Forensic Analysis of ReFS Journaling

Seonho Lee[a], Doowon Jeong[b,c,*], Jungheum Park[b], Hyunuk Hwang[a], Seungyoung Lee[a], Sangjin Lee[b]

[a]*The Affiliated Institute of ETRI*
[b]*Institute of Cyber Security & Privacy (ICSP), Korea University, Seoul, 02841, South Korea*
[c]*Department of Transdisciplinary Security, Dongguk University, Seoul, 04620, South Korea*

Abstract

Since the analysis of file system is a fundamental step in forensic investigation, file system forensics has been steadily researched. Especially, NTFS forensics has been mainstream research as it is used by Windows, a globally most-used operating system. When investigating NTFS, journaling analysis is an important procedure as it can identify which files are created, modified, and deleted. Meanwhile, Microsoft developed the Resilient File System (ReFS), which is also used in Windows, to maximize data availability; ReFS is also expected to be a popular file system. Similar to the $Logfile and the $UsnJrnl of NTFS, there are artifacts in ReFS: the Logfile and the Change Journal that document information regarding changes to the system.

In this paper, we present the structure and operation of the Logfile and the Change Journal. By kernel reverse engineering, we identify that the ReFS artifacts related to journaling are quite different from the NTFS artifacts; the ReFS artifacts use new record formats, named Log Record and USN_RECORD_V3, and the metadata of ReFS handling journaling files is distinct from that of NTFS. Through experiments, we identify logging patterns of transaction record and examine the mechanism of ReFS journaling. In this process, we enhance the knowledge of the metadata and structure of ReFS presented by previous research. Based on the result of our research, we also propose a forensic methodology of ReFS journaling and develop a tool, Awesome ReFS Investigation tool (ARIN), which is an open-source for analyzing the ReFS journal. These outcomes may provide considerable assistance to a forensic examiner trying to investigate ReFS volumes.

Keywords: File System, ReFS, Transaction, Journaling, Logfile

1. Introduction

In digital forensics, file system analysis has been steadily researched. A forensic examiner can identify files and corresponding metadata on a digital medium by analyzing a file system. When past traces about deleted files exists, the examiner may recover them; this is why many researchers focus on file system forensics. In particular, some file system record logs for a user's behavior, such as file creation, file deletion, file move, etc., which is very helpful in resolving a case [2]. Among the file systems, NTFS developed by Microsoft has a journaling function, which has been one of the most actively studied topics in digital forensics [7]. The NTFS has the $Logfile and the $UsnJrnl that log file system transactions [4]; they are relevant to any type of case including malware analysis, leakage of confidential information, and infringement of copyright [3]. Another reason why many examiners have been interested in NTFS forensics is that NTFS has been used by Windows for the system volume. As Windows still use NTFS as default file system, research results are actively used in practice.

Meanwhile, Microsoft also developed a new file system named the resilient file system[1] (ReFS) [9]. ReFS was also developed by Microsoft, which has been available since Windows 8 and Windows Server 2012 [8]. As expected to increase usage of ReFS, methods of forensic analysis for ReFS have also been studied. Metz [8] researched structure of ReFS v1.1 and 1.2 and Andrew [1] analyzed the ReFS by comparing with FAT32 and NTFS. Georges [5] documented the reverse engineering of ReFS v1.2. Nordvik et al. [12] described the main structure necessary to interpret ReFS v3.x and released the prototype tool to parse the structure of ReFS v1.2. Prade et al. [14] described the overall structure of ReFS by representing the core concept of ReFS and newly introduced data structures. Prade et al. also extended The Sleuth Kit allowing it to parse ReFS partitions and recover deleted data.

Our research began with the following question: as ReFS was also developed by Microsoft which also developed NTFS, are there the $Logfile and the $UsnJrnl or something similar on ReFS volume? We considered that the ReFS transaction, if it exists, should be analyzed as the transaction is a very important clue for a forensic analyst. In this paper, to answer the question, we parse ReFS referring to previous researches and identify system files. Based on the knowledge of ReFS, we introduce the Logfile and the Change Journal that have transaction logs, similar to the $Logfile and the $UsnJrnl of NTFS. We also figure out a principle of generating ReFS'transactions and the structure of the journaling files. Based on our research, we develop an automation tool parsing and analyzing the files. The

*Corresponding author.
Email addresses: seonho@nsr.re.kr (Seonho Lee), dwjung77@gmail.com (Doowon Jeong), jungheumpark@korea.ac.kr (Jungheum Park), sangjin@korea.ac.kr (Sangjin Lee)

[1]The most recent ReFS version is v3.4 on March 2020.

main contributions of this work can be summarized as follows.

- We introduce novel forensic artifacts stored on ReFS volume, similar to the $Logfile and the $UsnJrnl of NTFS.
- We describe the structures of the Logfile and the Change Journal which has not been studied. Based on the structures, transactions of the journaling files can be parsed.
- By kernel reverse engineering, we enhance the knowledge of metadata related to the journaling files.
- We release a tool named Awesome ReFS Investigation tool (ARIN)[2], an open-source tool for analyzing the ReFS journal.

The remainder of the paper is organized as follows: we describe an overview of ReFS to identify a file list related to journaling in Sec. 2. Next, Sec. 3 introduces the structures of the ReFS journaling files and explains the journaling principle in ReFS. Sec. 4 proposes a forensic methodology for investigating ReFS and Sec. 5 describes our implementation and experiment. Finally, Sec. 6 includes the main conclusion drawn from this study.

2. ReFS Overview

In order to identify ReFS structure and operation principle, we reverse engineered 'ReFS.sys' driver and 'refsutil.exe' utility developed to recover a damaged ReFS volume. They are stored in '%SystemRoot%\System32' on system volume of Windows 10 Enterprise Ver.1803. We used WinDbg 10.0.18362.1, x64dbg, Resource Hacker, and IDA Pro 7.2 to reverse engineering ReFS.sys and refsutil.exe. By reverse engineering, we identified the names of functions and internal fields given by the file system developers (see Fig. 1). We were also able to identify the names of metadata files, data structures, and event messages as ReFS.sys, which is a file system driver, includes all operation functions of ReFS.

In this section, we first introduce important data structures used in ReFS: ***page***, ***table***, and B^+ ***tree***. The data structures have been stressed in previous works, therefore, this paper briefly explains what is the structures and how they are used in ReFS. We also describe various metadata files of ReFS (See Fig. 2). Though previous research introduced most of metadata files, we review and enhance knowledge about principal metadata files used to examine ReFS journaling. This paper describes five metadata files: the Object ID Table, the Container Table, the Parent Child Table, the File System Metadata, and the Logfile Information Table.

2.1. ReFS data structure

As previous research stressed, ReFS uses ***page*** to allocate data. The page is at least as large as a cluster and its size ranges from 1 to 4 clusters. Every page has a header area that represents identifier, address, and page type [15].

```
__int64 __fastcall CmsLogRedoQueue::PerformRedo
{
  struct CmsStream *v3; // r15
  struct CmsBPlusTable *v4; // r12
  struct _SmsRedoRecord *v5; // rdi
  struct CmsTransactionContext *v6; // r14
  CmsLogRedoQueue *v7; // rsi
  struct _CmsKey *v9; // r13
  signed int v10; // ebx
  __int64 v11; // r8
  struct CmsBPlusTable *v12; // r9
  CmsLogRedoQueue *RedoOpcode; // rcx
  bool v14; // al
  CmsLogRedoQueue *v15; // rcx
                                        ~ snip ~
  if ( (unsigned int)RedoOpcode <= 0xF )
  {
    if ( (_DWORD)RedoOpcode == 0xF )
    {
      v20 = CmsLogRedoQueue::RedoDeleteTable(v7, v6, v5, v4, v3);
      _mm_storeu_si128((__m128i *)&v35, (__m128i)0i64);
    }
    else
    {
      if ( (unsigned int)RedoOpcode > 7 )
      {
        if ( (unsigned int)RedoOpcode <= 9 )
        {
          v20 = CmsLogRedoQueue::RedoSetRangeState(v7, v6, v5, v12, v3);
```

Figure 1: A screenshot of the result of reverse engineering ReFS.sys

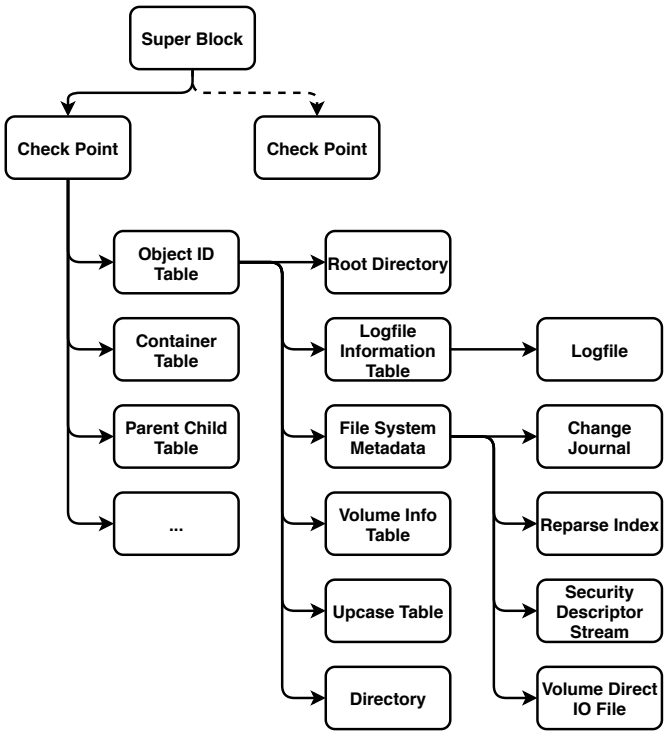

Figure 2: ReFS's metadata files

[2]URL: Closed for Double-Blind Peer Review

Table 1: Container Table Row Structure

Offset	Field	Description
0x10~0x17	Container ID	Container's ID
0x18~0x9F	Unknown	-
0xA0~0xA7	CSC	Starting position of container
0xA8~0xAC	CPC	The number of clusters per container

Everything from directories and files to metadata files is stored in particular data structure, named ***table*** [15]. In the table, there are several rows describing information about directories or files. For example, in rows of a directory's table, information of sub-files or sub-directories are recorded. In rows of a file's table, metadata of the file, such as allocation information, MAC time, and file name, is stored.

There is one other point which claims forensic researcher's attention; ReFS uses B^+ ***tree*** to locate attributes. The concept of the attributes is similar to the $MFT of NTFS. ReFS uses a single B^+ tree where metadata, allocators, files, and folders can be found instead of using the $MFT [1, 12]. In B^+ tree, the concept of key-value store is used. When logging a file system operation on ReFS volume, the operation is stored as a key-value unit, therefore it can be seen that the key-value is an atomic unit of file system log.

2.2. Object ID Table

In the Object ID Table, object IDs and location of all directories on ReFS volume are found. At the location of a directory, the name and metadata of the directory can be identified. The object ID table is also used to find full path of file; it is explained in Sec. 4.2.

2.3. Container Table

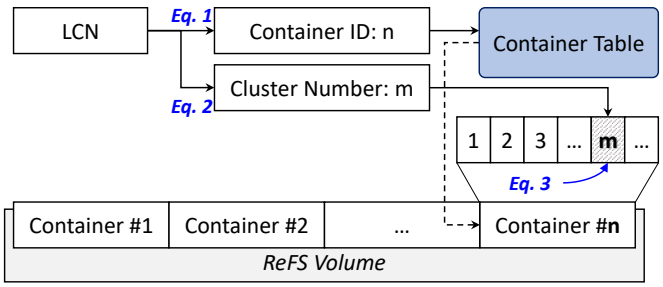

Figure 3: Process of Obtaining Physical Cluster Number from LCN

In ReFS, logical cluster number (LCN) is used to address physical offset on ReFS volume. To find physical offset from LCN, container ID and cluster number (CN) should be identified first, as seen in Fig. 3. The "containers" refers to equally segmented parts so ReFS volume is separated into multiple containers [15]. A container ID is calculated by Equation (1), and then a starting position of the container is identified by referring to the Container Table. In this process, the container start cluster (CSC) and the number of clusters per container (CPC) are also identified. The cluster number (CN) is calculated by Equation (2), and then physical cluster number (PCN) is identified by Equation (3).

$$Container\ ID = LCN >> CPC\_BitCount \tag{1}$$

$$CN = LCN\ \&\ (CPC - 1) \tag{2}$$

$$PCN = CSC + CN \tag{3}$$

2.4. Parent Child Table

The Parent Child Table establishes a relationship between parent directory and child directory. The directories are distinguished by object ID. Thus, if knowing an object ID of a specific child directory, an object ID of parent directory can also be identified. Based on a hierarchy of directories from the Parent Child Table, the full path of file or directory is determined with the Object ID Table as the names of all directories can be identified as described in Section 2.2.

2.5. File System Metadata

By exploring the Parent Child Table, it is found that the File System Metadata is a sub-directory of the root directory. There are four metadata files in the File System Metadata: the Change Journal, the Reparse Index, the Security Descriptor Stream, and the Volume Direct IO File.

In the Change Journal, change logs are recorded, similar to $UsnJrnl of NTFS. The Change Journal has consecutive logs listing changes of ReFS, however, for forensic analysts, it may be less useful than the Logfile because it is deactivated by default. The internal structure of the Change Journal is described in Sec. 3.1. The Reparse Index stores information of all symbolic link files on ReFS volume. It seems to be similar role to \$Extend\$Reparse used to keep track of all files that implemented reparse points. The Security Descriptor Stream is used to manage access security and ACL (Access control lists) of security descriptor. Its' role is similar to $Secure of NTFS. The Volume Direct IO File is unknown metadata; it is our future work.

2.6. Logfile Information Table

The Logfile Information Table stores information of the Logfile. As seen in Fig. 2, a location of the Logfile Information Table is identified by interpreting the Object ID Table. In the Logfile Information Table, size and location of the Logfile are stored as seen in Table 2. It also stores the location of the control area of the Logfile; in the area, the location of the last log record is represented.

Table 2: Logfile Information Table

Offset	Length	Description
0x10	0x8	Start offset of data area of Logfile
0x18	0x8	End offset of data area of Logfile
0x20	0x8	Size of data area of Logfile
0x28	0x8	LCN of the first entry in control area
0x30	0x8	LCN of the second entry in control area
0x38	0x8	Unknown

3. ReFS Journaling file

ReFS has journaling files: the Chang Journal and the Logfile. Similar to famous file systems such as NTFS, EXT, and APFS, ReFS also stores what files were overwritten, deleted, or created recently to the journaling files. We predicted that the journaling files are analogous to $UsnJrnl and $LogFile examined by [6, 7], however, we found out that their internal structure and contents are different from the NTFS's files. In this section, we introduce the structure and mechanism of the journaling files that can be used as forensic artifacts.

3.1. Change Journal

The Change Journal stores USN records that represent changes of file system. In ReFS, the Change Journal is deactivated by default, therefore, if a user wants to use USN journaling, the user should activate it using fsutil. When a file or directory of ReFS volume is edited, the Change Journal records the explanation of the changes. Therefore, a forensic examiner can trace the past event occurred in ReFS volume if the examiner can interpret the Change Journal. The Change Journal played a similar role to $UsnJrnl, however, there are major differences in allocation method and internal structure.

When $UsnJrnl tried to store a new transaction and the $UsnJrnl is fulled of previous transactions, the previous records turn into the spare area [2]; it means that new blocks of disk are allocated for the new transaction. This property is used to carve the deallocated (deleted) transactions if they have not yet been overwritten [16]. Unlike to the $UsnJrnl, the Change Journal use a circular buffer; the oldest transaction are constantly overwritten by new transaction. It means that it is nearly impossible to recover deleted USN journal in ReFS.

The Change Journal uses USN_RECORD_V3 format when storing USN journal [11]. As $UsnJrnl uses USN_RECORD_V2 [10], the method to analyze NTFS journaling is not directly applied to the Change Journal. The structure of USN_RECORD_V3 is similar to that of USN_RECORD_V2; the only difference is the field of file reference number. The USN_RECORD_V2 allocates 32 bits to represent the File Reference Number field, however, USN_RECORD_V3 allocates 128 bits (See Fig. 4). As the name and contents of the remaining fields are identical to USN_RECORD_V2, USN_RECORD_V3 also holds valuable information regarding what happened on the volume.

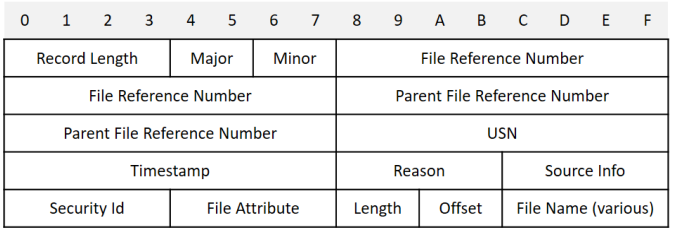

Figure 4: USN RECORD V3

3.2. Logfile

Similar to NTFS and EXT, ReFS records transactions to protect the data integrity across failures. The transactions of ReFS are documented in the LogFile. Though both NTFS and ReFS are developed by Microsoft, there are main differences:

- The Logfile of ReFS documents only redo operation, whereas $Logfile of NTFS documents redo and undo operation.
- The Logfile uses a totally new structure.
- The opcode of the Logfile is also different from the opcode of $Logfile

ReFS uses AOW (Allocate On Write) transactional model, which means that a new page is allocated when metadata of ReFS is updated; the previous page is not overwritten [9]. Thus, ReFS do not need to document undo operation as it just prepare failure of change operation, analogous to EXT file system.

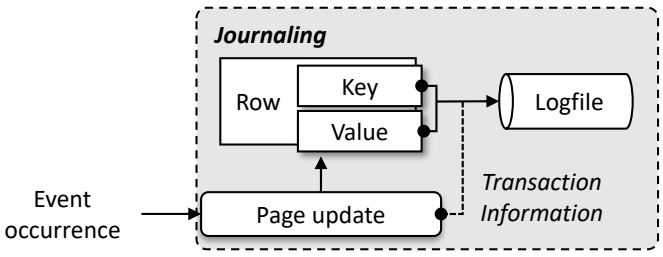

Figure 5: The mechanism of ReFS Journaling

The mechanism of ReFS journaling is shown in Fig. 5. When an event such as creating files or directories, renaming files or directories, or changing the contents of files occurs, operations that related pages occur. The updating page means that the table's row (key-value) of page is changed, therefore, the row is recorded as a file system log. Multiple operations are regarded as a transaction. And then, the transaction is written to the Logfile with transaction information.

As seen in Fig. 6, Log Record consists of Redo Records and Log Records are included in Logfile Entry; it is regarded as an atomic unit of the Logfile. Each entry is contained in Control area or Data area. The Control area stores metadata of the Data area, which contain an information of ReFS transactions. The Data area also use a circular buffer like the Change Journal. In $LogFile of NTFS, there are also similar areas: Restart area

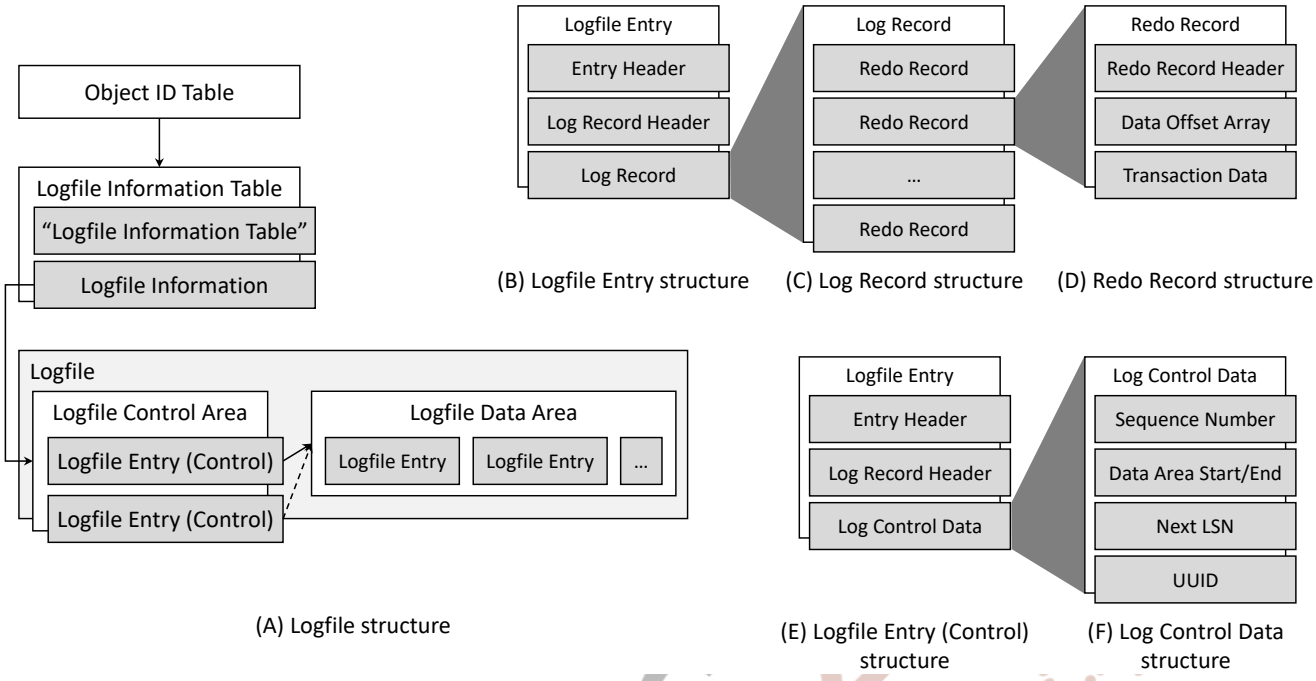

Figure 6: Logfile structure

and Logging area. Whereas the Restart area and Logging area of $LogFile are allocated consecutively, the Control area and Data area are not.

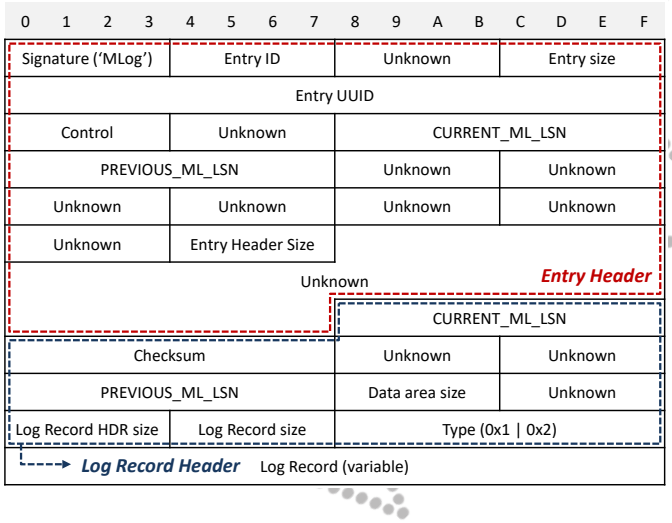

Figure 7: Logfile entry structure

The overall structure of the Logfile is shown in (A) of Fig. 6. The Logfile Entry consists of Entry Header, Log Record Header, and Log Record (See (B) of Fig. 6). The Entry Header is 120-bytes and it starts with 4-bytes signature 'MLog'. In the Entry Header, LSNs of current and previous Log Entry are stored. Log Record Header is 56-bytes and it stores checksum of the Entry Header. In the Log Record Header, there is a field named "Type" that represents whether the Logfile Entry is included in Control area or Data area; 0x1 means Control area and 0x2 means Data area (See Fig. 7).

The Log Record consists of several Redo Records. The Redo Record consists of Redo Record Header, Data Offset Array, and transaction data (see (D) of Fig. 6 and Fig. 8) The Redo Record Header includes opcode and location of Data Offset Array to redo when file system operation is failed. The opcode is designed to represent 28 file system operations as seen in Table 3. The Data Offset Array contains location and size of transaction data for redo. The location indicates offset based on the starting position of Redo Record. The transaction data also contains opcode that represents tasks to redo. With analysis of opcode and transaction data, what operation actually happened can be identified.

The structure of the Logfile Entry in Control area is shown as (E) of Fig. 6. The Entry Header and Log Record Header are same as those of Data area, however, the Log Control Data, instead of Log Record, is stored. In the Log Control Data, a sequence number and LSN representing location the next record to be allocated are stored.

4. ReFS forensic methodology

Since the Change Journal records all changes based on file or directory, like $UsnJrnl, analyzing the Change Journal is an intuitive method for a forensic analyst. Though the Change Journal has a different structure from the $UsnJrnl, they have similar information such as USN, timestamp, offset, file name, etc. Therefore, technical know-how to examine the $UsnJrnl may valid to analyze the Change Journal. However, in practice, the forensic investigators are unlikely to encounter the Change

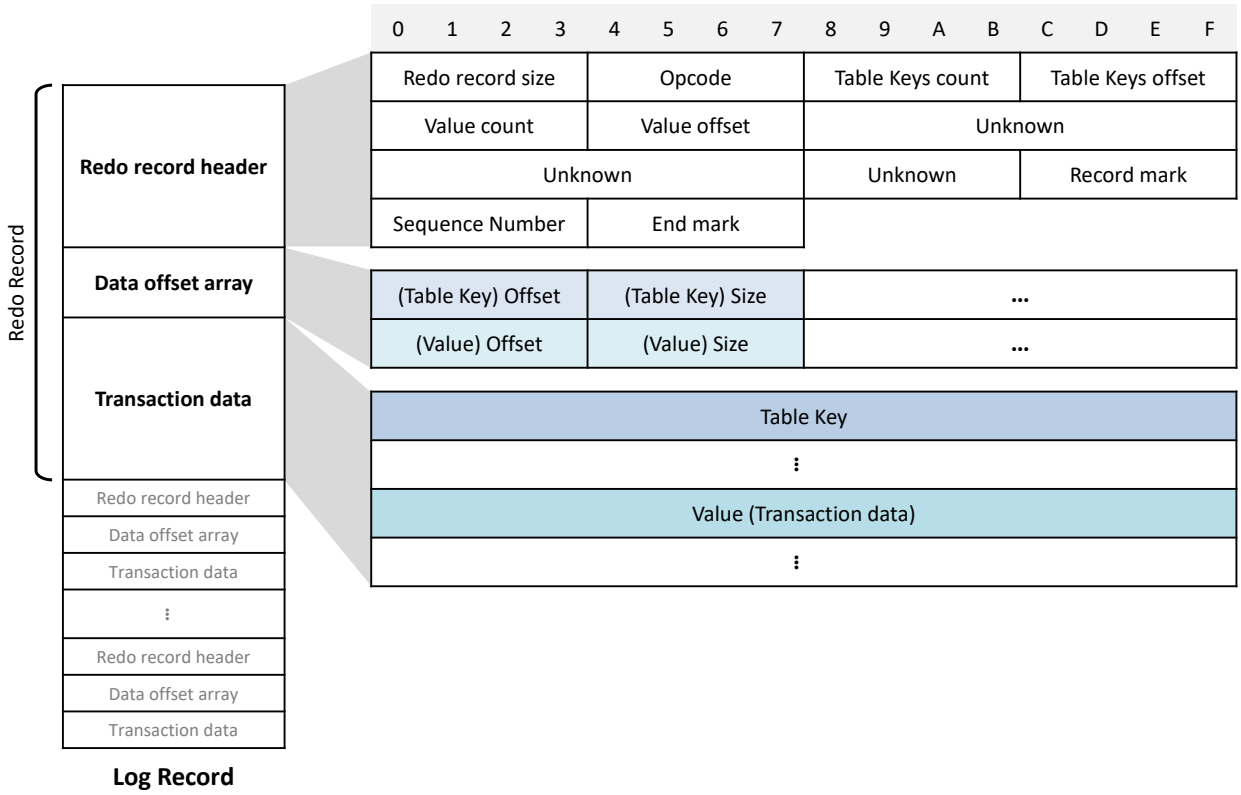

Figure 8: Redo Record structure

Journal as it is deactivated by default in the current ReFS version 3.4.

On the other hand, the Logfile is activated by default. It records all changes of the file system, but it uses transaction and operation level. When a simple event such as file creation occurs, many transactions are generated and recorded to the Logfile. Thus, inferring past events by analyzing the transactions is important for forensic investigators.

4.1. Tracking user's behaviour through analyzing Logfile

In this section, we present a method to track past file operation that a user did on ReFS volume. To identify logging patterns on the Logfile, we experiment as follow: file creation, update, renaming, move, copy, and deletion and directory creation, renaming, move, copy, and deletion. We collected the Logfile and then analyzed transactions after event occurrence.

Based on the result of the experiment, we draw finite-state machine (FSM) about operation patterns when the event occurs. The following is a description of states of the FSM described in Fig. 9 and Fig. 10.

- **I: Initial** - The initial state
- **O: Open** - The sate of getting handle
- **W: Write** - The state of writing row of table of file or directory
- **U: Update** - The state of updating table of file or directory

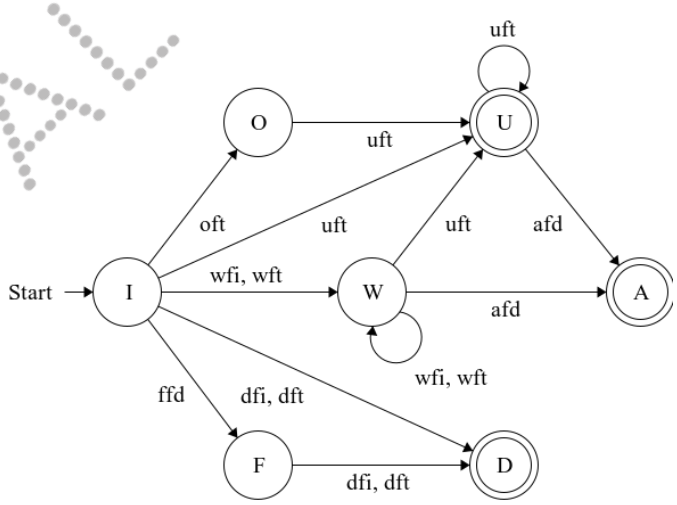

Figure 9: A Finite State Machine of ReFS events with file

- **D: Delete** - The state of deleting row of table of file or directory
- **A: Allocate** - The state of allocating file data on disk
- **F: Free** - The state of deallocating file data on disk

In the FSM, there are several inputs, which also can be described as execution, are recorded to transaction logs. The following is a description of inputs of the FSM.

Table 3: ReFS Redo opcode and operation

Opcode	Redo Operation	Opcode	Redo Operation
0x00	Open Table	0x10	Redo Value as Key
0x01	Redo Insert Row	0x11	Redo Add Schema
0x02	Redo Delete Row	0x12	Copy Key Helper
0x03	Redo Update Row	0x13	Redo Add Container
0x04	Redo Update Data with Root	0x14	Redo Move Container
0x05	Redo Reparent Table	0x15	Copy Key Helper
0x06	Redo Allocate	0x16	Redo Cache Invalidation
0x07	Redo Free	0x17	Redo Generate Checksum
0x08	Redo Set Range State	0x18	Redo Container Compression
0x09	Redo Set Range State	0x19	Redo Delete Compression Unit Offsets
0x0A	Redo Duplicate Extents	0x1A	Redo Add Compress Unit Offsets
0x0B	Redo Modify Stream Extent	0x1B	Redo Ghost Extents
0x0C	Redo Strip Metadata Stream Extent	0x1C	Redo Compaction Unreserve
0x0D	Redo Set Integrity		
0x0E	Redo Set Parent Id		
0x0F	Redo Delete Table		

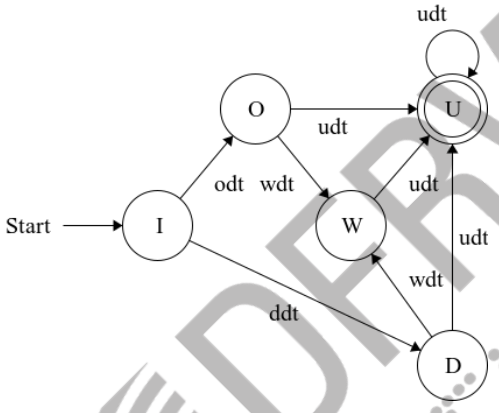

Figure 10: A Finite State Machine of ReFS events with Directory

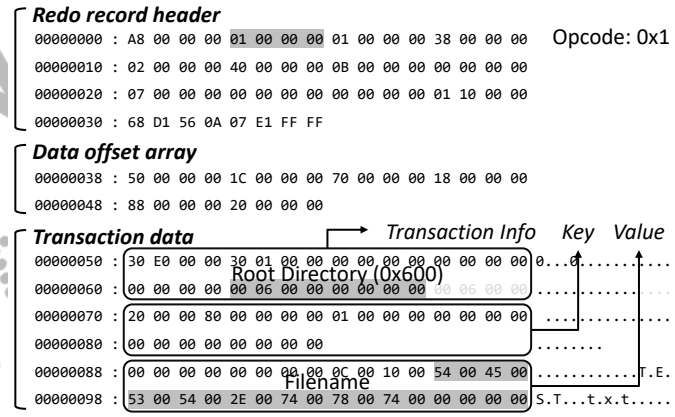

Figure 11: An example of the Redo Record

- **oft** - open table of file
- **wfi** - write file index
- **wft** - write table of file
- **uft** - update table of file
- **dfi** - delete file index
- **dft** - delete table of file
- **afd** - allocate file data
- **ffd** - free file data
- **odt** - open table of directory
- **wdt** - write table of directory
- **udt** - update table of directory
- **ddt** - delete table of directory

Based on the FSMs, we figured out operation patterns of file system event such as file creation, deletion, update, etc. For example, when creating a file, operations seen in Table 4 are generated sequentially. Each transaction has their opcode and data, as seen in Fig. 11. The figure shows an example of the Redo Record that opcode is 0x1 (Redo Insert Row). Some record contains the name or time information of the created file because the transaction data applied to the page of metadata file is stored in key-value. The Fig. 11 is a screenshot corresponding to the first operation of Table 4; in this example, the name of file "TEST.txt" is identified. Operation patterns of events including file creation are described in Table 5.

Table 4: Operation pattern of file creation

Operation (opcode)	Description	Remarks
Insert Row (0x01)	Add file index to table of directory	The name of created file
Update Data with Root (0x04)	Update table of directory	-
Value as Key (0x10)	Set value of file index key in table of directory	-
Open Table (0x00)	Create table of file	The name of created file
Update Data with Root (0x04)	Update table of file	The time information of created file
Insert Row (0x01)	Add attribute to table of file	-
Open Table (0x00)	Access table of file	The name of created file

Table 5: Operation patterns of events

Event	Operation sequence (opcode)
File creation	0x01 → 0x04 → 0x10 → 0x00→ 0x04→ 0x01→ 0x00
File deletion	0x0F→ 0x02→ 0x0F→ 0x02→ 0x04
File content modification	0x06→ 0x04→ 0x04→ 0x04→ 0x04→ 0x08
File renaming	0x02→ 0x05→ 0x01→ 0x04→ 0x04
Directory creation	0x00→ 0x00→ 0x04→ 0x10→ 0x01→ 0x01→ 0x01→ 0x0E→ 0x03→ 0x04
Directory deletion	0x02→ 0x0F→ 0x02→ 0x0F→ 0x12→ 0x04
Directory renaming	0x02→ 0x02→ 0x01→ 0x01→ 0x04

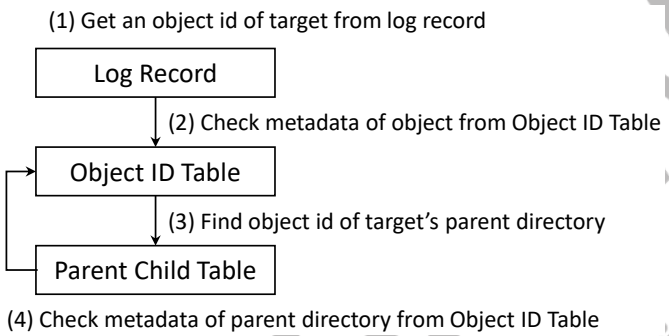

Figure 12: Identifying full path using the Object ID Table and Parent Child Table

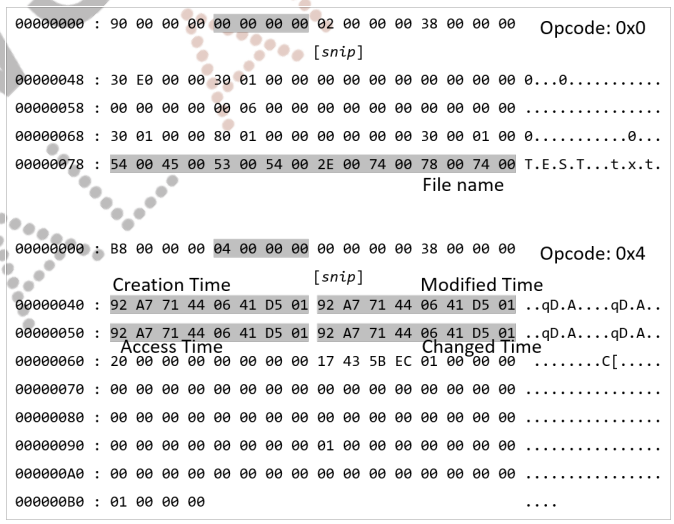

Figure 13: The fourth and fifth Redo Records generated by file creation event

4.2. *Identifying path of file & directory*

In transaction data of the example of Fig. 11, an object ID is stored. It means that table of a directory that has the object ID is updated. Note, an object ID of root directory is 0x600 in ReFS; the fact that the file named "TEST.txt" is created on root directory can be found. If the object ID is some other value, referring to the Object ID Table and Parent Child Table, the full path can be taken. Fig. 12 shows the flowchart describing how to identify the full path of the targeted file or directory.

4.3. *Analyzing transaction time*

Identifying the time when an event occurred is important for forensic investigators. There is no specific field that records the time in the Log Record, however, the time can be verified indirectly by exploring the Redo Record that updates time information of file or directory. Table 6 summarizes methods to infer the event time. To concentrate more on our research, we present an example of verifying the event time from the Redo Record.

Table 6: Method to identify event time (The creation, modified, and changed time are stored in transaction data of redo record)

Event	Event time
File creation	Creation time in the 5th record
File deletion	Changed time in the 5th record
File content modification	Modified time in the 4th record
File renaming	Changed time in the 4th record
Directory creation	Creation time in the 3th record
Directory deletion	Changed time in the 6th record
Directory renaming	Changed time in the 5th record

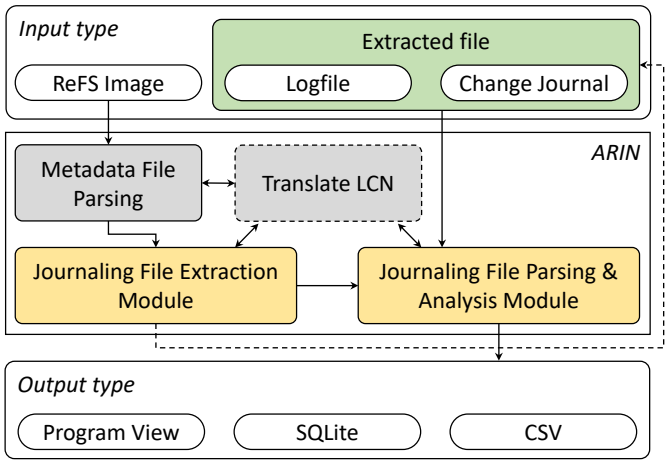

Figure 14: The process of the ARIN

Table 7: The log of the experiment (The first 10 records)

Time	The user's behavior
2019-11-07 16:24	Volume mount
2019-11-07 16:25	Activate Change Journal
2019-11-07 16:26	Create file (TEST.txt)
2019-11-07 16:27	Write file content
2019-11-07 16:27	Copy & paste file (TEST-copy.txt)
2019-11-07 16:27	Rename copy file (TEST2.txt)
2019-11-07 16:28	Create directory (TESTDIR)
2019-11-07 16:28	Move file (TEST.txt → TESTDIR)
2019-11-07 16:28	Delete file (TEST2.txt)
2019-11-07 16:28	Delete Directory (TESTDIR)

Fig. 13 shows two Redo Records generated when creating a file. They correspond to the fourth and fifth operation of Table 4. In the second transaction data of Fig. 13, the time information used to update metadata of the file is recorded. In this manner, the event time can be identified.

5. Implementation and Experiment

Based on our study, we developed a forensic tool, named Awesome ReFS Investigation tool (ARIN), that collects and analyzes the Change Journal and the Logfile on an image of ReFS volume. The application is implemented by python 3.6.8 and PyQt5. It supports ReFS version 3.x.

A diagram of ARIN's process is seen as Fig. 14. There are two types of input: the ReFS image and the journaling files including the Change Journal and the Logfile. When the ReFS image is entered, the ARIN parses the journaling files and then analyzes them. If a forensic investigator already has the Change Journal and the Logfile, the investigator has only to enter them. Fig. 15 shows a screenshot of the ARIN. As seen in the screenshot, an investigator can identify event history and timestamp parsed from the Logfile.

In order to assess the performance of our implementation, we used experimental ReFS volume. Although there are public images like [13], they are not suitable to verity our implementation because timeline of user's action is not provided. For that reason, we produced sample image to test our methodology. After creating a vhd file, we formatted the vhd with ReFS. We also activated the Change Journal using fsutil to compare with the result of analyzing the Logfile. After experimental setup, an experimenter worked on some files and the experimenter's works were recorded simultaneously (See Table 7).

We analyzed the user's behaviour using our developed tool, and then compared between the log of the experiment regarded as an answer and the result of the tool. As a result, we were able to identify all past behaviours about file operation. We also verified our method to infer past file operations based on transaction patterns from the Logfile, by comparing with USN records of the Change Journal. As seen in Fig. 16, we could identify that the timestamp and file operation estimated by our tool are exactly matched with the result of analyzing the Change Journal. It refers to the fact that our study can be very useful for forensic examiners, even if the Change Journal is deactivated.

6. Conclusion

File system forensics is essential to identify file and folder, time information, and recover deleted data. Especially, through analyzing journaling of file system, a forensic examiner can reveal user's past behaviour such as file creation, update, deletion, and move on a volume. In this paper, we studied ReFS journaling to provide advanced methodology of ReFS investigation.

By kernel reverse engineering, we introduced Logfile and Change Journal as novel forensic artifacts of ReFS. We described internal structure of Logfile and Change Journal, and identified difference between the novel artifacts of ReFS and famous artifacts, $Logfile and $UsnJrnl, of NTFS. In this process, material facts were found out; 1) the internal structure of Logfile is totally different from $Logfile, 2) Change Journal has similar structure of $UsnJrnl, however, unlike $UsnJrnl, it is rarely impossible to recover deleted USN journal from unallocated area as Change Journal use circular buffer, and 3) only Logfile is activated by default when a volume is formatted by ReFS, therefore, forensic analysis of Logfile is crucial for an investigator. We also proposed ReFS forensic methodology based on tracking user's past behaviour and event time from transactions of the new artifacts. Based on the methodology, we developed a tool named Awesome ReFS Investigation tool (ARIN), which is currently the only tool for exploring the ReFS journal. The result of this study can provide a significant contribution to digital forensics and offers forensic examiners practical assistance in their investigation.

References

[1] Andrew, H., 2015. Forensic investigation of microsoft's resilient file system (refs).
URL `http://resilientfilesystem.co.uk/`

[2] Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley.

[3] Cho, G., 2013. A computer forensic method for detecting timestamp forgery in ntfs. Computers & Security 34, 36–46.

[4] Cho, G.-S., Rogers, M. K., 2011. Finding forensic information on creating a folder in $logfile of ntfs. In: International Conference on Digital Forensics and Cyber Crime. Springer, pp. 211–225.

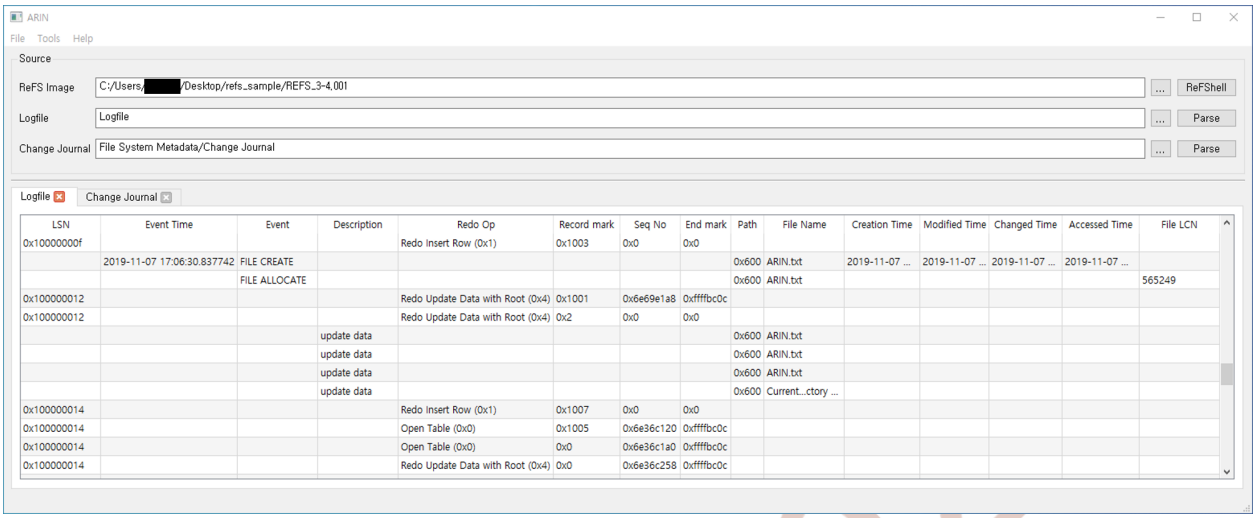

Figure 15: Screenshot of the ARIN

LSN	Event Time	Event	Description	Redo Op	Record mark	Seq No	End mark	Path	File Name
	2019-11-07 16:26:08.649324	FILE CREATE						0x600	TEST.txt
			update data					0x600	TEST.txt
			update data					0x600	TEST.txt
			update data					0x600	TEST.txt
			update data					0x600	Current...ctory ...
0x100000016				Redo Insert Row (0x1)	0x1007	0x0	0x0		
0x100000016				Open Table (0x0)	0x1005	0x6c160120	0xffffbc0c		
0x100000016				Open Table (0x0)	0x0	0x6c1601a0	0xffffbc0c		
0x100000016				Redo Update Data with Root (0x4)	0x0	0x6c160258	0xffffbc0c		

USN	Timestamp	File Name	Reason
0x0	2019-11-07 16:26:08	TEST.txt	FILE CREATE
0x60	2019-11-07 16:26:08	TEST.txt	FILE CREATE | CLOSE
0xc0	2019-11-07 16:26:08	TEST.txt	BASIC INFO CHANGE
0x120	2019-11-07 16:26:08	TEST.txt	BASIC INFO CHANGE | CLOSE
0x180	2019-11-07 16:26:09	$RECYCLE.BIN	FILE CREATE
0x1e8	2019-11-07 16:26:09	$RECYCLE.BIN	FILE CREATE | CLOSE

Figure 16: Comparison between the result of the Logfile (above) and the Change Journal (below)

[5] Georges, H., 2018. Resilient filesystem. Master's thesis, NTNU.

[6] Junghoon, O., 2013. Ntfs log tracker.
URL http://forensicinsight.org/wp-content/uploads/2013/06/F-INSIGHT-NTFS-Log-TrackerEnglish.pdf,2013

[7] Lees, C., 2013. Determining removal of forensic artefacts using the usn change journal. Digital Investigation 10 (4), 300–310.

[8] Metz, J., 2013. Resilient file system (refs).
URL https://github.com/libyal/libfsrefs/blob/master/documentation/ResilientFileSystem(ReFS).pdf

[9] Microsoft, 2012. Building the next generation file system for windows: Refs.
URL https://blogs.msdn.microsoft.com/b8/2012/01/16/building-the-next-generation-file-system-for-windows-refs/

[10] Microsoft, 2018. Usn_record_v2 structure.
URL https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-usn_record_v2

[11] Microsoft, 2018. Usn_record_v3 structure.
URL https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-usn_record_v3

[12] Nordvik, R., Georges, H., Toolan, F., Axelsson, S., 2019. Reverse engineering of refs. Digital Investigation 30, 127–147.

[13] Nordvik, R., Toolan, F., Axelsson, S., Georges, H., 2019. Data for: Reverse engineering of refs.
http://dx.doi.org/10.17632/938fjzx8rc.1.

[14] Prade, P., Groß, T., Dewald, A., 2020. Forensic analysis of the resilient file system (refs) version 3.4. Forensic Science International: Digital Investigation 32, 300915.

[15] Prade, P., Groß, T., Dewald, A., December 2019. Forensic analysis of the resilient file system (refs) version 3.4. Tech. rep., Friedrich-Alexander-Universit at Erlangen-Nurnberg, Dept. of Computer Science.

[16] Uijtewaal, F., van Prooijen, J., 2016. Usnjrnl parsing for file system history project report. University of Amsterdam.