Building Imaging Forensic Application Using Libewf and Design Pattern Approach

Muhammad Miftahul Huda[1], Gusti Made Arya Sasmita[2] and Kadek Suar Wibawa[3]

[1] Information Technology Department, Faculty of Engineering, Udayana University Badung, 80362, Indonesia

[2] Information Technology Department, Faculty of Engineering, Udayana University Badung, 80362, Indonesia

[3] Information Technology Department, Faculty of Engineering, Udayana University Badung, 80362, Indonesia

Abstract

Computer science has many branches. Digital forensics is one of these branches. One application of digital forensic science is software for digital forensic purposes itself. The ewf file format is a commonly used forensic disk image format. Using Libewf to help programmers develop a forensic imaging application in storing acquisition data into ewf files. Application development can be focused on designing business logic, data flow, choosing design pattern, and implementation to program code. The role of the design pattern in building forensic applications aims to classify the related components into an object, dividing the program flow into small sub-programs, so that it can improve program code reusability and be easy to maintain. We can also know which design pattern is suitable to apply to certain problems in making digital forensic applications.

Keywords: *Digital forensic, Ewf file format, Libewf, Design pattern, Imaging, Qt frame work.*

1. Introduction

Digital forensic is one of specialzation of computer science. It is used to proofing the crimes that involve computer technologies and devices[1]. Officers or investigators as digital forensic analyst must use methods or procedures that proofed scientifically. Thus forensic applications are created to ensure the principle chain of custody[2] and integrity of data that contained inside evidences.

Application for imaging media storage is one from many forensic applications. It is useful to acquire data digital that stored inside media storage such as harddisk, thumb drive, memory card, and optical drive. As what we already known data digital inside media storage are volatile and tend to lose or broken if we not care enough maintaining the media storage. Imaging application has a role to makes snapshoot from the current media storage's state[3]. With the disk image[4] as output from imaging process, officer can performs any analyst methods using it and keep the original media storage stay safe.

Libewf[5] as open source library is able to create ewf file that can store evidence data together with case's metadata. Programmers can build their own imaging application or also ewf file reader application with using Libewf library as interface that can handle data that needed from or into ewf file and the application. Besides for C/C++, this library has support for Python programming language.

2. Overview

The proposed forensic application is a desktop GUI based application. It is builded with using Libewf, thus the application will has ability to creating ewf file from media storage, retrieving metadata and storage bytes from ewf file using functions that imported from Libewf. GUI components for application and logic implementation are created by using Qt C++ library.

2.1 Application Architecture

Author's architecture of the application designed with different individual components that will handle different task according their functionality. One component can be used together with other components to accomplish complex process. Application's architecture is depicted in Figure 1.

In application layer, application contains main components that grouped together as utility and Libewf as the backend library. Operating Sytem (OS) layer has functionality to provides how application access/interacts with Storage layer, thus application can access media storage with procedure provided by OS then performs imaging process.

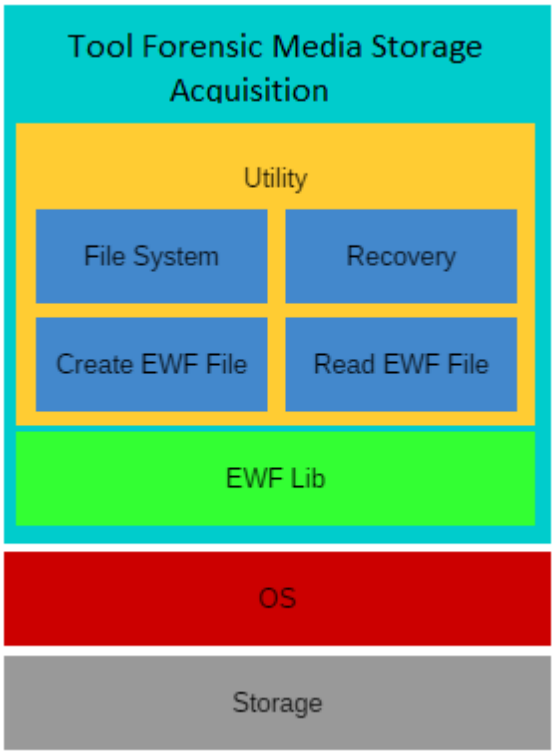

Fig. 1 Architecture of application.

2.2 Create EWF File Component

Create ewf file componen has a role to creates disk image from metada given by user and raw bytes from source media storage. The process is straigh forward from the start until process end with using some functions from Libewf. Here Figure 2 is pseudocode about how component create ewf file works, pseudocode with prefix "libewf_" means it use function from Libewf.

```
    libewf_create ewfhandle instance
    libewf_open ewfhandle instance
    open media storage
    libewf_insert metadata values into
ewfhandle
    libewf_set_compression ewfhandle
    create hash value instance

    while media storage sector 0 to last
sector then
        read sector
        update hash value from sector
        libewf_write sector to ewfhandle
    end while

    libewf_set_hash hash into ewfhandle
    libewf_write_finalize ewfhandle instance
    libewf_close ewfhandle instance
    close media storage
```

Fig. 2 Pesudocode create ewf file

The big process is the looping section. The time to complete imaging process is directly proportional with media storage's total sectors. That we can see the simple implementation of the loop pseudocode in Figure 3

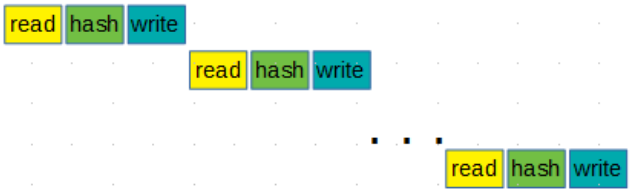

Fig. 3 Simple loop implementation

Each loop start after another one have finished it's write process. There is one case if write process takes longer time than what it should means new loop process will not started and get delayed, thus that will increases amount of time imaging pocess. From that figure too we know, while write process run, source storage is in idle state because there is no read process running.

To press the amount of imaging process time and idle time of source storage[6], we can make write process run simultaneously with read process in the next loop. To make this posible, we need add another threads to execute write process. We can achive big throughput in short time with this method.

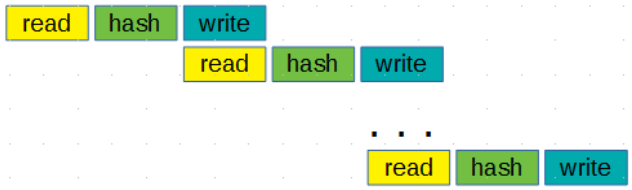

Fig. 4 Looping with multi thread

From Figure 4 we can see the read process will executed after calculating hash process executed in main thread and the other threads will execute write process asyncronously. There are many multi threading paterns we can use to implementing this approach. As an example we can use thread pool patern, that the main thread executes read and calculate hash process then notify and give the data to one of available worker threads to execute write process. Another example is with using Semaphore patern, that the main thread will execute read and calculate hash process then store the data into a list or queue, while thread workers watching the list or queue and perform get the data then write process when data available inside list or queue.

2.3 Read EWF File Component

Read ewf fle compoent has role as interface between application and Libewf to open and to read ewf file. This component has ewfhandle instance that used to hold user's opened ewf file. As interface means this component will hides or emulates any processes related with Libewf function to initializating an ewfhandle instance, open file ewf, close ewfhandle instance, read storage data from ewf file, seek position from ewf file, and error handling for the application. Therefor in implementation, this componenent will and only provides open(), read(), seek(), close(), and function to retrives ewf metadata. Other components that need data from ewf file should access or request from this component.

2.4 File System Component

File System component has roles to read meta data that stored inside partition and translate directory structure into directory list. File system is one aspect of operating system that the most felt by user[7]. File system gives user a method to save data as strcture file and directory[8]. Inside ewf file exist partition that also contains file system inside, thus with implementing file system logic in file system component for application we can retrives those directories and files that stored inside ewf file. When user wants to see or request what are files inside a folder, this component will called by application to process that request.

There are many file systems exist that supported by many operating system too. To support many file system logics for the application that easy to maintain, author use abstract class and inheritance. That is common scenario when we need object or instance with farious implementation in one function, then we need child classes to implement abstract method from parent class. In Object Oriented Programming we can assign object of parent class for it's child class instance (up casting), we can let an empty object of parent class declared then assign it at running time with apropriate child class instance. Author in implementation inside application uses this method for every object that needs to use file system component object. These object only have empty file system object inside and it will be filled at the run time apropriate by file system type the partition has.

To handle instantiating various types of file system component object at run time, author use a factory pattern[9]. This approach will increases code's modularity along with easy to maintain. All of logics to instantiating a correct file system component object will be handled inside factory object. Component that needs file system instance only needs to calls factory object with some parameters passed to factory object, then it gets file system instance.

File system component has dependency with read EWF file instance for it's data source. Author creates data transaction between file system component and read EWF file component in sector level. Each time the file system instance need to get data, it will requests read EWF file instance to reads sectors where that data stored. Any translation between file system's logical address into physical address in read EWF file should be handled by file sysytem object.

2.5 Recovery Component

Recovery component has role to export file from file system instance into user's storage, so file can be opened and analized using other tools. This component works with metadata file as the input, that will inform there is a file with some size of bytes will be written into user's storage, and where is it first cluster in the file system. Next, this component need path in the user's storage as the output's destination. Recovery process then works by requesting files'content from file system instance until all bytes of the file already written to the output's destination.

Recovery component need file's content from files system instance. In this component point of view, it will request cluster by cluster to the file system instance. By that, file system component should be able to handle cluster address translation to the apropriate logical address inside file system before translated into physical address. Complete addresses translation for file system component is depicted in Figure 5. The function will be implemented by child classes acording what file system type they will purposed to handle.

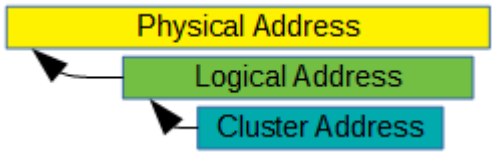

Fig. 5 File system address translation

3. Implemented Features

3.1 Aquisition

This feature is used to copying byte per byte for each sectors from original media storage into an ewf file as output. Ewf file as disk image will be used for further analysis and investigation

This feature use Create ewf file component as core process that will be executed after user provides data source and metadata for for imaging process. Firstly user will see choose disk dialog that depicted in Figure 6. User should chooses which media storage that want to be aquired either logical drive type to acquire whole data of one partition inside the logical drive or physical drive type to acquire whole data inside selected device.

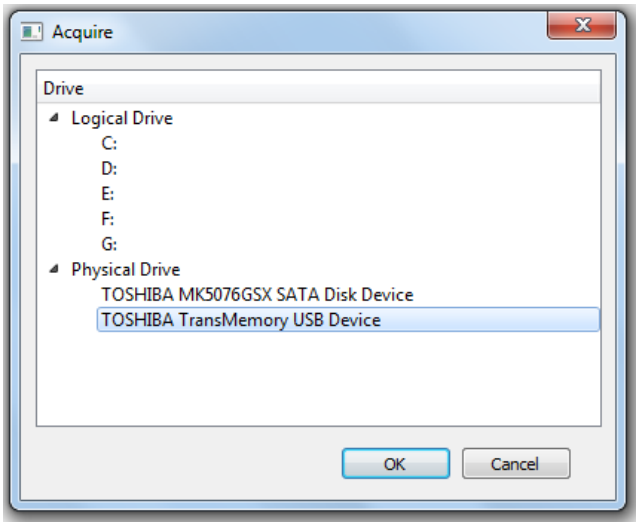

Fig. 6 Choose drive dialog

Secondly after user chooses media storage, user will see information form like in Figure 7 to input case metadata for media storage and output file destination. In this form, user able to sees media storage's physical specification.

Finally after user provides evidence's metadata and correct output path in information form, user can proceed into imaging process in the Figure 8 and waits until this process finished.

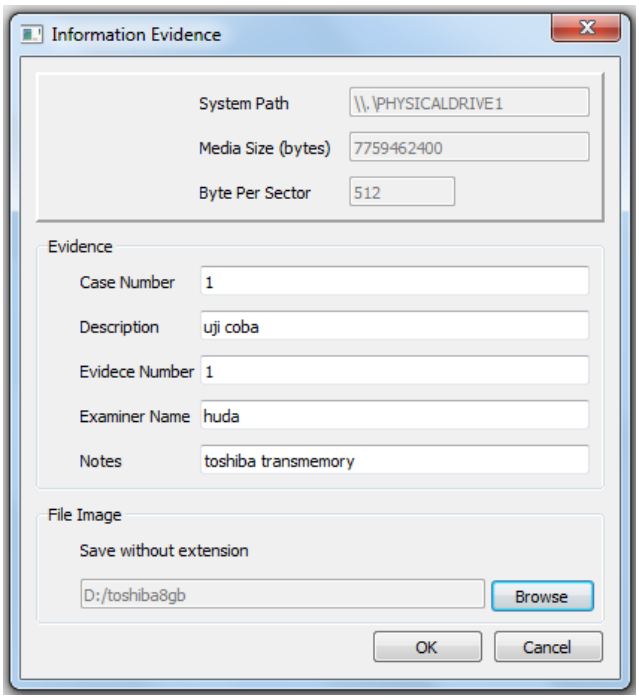

Fig. 7 Information form

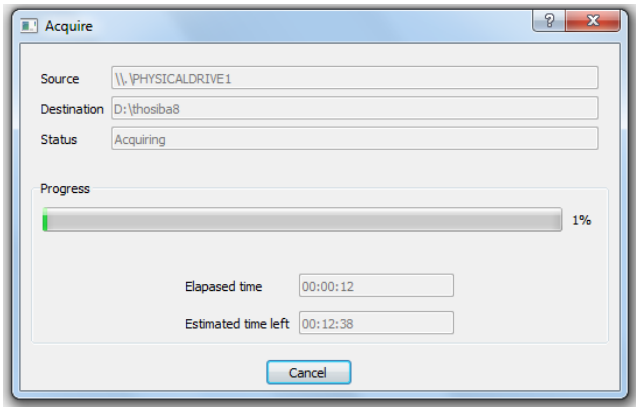

Fig. 8 Imaging process

3.2 Open EWF File

This feature intended to retrive data stored inside ewf file, thus user can be able to sees metadata that inserted at acquisition also with all bytes from media storage that acquisitioned. This feature consist of some sub components that using different utility component according to their functionality. Figure 9 shows an ewf file that opened inside open ewf file window.

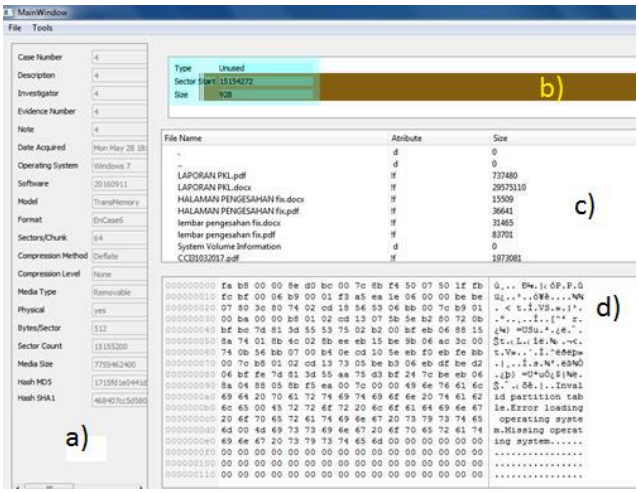

Fig. 9 Open ewf file

3.3 Metadata Panel

Metadata panel is used for displaying ewf file's metadata such as storage's size, total sectors, ewf type, chunk size, case metadata, and the last is storage's hash checksum value. When ewf file success opened, the first task to do are retrieving metadata stored inside ewf file, then give it to metadata panel to be displayed. Metadata panel is shown in Figure 9a.

3.4 Block Visualizing

Block visualizing is feature to create a visual representation of blocks of sectors contained inside media storage. Sectors of media storage will be divided into some fragments or blocks, as example sector 0 is used for MBR that contains partition table, next is some sectors started at position *n* until sector *m* will be used as a partition like what written in partition table, and another blocks of sectors as partition if there are more than one partition exist in partition table.

With block visualizing, user can see how many partitions exist inside media storage, counts how many sectors inside block, and which block contains the most or the smallest sectors. User can do these processes manually by looking at partition table and calculate them. Block visualizing help user from doing these manual calculations.

Unused sectors also included and displayed in block visualizing. Unused sectors may exist inside media storage between two partitions, between MBR and first partition, even after the last partition. Continues unused sectors will be grouped as one block then labelled as unused block.

Author makes block visualizing become informative, attractive, and simple for user. When user hovers mouse's pointer on block, an information panel will appear that contains type of block either MBR, partition, or unused block, then block's offset of start sector, the last is block's size that sum of sectors contained inside block. Block visualizing is shown in Figure 9b.

3.5 Directory Listing

Directory listing is used to displaying files and directories inside partition. With this, user also can performs change directory either entering directory or move back to parent directory.

This feature using file system component implementation to retrieve raw bytes inside directory's clusters. All retrieved metadata will be translated into appropriate item either directory or file with it's name, size, first cluster, and is it already deleted or still exist.

Data that displayed into user are item's name, type, and size. Item with directory type will be marked with character "d" and character "f" for file type. Deleted item will be marked with character "!" in front of it's type. Directory listing is displayed in Figure 9c.

3.6 Hex View

Hex view is used to displaying raw bytes contained inside media storage from start sector until last sector. This component will display each bytes with two digits hexadecimal number in the left part, then followed with it's character representation in the right part. Hex view is displayed in Figure 9d.

3.7 File Recovery

File recovery is feature to export file type item that listed inside directory listing either the file still exist or deleted file. This feature use recovery component to complete it's task.

To perform recovery task user must performs right click on file type item that listed in directory listing like in Figure 10 as example. Next step is user should choose destination directory for application to write output file, this can be shown in Figure 11. Finally Recovery component will do it's task to get file's data cluster by cluster and write them into output file.

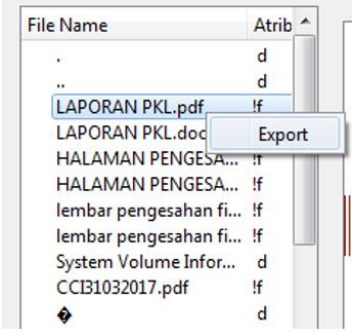

Fig. 10 Right click on file type item

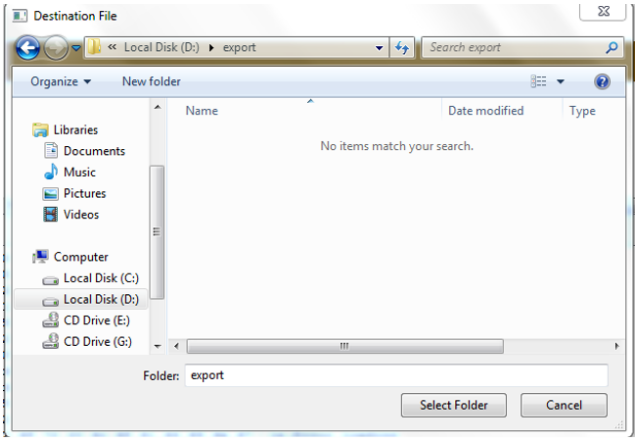

Fig. 11 Choose destination directory

4. Conclusions

Design pattern helps to create better code by separating the logic into smaller pieces that most of them can be reusable for other components. For imaging application authors create four main component utilities that applied in main program appropriate to their usability. Decreasing idle time of resource can make long running process done faster, like imaging process, by decreasing media storage source's idle time with using other threads to perform write process asynchronously. For that thread pool pattern is chosen to maintains thread workers. Factory pattern is suitable to maintain collection of object instantiation, thus this pattern is used to handle instantiation of any type of file system objects.

In the future, authors hope can improve this application either by adding new features or improving algorithm that can make application run faster, more reliable, and better memory management.

Acknowledgment

Authors give thank you very much to team Laboratorium Forensik POLDA Bali for their support provide authors place and devices related digital forensic for application test. Also with their advices that help authors make application that fulfill principle of digital forensic.

References

[1] M. N. Al-Azhar, *Digital Forensic Panduuan Praktis Investigasi Komputer*. Jakarta: Salemba Infotek, 2012.
[2] ACPO, "ACPO Good Practice Guide for Digital Evidence," no. March, p. 41, 2011.
[3] C. Prosise and M. Kevin, *Incident Response&Computer Forensics, Second Edition*, 2nd ed. New York: "McGraw-Hill, 2003.
[4] S. Vandeven, "Forensic Images: For Your Viewing Pleasure," *SANS Inst.*, p. 38, 2014.
[5] Joachimmetz, "LibEwf." [Online]. Available: https://github.com/libyal/libewf/. [Accessed: 09-Dec-2015].
[6] S. H. . Hamid, M. H. N. . Nasir, W. Y. Ming, and H. Hassan, "Improving the Performance of the Authorization Process of a Credit Card System Using Thread-Level Parallelism and Singleton Pattern," *Res. J. Inf. Technol.*, vol. 1, no. 1, 2009, pp. 30–40.
[7] G. Gagne, B. P. Galvin, and A. Silberschatz, *Operating System Concepts*, 8th ed. USA: John Wiley & Sons, 2008.
[8] B. Carrier, *File System Forensics Analysis*. Addison Wesley Profesional, 2005.
[9] E. B. Eskca, S. Bondugula, and E. T. Tarik, "Simplifying the Abstract Factory and Factory Design Patterns," *ARPN J. Sci. Technol.*, vol. 4, no. 12, 2014, pp. 789–794.

Muhammad Miftahul Huda born in Denpasar, 16 February 1995. He was Educated in Department of Information Technology, Udayana University.

Gusti Made Arya Sasmita born in Pengastulan, 26 March 1973. A lecturer in Department of Information Technology from Udayana University. Received the Bachelor of Engineering degree in Electrical Engineering from Udayana University, and Master of Engineering degree in Electrical Engineering from Gadjah Mada University.

Kadek Suar Wibawa born in Sangsit, 16 August 1983. A lecturer in Department of Information Technology from Udayana University. Received the Bachelor of Applied Science degree in Electrical Engineering from Bandung Institute of Technology, and Master of Engineering degree in Electrical Engineering from Bandung Institute of Technology.