident Analysis (WDFIA 2013),* pp. 133-143. Lisbon, Portugal, May 2013.
[4] Martin Karresand, Stefan Axelsson, and Geir Olav Dyrkolbotn, "Using NTFS Cluster Allocation Behavior to Find the Location of User Data)," *Digital Investigation*, Vol. 29, Supplement, pp. S51-S60, July 2019. DOI:https://doi.org/10.1016/j.diin.2019.04.018
[5] Aaron Burghardt and Adam J. Feldman, "Using the HFS+ journal for deleted file recovery," *Digital Investigation*, Vol. 5, pp. S76–S82, 2008. DOI:10.1016/j.diin.2008.05.013
[6] S. G. Bang, S. J. Jeon, D. H. Kim and S. J. Lee “A Study to Improve Ration of Deleted File Using the Parsing Algorithm of the HFS+ Journal File, *“KIPS Transaction on Computer and Communication Systems,”* Vol.5, No.12 pp.463-470, 2016. DOI:https://doi.org/10.3745/KTCCS.2016.5.12.463
[7] Gyu-Sang Cho, “An Arbitrary Disk Cluster Manipulating Method for Allocating Disk Fragmentation of Filesystem,” *Journal of KSDIM*, Vol. 16, No. 2, pp.11-25, 2020. DOI:http://dx.doi.org/10.17662/ksdim.2020.16.2.011
[8] HFS Plus, Wikipedia, *https://en.wikipedia.org/wiki/HFS_Plus.*
[9] Technical Note TN1150, HFS Plus Volume Format, *https://developer.apple.com/legacy/library/technotes/tn/tn1150.html*.
[10] Cory Altheide and Harlan Carvey, *Digital Forensics with Open Source Tools*, pp. 123-141, 2011
[11] Amit Singh, *Mac OS X Internals: A Systems Approach*, Addison-Wesley Professional, 2006.
[12] The Eclectic Light Companny, Inside the file system: 2 HFS+ volumes, *https://eclecticlight.co/2020/10/07/inside-the-file-system-2-hfs-volumes/*