ISSN : 1976-5304
https://kdfs.jams.or.kr

ReFS 저널링 파일 분석

이 선 호, 최 호 용, 박 정 흠, 이 상 진
고려대학교 정보보호대학원

Analysis of ReFS Journaling File

Seonho Lee, Hoyong Choi, Jungheum Park, Sangjin Lee
Graduate school of Information Security, Korea University

요 약

포렌식 조사에서 파일시스템의 저널링 파일을 분석하는 것은 사용자의 행위를 파악하고 추적하기 위해 꼭 필요하다. 2012년에 공개된 ReFS는 저널링 파일뿐만 아니라 파일시스템 전체 구조에 대한 분석도 미흡하다. 본 논문에서는 ReFS의 저널링 파일인 Change Journal과 Logfile을 분석하여 ReFS에서 발생한 사용자 행위를 추적하는 방안을 제시한다. 특히, Logfile에서 특정 행위에 따른 트랜잭션 로그 패턴이 어떻게 나타나는지 실험을 통해 분석했다. 또한, 분석한 트랜잭션 로그 패턴을 Logfile에서 식별하는 것으로 ReFS에서 발생한 파일 작업, 작업이 발생한 시점을 특정할 수 있으며, 이를 포렌식 조사에 활용할 수 있음을 제안한다.

주제어 : 디지털 포렌식, 파일시스템, ReFS, 트랜잭션 저널링, USN 저널링

ABSTRACT

Analyzing the filesystem's journaling files in a forensic investigation is essential to understanding and tracking user behavior. Although much research has been conducted on ReFS released in 2012, the analysis of journaling file and file system structure is insufficient. In this paper, we propose a method for tracking user behavior in ReFS by analyzing ReFS's journaling files, Change Journal and Logfile. In particular, we analyzed through the experiment how the transaction log pattern according to the specific behavior in the logfile. In addition, by identifying the analyzed transaction log pattern in the logfile, it is possible to specify the file operation occurred in ReFS and when the operation occurred, which can be used for forensic investigation.

Key Words : Digital Forensics, File System, ReFS, Transaction Journaling, USN Journaling

Ⅰ. 서 론

2012년 마이크로소프트는 윈도우 8과 윈도우 서버 2012의 출시와 함께 신규 파일시스템인 ReFS(Resilient File System)를 공개하였다[1]. ReFS는 대용량 저장장치에 특화되어 데이터의 무결성을 보장하고 손상되었을 경우 복원되도록 설계된 파일시스템이다[2]. ReFS는 지속적으로 업데이트되고 있으며 최신 버전의 윈도우 10과 윈도우 서버 2016에서도 지원하고 있다. 최근 출시된 윈도우 서버 2019에서도 ReFS를 지원하고 있으며, 대용량의 저장공간과 데이터 무결성을 필요로 하는 시스템에서 ReFS의 사용이 예상된다.

시스템에서 발생한 과거의 행위를 조사하기 위해 파일시스템 내 파일의 생성·수정·삭제와 같은 작업을 추적하는 것은 중요하다. 현재 윈도우 시스템에 대한 파일시스템 분석과 연구는 NTFS 위주로 진행되고 있는[3], 반면 ReFS에 대한 연구는 상대적으로 부족한 편이다. EnCase에서도 ReFS 버전 1.2에 대해서만 부분적으로 분석할 수 있으며 FTK나 Autopsy에서는 분석할 수 없다. ReFS는 정식으로 윈도우 운영체제에서 지원하는 파일시스템이므로 포렌식 조사 과정에서 ReFS로 포맷된 볼륨을 분석할 경우가 발생할 수 있다. 따라서 ReFS에 대한 구조와 아티팩트 분석에 관한 연구가 필

※ 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-01000, 디지털 포렌식 통합 플랫폼 개발)
- Received 06 September 2019, Revised 19 September 2019, Accepted 26 September 2019
- 제1저자(First Author) : Seonho Lee (Email : seonho@korea.ac.kr)
- 교신저자(Corresponding Author) : Sangjin Lee (Email : sangjin@korea.ac.kr)

요하다. 특히 ReFS에는 파일시스템의 변경 사항을 기록하는 Change Journal이 존재하며, 버전 3부터는 트랜잭션 저널링 기능을 담당하는 Logfile이 추가되었다. 각 파일은 NTFS의 $UsnJrnl과 $LogFile과 유사한 특성이 있으며, ReFS 볼륨에서 발생한 파일시스템 작업을 파악할 수 있기 때문에 포렌식 아티팩트로 활용할 가치가 크다. 본 논문에서는 Change Journal과 Logfile에 대한 구조를 분석하고, 포렌식 조사에 활용할 수 있는 방안을 제시한다.

Ⅱ. 기존 연구 분석

ReFS에 대한 기본적인 구조와 메타데이터는 기존에 연구된 바 있다[4, 5, 6, 7, 8]. 대부분의 기존 연구[4, 6, 7]들은 ReFS 버전 1.2에 대해 분석하였다. 2015년에 Andrew는 ReFS 버전 1.2와 NTFS를 비교하고 파일 생성·수정·권한 변경·이름 변경과 같은 실험을 통해 파일과 디렉터리의 메타데이터를 분석한 결과를 발표하였다[4]. 이와 별개로 ReFS 버전 1.2의 구조를 분석하여 개발된 도구와 문서도 존재한다[9]. 한편, ReFS 버전 1.2의 파일시스템 메타데이터 파일에 대한 분석도 진행되었다[7]. 이 연구에서는 주요 시스템 파일에 접근하기 위한 메타데이터 파일을 분석하였으며, 파일의 할당 정보를 관리하는 $Allocator_Lrg, $Allocator_Med, $Allocator_Sml, 그리고 디렉터리 간 부모-자식 관계를 정의하는 $Object 파일을 분석하였다.

2018년에는 ReFS에서 데이터를 저장하는 방식과 파일시스템의 메타데이터 구조를 이용하여 삭제된 데이터를 복구하는 기술에 대한 연구가 발표되었다[6]. 이 연구에서는 ReFS의 주요 버전(버전 1.2, 버전 3)에서 파일시스템 메타데이터 구조를 분석하였다. 아울러 ReFS 버전 3에서 추가된 메타데이터 파일인 Container Table의 구조를 밝혀냈으며, 파일시스템 내 주소 체계에 대해 연구하였다. 다만, 메타데이터 구조 내 필드의 의미만 분석하였으며 전반적인 ReFS의 데이터 구조에 대한 연구는 이루어지지 않았다. 2019년에는 ReFS를 역공학한 연구[8]가 발표되었다. 이 연구에서는 역공학을 통해 밝혀낸 ReFS 구조를 기반으로 메타데이터 카빙과 데이터 재구성을 위한 복구 기법을 제안하였다. 또한, ReFS의 동작 방식이 파일 복구에 관해 영향을 미치는 부분을 분석하고, 이를 NTFS와 비교하여 어떤 차이가 있는지 논하였다. 하지만 ReFS 버전 3부터 확인된 트랜잭션 저널링에 대한 분석은 진행되지 않았다.

Ⅲ. ReFS 데이터 구조

파일시스템 데이터 구조는 파일시스템에서 데이터를 저장하고 관리하기 위한 형태와 방식을 의미한다. 예를 들어 NTFS의 MFT 엔트리 구조와 내부 속성 구조는 NTFS의 대표적인 데이터 구조이다. ReFS에서는 메타데이터 파일 내에 데이터를 저장하기 위해 [그림 1]과 같이 페이지(Page) 구조를 사용한다. 페이지 구조는 ReFS에서 가장 상위 개념의 데이터 구조이며 페이지 헤더와 테이블로 구성된다. 페이지 헤더는 ReFS 버전에 따라 다른 크기를 가지며 해당 페이지가 담고 있는 메타데이터 파일의 종류를 나타내는 Object ID를 비롯하여 페이지와 관련된 정보가 저장되어 있다.

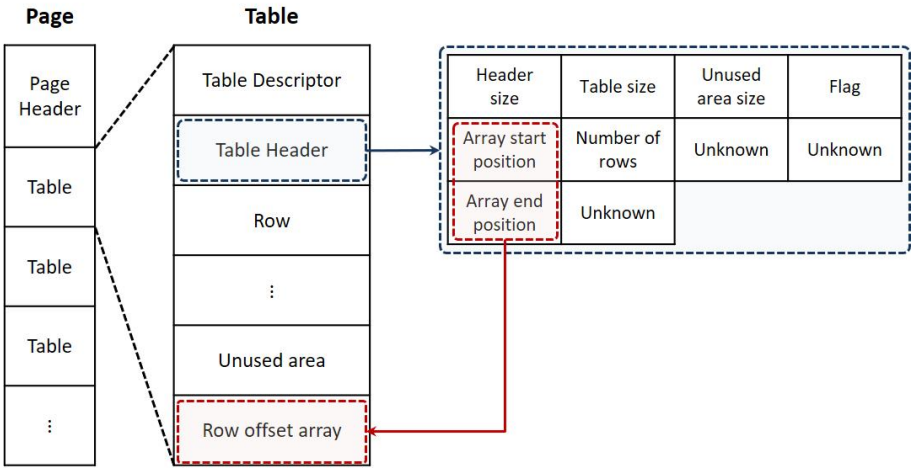

그림 1. Page와 Table 구조
Fig. 1. Page and Table Structure

테이블은 내부에 여러 개의 Row를 가지며 B+Tree 형태로 Row를 관리한다. 만약 테이블 내에 존재하는 Row의 개수가 계속 증가하여 하나의 페이지에 담지 못하면 별도의 공간에 페이지와 테이블을 할당해서 Row를 저장한다. 이때 새로 할당되는 테이블은 자식 테이블(Child Table)이라 한다. 테이블의 구조는 테이블 설명자(Table Descriptor), 테이블 헤더(Table Header), Row, Row 오프셋 배열(Row offset array)로 구성되어 있다. 테이블 설명자는 테이블에 존재하는

Row의 총 개수, 자식 테이블의 개수를 저장하고 있다. 테이블 헤더는 테이블 내 Row를 찾아가기 위한 배열의 위치 정보와 자식 테이블에 대한 존재 여부, 테이블 내 사용되지 않는 영역에 대한 정보를 가지고 있다. Row 오프셋 배열은 테이블 내에 존재하는 Row의 위치 정보를 배열 형태로 가지고 있다. 이때 Row의 위치 정보는 테이블 헤더를 기준으로 한다.

Row는 메타데이터 파일의 목적에 따라 데이터를 저장하는 데이터 구조이다. Row 내부 구조는 〔그림 2〕와 같이 헤더(Header), 키(Key), 값(Value)으로 구성되며 값 대신 테이블이 위치할 수 있다. Row의 Key는 테이블에 구현된 B+Tree의 키로 사용되며, 메타데이터 파일에서 저장하고 관리하고자 하는 데이터는 Row의 Value에 저장된다.

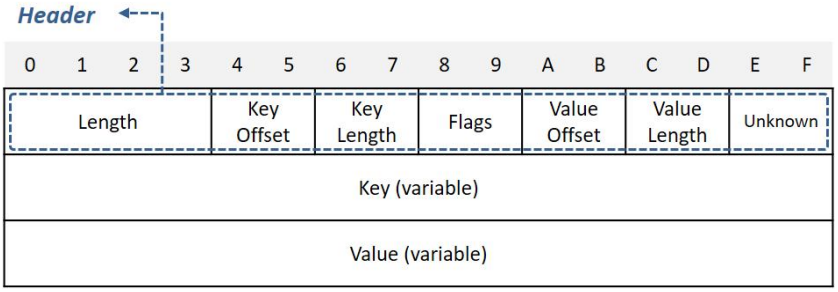

그림 2. Row 내부 구조
Fig. 2. Row internal structure

예를 들어 디렉터리는 디렉터리에 존재하는 파일이나 하위 디렉터리의 메타데이터를 디렉터리 테이블[1] 내 Row에 각각 저장하여 관리한다. 이때 파일이나 하위 디렉터리의 메타데이터는 각 Row의 Key와 Value에 〔그림 191〕과 같이 저장된다. Key에 저장되는 Type 필드의 값에 따라 파일과 하위 디렉터리를 구분할 수 있다. 아울러 Key에 저장되는 파일과 하위 디렉터리 이름은 디렉터리의 인덱스와 같은 역할을 한다. 본 논문에서는 파일에 대한 메타데이터를 저장하는 Row는 File Record, 하위 디렉터리에 대한 메타데이터를 저장하는 Row는 Directory Record라고 지칭한다.

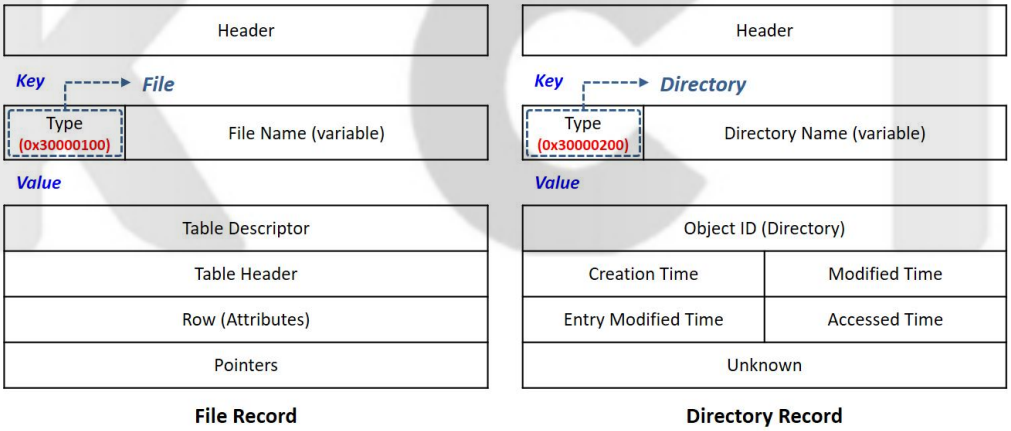

그림 3. 디렉터리 내 테이블의 Row에 저장되는 File Record와 Directory Record 구조
Fig. 3. File Record and Directory Record structure stored in row of table in directory

File Record에서 Value는 테이블 구조로 되어 다시 여러 개의 Row를 가질 수 있다. File Record 테이블[2]의 Row에는 파일에 대한 특정 유형의 속성(Attribute)을 저장한다. 일반적으로 데이터 속성이 존재하여 파일의 실제 데이터가 할당된 위치 정보를 가지고 있으며, ADS가 있는 경우 별도의 ADS 속성을 추가로 가지고 있다. 이러한 File Record의 형태는 NTFS의 MFT 엔트리에서 $DATA 속성이 존재하는 것과 유사하다. Row에 들어가는 데이터 속성과 ADS 속성의 구조는 〔그림 4〕와 같으며, Key에 존재하는 ATTR Type 필드의 값에 따라서 속성의 유형을 구분할 수 있다. ATTR Type 필드 값이 0x80이면 데이터 속성이며, 0xB0이면 ADS 속성이다. ReFS에서 ADS를 별도의 속성으로 구분하는 것은 NTFS에서 ADS가 $DATA 속성으로 들어갔던 것과 차이가 있다.

1) 디렉터리는 메타데이터 파일로 페이지 구조를 가진다. 여기서 디렉터리 테이블은 디렉터리가 가지는 페이지 구조 내 테이블을 가리킨다.

2) File Record의 Value 내 테이블을 의미한다.

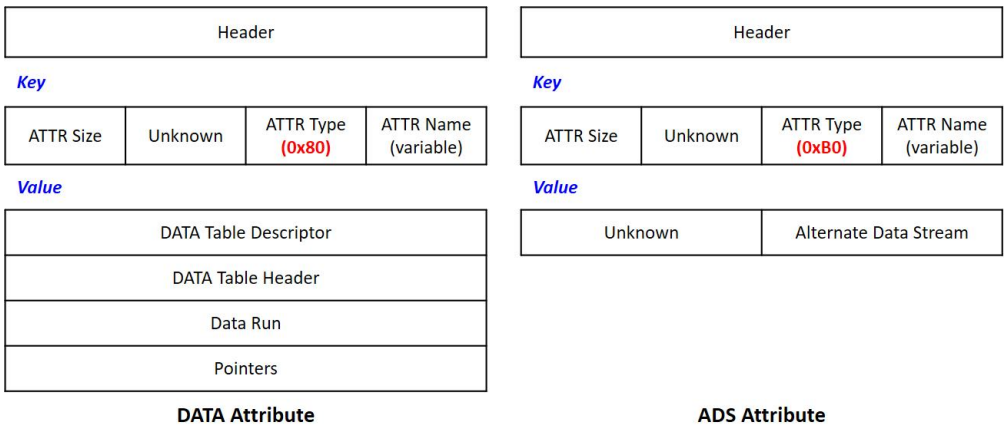

그림 4. Data Attribute와 ADS Attribute 구조
Fig. 4. Data Attribute and ADS Attribute Structure

디렉터리에 존재하는 파일에 대해서는 File Record 외에 파일 ID 정보가 추가로 디렉터리 테이블의 Row에 할당된다. 본 논문에서는 이러한 Row를 File Index로 부른다. ReFS에서 파일 ID는 디렉터리의 Object ID와 디렉터리 내 파일의 순서 번호가 조합된 128bit의 크기로 구성된다. 파일의 순서 번호는 파일이 디렉터리에 생성된 순서대로 부여되는 번호이다. 예를 들어 루트 디렉터리에 두 번째로 생성된 파일의 파일 ID는 0x00000000000006000000000000000002이다. 이는 루트 디렉터리의 Object ID 0x600과 파일의 순서 번호 0x2를 조합한 것이다. File Index의 구조는 〔그림 5〕와 같다. ReFS 버전 3 이전에서는 부모 디렉터리의 Object ID도 File Index에 포함되었으나, ReFS 버전 3부터는 파일의 순서 번호만 저장한다.

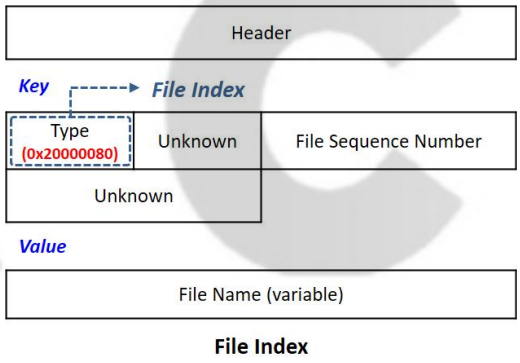

그림 5. ReFS 3.4에서 File Index 구조
Fig. 5. File Index Structure in ReFS 3.4

Directory Record의 Value에는 하위 디렉터리에 대한 Object ID와 하위 디렉터리의 시간 정보가 들어가 있다. Directory Record의 Object ID는 Object Table[3]에서 하위 디렉터리의 위치 정보를 찾기 위해 사용된다. 하위 디렉터리의 위치를 찾아가면 페이지 구조를 가지는 디렉터리를 확인할 수 있다.

Ⅳ. ReFS 저널링 파일

ReFS 버전 3에는 파일시스템을 구성·관리하기 위한 메타데이터 파일뿐만 아니라 포렌식 아티팩트로 활용할 수 있는 파일시스템 변경 저널링 파일인 Change Journal과 트랜잭션 저널링 파일인 Logfile이 존재한다. 각 파일시스템 저널링 파일은 서로 다른 경로에 존재하며, ReFS 내 파일의 생성·수정·삭제와 같은 행위와 관련된 데이터가 저장되어 있다. 트랜잭션 저널링 파일은 트랜잭션 저널링을 제공하는 다른 파일시스템에서도 찾을 수 있다. 트랜잭션 저널링을 제공하는 다른 파일시스템과 ReFS의 트랜잭션 저널링의 주요 기능과 특징을 비교한 결과는 〔표 1〕과 같다. ReFS 트랜잭션 저널링은 EXT 파일시스템과 동일하게 Undo 작업을 저널링하지 않는다는 점에서 NTFS와 차이가 있다. 이 외의 나머지 특징은 다른 파일시스템의 저널링과 동일하다.

3) Object Table은 메타데이터 파일과 모든 디렉터리에 대해 위치 정보를 가지고 있는 메타데이터 파일이다.

표 1. 파일시스템 간 트랜잭션 저널링 기능 비교표
Table 1. Transaction journaling function comparison table between file systems

	ReFS	NTFS	EXT FS
Redo operation	○	○	○
Undo operation	×	○	×
Timestamp	×	×	×
Circular buffer	○	○	○

NTFS의 저널링 파일 $UsnJrnl과 $LogFile은 분석[10, 11]되어 포렌식 조사에 활용되고 있으나 ReFS의 저널링 파일은 분석이 되지 않았다. 본 절에서는 ReFS의 저널링 파일인 Change Journal과 Logfile에 대한 상세한 구조와 내부에 저장되는 정보를 설명한다.

4.1 Change Journal

ReFS 볼륨의 루트 디렉터리 하위에 File System Metadata 디렉터리가 있다. File System Metadata 디렉터리는 루트 디렉터리 테이블에 Directory Record로 존재하지는 않지만, Parent Child Table[4]에서 File System Metadata 디렉터리가 루트 디렉터리 하위에 속한 것을 알 수 있다. File System Metadata 디렉터리 하위에는 Change Journal, Reparse Index, Security Descriptor Stream, Volume Direct IO File로 총 4개의 메타데이터 파일이 존재한다. 각각의 파일들은 페이지 구조로 파일시스템 내 파일을 관리하기 위한 메타데이터를 저장하고 있다. Change Journal은 파일시스템 내 변경 사항을 기록하는 파일로 NTFS의 $UsnJrnl과 동일하게 USN 레코드[5]를 기록한다. 즉, ReFS 볼륨의 파일이나 디렉터리가 변경되면 Change Journal은 변경 사항에 대한 설명과 파일 또는 디렉터리의 이름을 업데이트한다. 따라서 Change Journal을 분석하면 과거에 ReFS에서 발생한 파일 작업에 대해 추적할 수 있다.

Change Journal은 NTFS의 $UsnJrnl과 구조상 차이가 있다. $UsnJrnl은 기본으로 할당된 영역을 전부 사용한 이후 데이터를 추가할 때 이전에 할당된 영역을 sparse 영역으로 만든다[12]. ReFS에서도 sparse 기능을 지원하지만, Change Journal은 sparse 기능을 이용하지 않고 가장 오래된 레코드를 덮어쓰는 원형 버퍼 방식으로 동작한다. $UsnJrnl에서 USN은 파일의 시작을 기준으로 레코드의 오프셋을 의미했지만 Change Journal에서는 단순히 레코드 크기만큼 증가하는 순서 번호이다. 한편 $UsnJrnl에서는 USN_RECORD_V2[13]를 사용한 것과 달리 Change Journal에서는 [그림 6]과 같은 USN_RECORD_V3[14]를 사용한다.

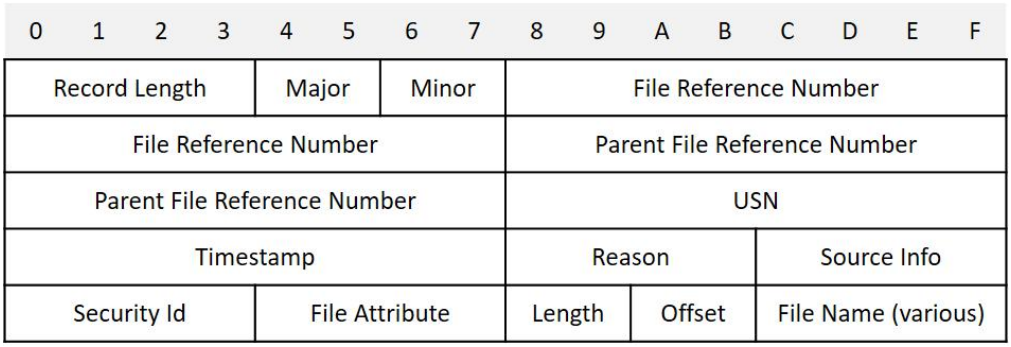

그림 6. USN_RECORD_V3 구조
Fig. 6. USN_RECORD_V3 Structure

USN_RECORD_V2와 USN_RECORD_V3는 FileReferenceNumber 필드를 표현하기 위해 사용하는 비트 수가 64bit에서 128bit로 확장된 차이만 있으며 나머지 필드는 모두 동일하다. 이는 각각의 레코드에서 변경 사항에 대한 설명을 의미하는 reason 필드에서 사용되는 플래그 값이 같음을 의미한다. reason 필드는 파일이나 디렉터리에 대해 발생한 행위를 알 수 있는 중요한 필드이다. 아울러 Timestamp 필드는 그러한 행위가 발생한 시간 정보를 기록하고 있다. 결국, Change Journal의 USN 레코드를 분석하면 과거 특정 시점에 ReFS에서 발생한 파일 작업을 알 수 있다.

4) Parent Child Table은 ReFS 볼륨에 존재하는 디렉터리 간 부모-자식 관계를 관리하는 메타데이터 파일이다.

5) USN 레코드는 파일시스템의 변경 사항을 기록하기 위해 Microsoft에서 정의한 구조이다.

4.2 Logfile

ReFS 버전 3에는 파일시스템 내 메타데이터 파일의 구성에 변경을 가하는 작업이 실패할 경우를 대비하여 트랙잭션을 저널 파일에 기록하는 저널링 기능이 존재한다. 이는 NTFS나 EXT 파일시스템에서 지원하는 저널링 기능과 동일하다. ReFS에서 트랜잭션을 기록하는 저널 파일의 이름은 Logfile이다. ReFS의 Logfile은 NTFS의 저널 파일인 $LogFile과 큰 차이가 있다. NTFS의 $LogFile은 트랜잭션에 대한 Redo/Undo 데이터를 저장하고 있으나, ReFS에서는 Redo 데이터만 Logfile에 기록한다. ReFS는 변경된 메타데이터 파일의 페이지를 기존의 데이터에 덮어쓰는 방식으로 수정하지 않고 새롭게 할당하는 방식인 AOW(Allocate On Write) 방식을 사용한다[1]. 따라서 변경하는 작업 자체가 실패할 경우를 대비해 Redo 데이터만 Logfile에 기록한다. Redo 데이터만을 기록하는 것은 EXT 파일시스템의 저널링과 유사하다. ReFS의 트랜잭션 저널링 동작 방식은 [그림 7]과 같다. 메타데이터 파일의 페이지에서 변경되거나 추가될 각각의 데이터는 Redo 레코드(Redo Record)로 취급된다. 예를 들어 파일 생성 시 디렉터리에 새로 추가되는 Row의 Key-Value 데이터는 하나의 Redo 레코드가 되며, Redo 레코드가 모여 로그 레코드로 Logfile에 저장된다.

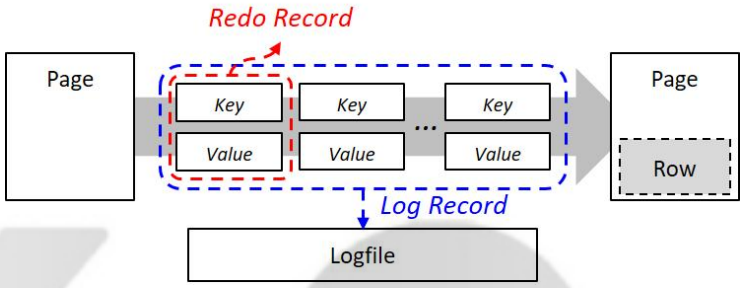

그림 7. ReFS 트랜잭션 저널링
Fig. 7. ReFS Transaction Journaling

Logfile은 Logfile Information Table을 통해 접근한다. Logfile Information Table은 Logfile의 컨트롤(Control) 영역의 시작 위치에 대한 주소 정보를 가지고 있는 메타데이터 파일이다. Logfile은 기본적으로 0x1000 바이트 크기의 엔트리들로 구성되어 있으며 컨트롤 영역과 데이터(Data) 영역으로 구분된다. 이는 NTFS의 $LogFile에서 재시작(Restart) 영역과 로깅(Logging) 영역이 나뉜 것과 유사하다. 하지만 ReFS의 Logfile은 컨트롤 영역과 데이터 영역이 연속되어 할당되어 있지 않으며, 영역 구분 없이 동일한 구조를 가진다. Logfile에 대한 전체적인 구조와 관련 메타데이터 파일 간의 관계는 [그림 8]과 같다.

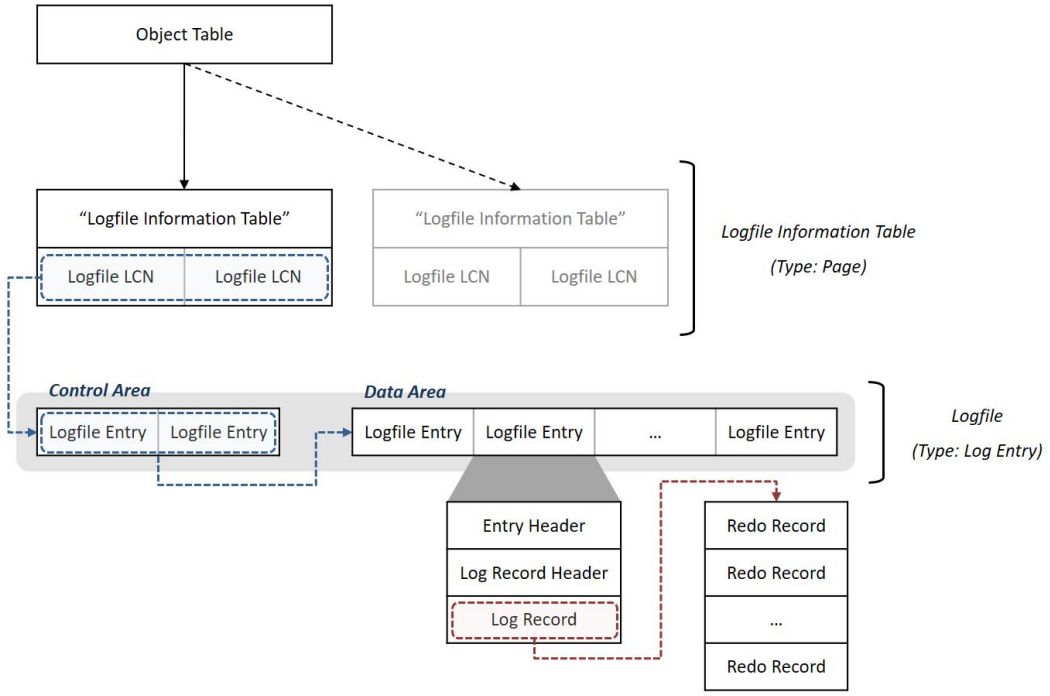

그림 8. Logfile 전체 구조와 관련 메타데이터 파일 간의 관계
Fig. 8. Relationship between the entire logfile structure and related metadata files

Logfile은 ReFS의 다른 메타데이터 파일과 달리 페이지 구조를 사용하지 않는다. Logfile 내부의 엔트리는 [그림 9]와 같이 로그 엔트리의 구조를 가진다. 로그 엔트리는 엔트리 헤더(Entry Header), 로그 레코드 헤더(Log Record Header), 로그 레코드(Log Record)로 구성된다.

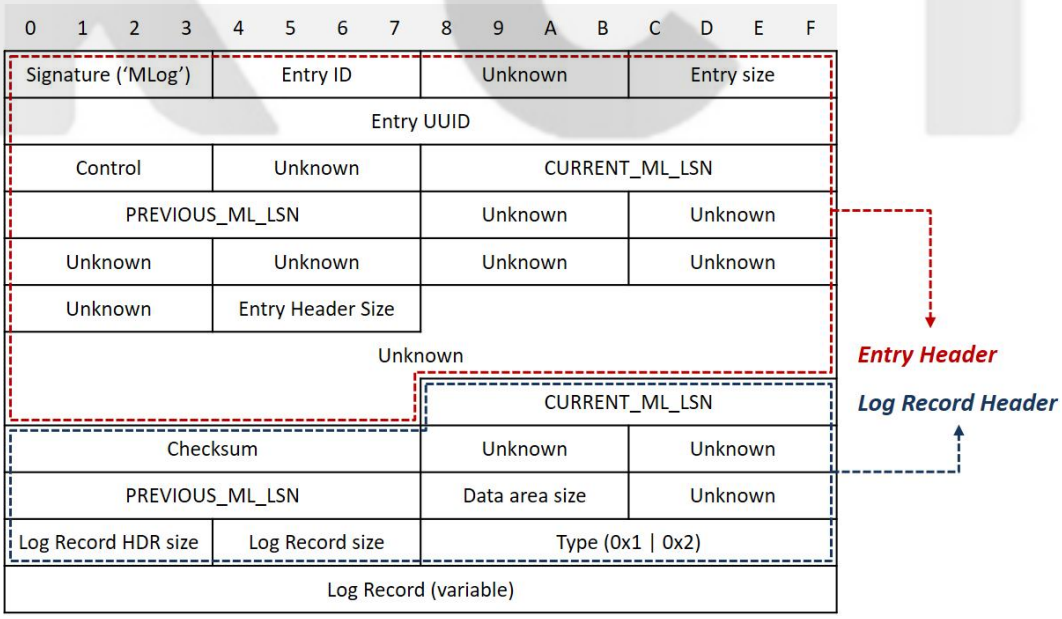

그림 9 ReFS 로그 엔트리 구조
Fig. 9. ReFS log entry structure

엔트리 헤더는 120 바이트 크기로 Logfile에서 로그 엔트리를 식별할 수 있는 정보를 저장하고 있다. 엔트리 헤더의 처음 4 바이트는 'MLog' 시그니처가 할당되어 있어 로그 엔트리임을 확인할 수 있다. 이 외에 엔트리 헤더에는 현재 로그 엔트리의 LSN(Logfile Sequence Number)과 이전 로그 엔트리의 LSN을 저장한다.

로그 레코드 헤더는 56 바이트 크기로 엔트리 헤더의 체크섬 값을 저장하며, 현재와 이전 로그 엔트리의 LSN을 엔트리 헤더와 동일하게 저장한다. 또한, 로그 레코드 헤더는 타입 필드를 가지고 있어 컨트롤 영역의 엔트리와 데이터 영역의 엔트리를 구분하는 역할을 한다.

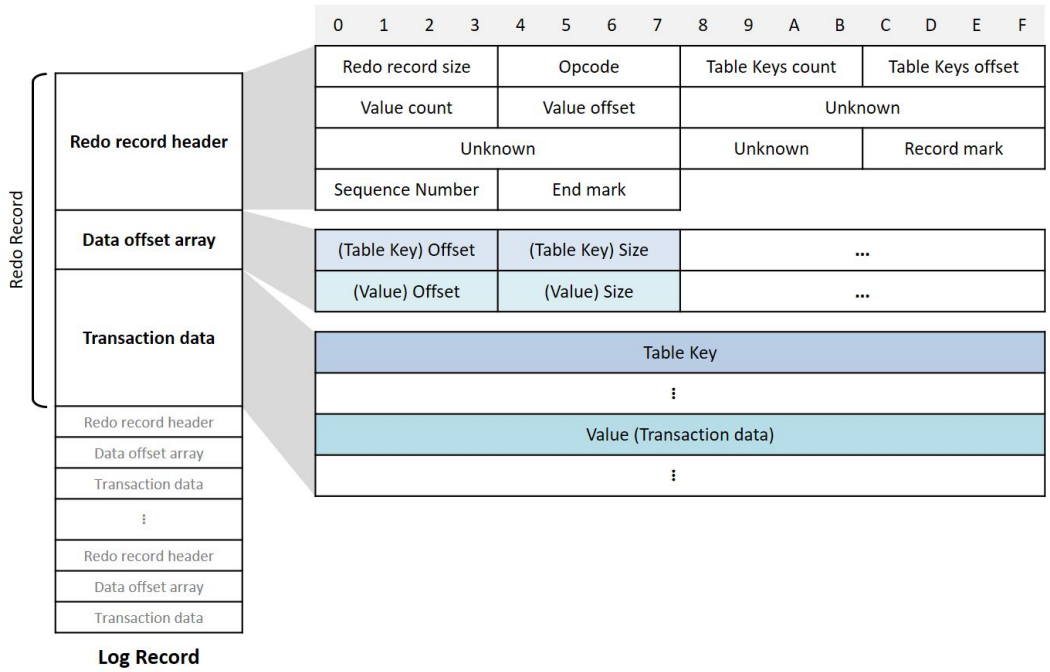

그림 10. 로그 레코드 구조
Fig. 9. Log record structure

로그 레코드에 저장되는 데이터는 컨트롤 영역과 데이터 영역에 따라 다르다. 컨트롤 영역 내 로그 레코드에는 NTFS의 $LogFile에서 재시작 영역과 유사하게 다음 레코드가 할당될 위치 정보나 다음 LSN 정보를 관리하고 있다. 반면 데이터 영역의 로그 엔트리에는 파일시스템의 트랜잭션 데이터가 저장된 Redo 레코드들로 구성된다. 데이터 영역 내 로그 엔트리의 로그 레코드에 대한 자세한 구조는 [그림 10]과 같다.

표 2. ReFS Redo Operation and Opcode
Table 2. ReFS Redo Operation and Opcode

Opcode	Redo Operation	Opcode	Redo Operation
0x0	Open Table	0x10	Redo Value as Key
0x1	Redo Insert Row	0x11	Redo Add Schema
0x2	Redo Delete Row	0x12	Copy Key Helper
0x3	Redo Update Row	0x13	Redo Add Container
0x4	Redo Update Data with Root	0x14	Redo Move Container
0x5	Redo Reparent Table	0x15	Copy Key Helper
0x6	Redo Allocate	0x16	Redo Cache Invalidation
0x7	Redo Free	0x17	Redo Generate Checksum
0x8	Redo Set Range State	0x18	Redo Container Compression
0x9	Redo Set Range State	0x19	Redo Delete Compression Unit Offsets
0xA	Redo Duplicate Extents	0x1A	Redo Add Compress Unit Offsets
0xB	Redo Modify Stream Extent	0x1B	Redo Ghost Extents
0xC	Redo Strip Metadata Stream Extent	0x1C	Redo Compaction Unreserve
0xD	Redo Set Integrity		
0xE	Redo Set Parent Id		
0xF	Redo Delete Table		

Redo 레코드는 Redo 레코드 헤더(Redo record header)와 데이터 오프셋 배열(Data offset array), 트랜잭션 데이터(Transaction data)로 구성된다. Redo 레코드 헤더는 파일시스템 작업이 실패할 경우 재실행(Redo)하기 위한

Opcode와 데이터 오프셋 배열에 대한 위치 정보를 가진다. Opcode는 〔표 2〕와 같이 재실행해야 하는 28개의 파일시스템 작업으로 지정되어 있다. 재실행해야 하는 작업은 원래 파일시스템에서 성공적으로 처리됐어야 하는 작업이다. 따라서 Redo 레코드로부터 과거에 성공적으로 처리된 파일시스템 작업과 데이터를 추적할 수 있다. 예를 들어 디렉터리에 File Record를 추가하는 작업의 실패를 대비한 Redo 레코드(Opcode는 0x1)가 남아있는 경우, 디렉터리에 파일을 생성한 사실을 알 수 있다. 데이터 오프셋 배열에는 재작업에 사용할 트랜잭션 데이터들의 위치 정보와 크기를 저장하고 있다. 위치 정보는 Redo 레코드의 시작 위치를 기준으로 저장하고 있다. 트랜잭션 데이터는 메타데이터 파일의 페이지에 추가·변경되는 데이터나 트랜잭션에 대한 메타데이터를 담고 있다. Opcode는 단순히 재실행해야 하는 작업을 지정한 것으로 실제 어떤 작업이 발생했는지는 트랜잭션 데이터를 함께 분석해야 알 수 있다.

Ⅴ. ReFS 파일 작업 분석

ReFS의 Change Journal과 Logfile은 ReFS 볼륨에서 과거 특정 시점에 발생한 이벤트를 기록한다. 즉, ReFS 볼륨에서 사용자가 한 행위를 조사하기 위한 아티팩트로 Change Journal과 Logfile을 활용할 수 있다. 본 절에서는 파일시스템에서 발생할 수 있는 행위 중 가장 기본적인 파일의 생성과 삭제에 관해 Logfile에 실제로 데이터가 어떻게 저장되는지 실험을 통해 확인한다. 그리고 저장된 데이터를 분석하여 과거에 발생한 행위를 추적하고 행위가 발생한 시점을 추정하는 방법을 제안한다. 실험은 10GB 크기의 가상디스크를 ReFS 3.4로 포맷한 볼륨에서 진행하였다. 그리고 실험 대상 볼륨의 루트 디렉터리 하위에서 텍스트 파일(파일명: TEST.txt)을 생성하고 삭제한 뒤 Logfile을 수집하여 분석하였다.

5.1 파일 생성

ReFS 볼륨에서 파일을 생성한 경우 디렉터리에 File Index와 File Record가 추가된다. 루트 디렉터리 하위에 TEST.txt 파일을 생성하면 가장 먼저 파일에 대한 File Index가 디렉터리에 생성된다. 이 작업이 실패할 경우를 대비하여 〔그림 11〕과 같이 Redo 레코드로 로그를 남긴다. Redo 레코드의 Opcode가 0x1(Redo Insert)이므로 메타데이터 삽입(Insert) 작업에 대한 레코드이다. 이와 함께 메타데이터 삽입 작업 데이터로 트랜잭션에 대한 Table Descriptor 데이터와 디렉터리에 삽입될 Key-Value 데이터가 존재한다. 트랜잭션 내 Table Descriptor 데이터는 Redo 레코드의 데이터가 어떤 메타데이터 파일의 페이지에 적용되는지에 대한 정보를 담고 있다. 〔그림 11〕에서는 Table Descriptor 데이터에서 루트 디렉터리를 의미하는 0x600을 확인할 수 있다. 즉, 〔그림 11〕의 트랜잭션은 루트 디렉터리에 TEST.txt 파일에 대한 File Index에서 사용할 Key-Value를 삽입하는 것이다.

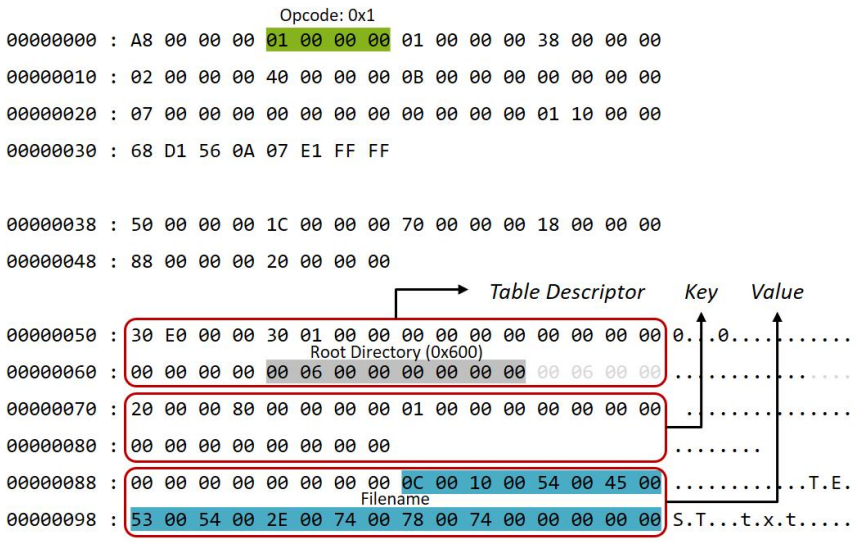

그림 11. File Index 생성에 대한 Redo Record
Fig. 11. Redo Record for File Index creation

File Index가 삽입되면 디렉터리를 업데이트한 후 파일의 메타데이터를 추가하는 작업을 한다. 〔그림 12〕는 File Record를 추가하는 것에 대한 Redo 레코드이다. Opcode가 0x0(Open Table)으로 File Record 내 테이블에 접근하는 작업에 대한 Redo 레코드임을 의미한다. TEST.txt 파일에 대한 속성 정보는 File Record의 테이블에 저장되므로 테이블에 접근하여 업데이트하는 작업이 필요하다. 이때 업데이트하려는 파일의 메타데이터에서 시간 값을 획득할 수 있다. 이 경우에 메타데이터의 파일 생성시간 값을 파일을 생성한 시점으로 추정할 수 있다. 로그 레코드는 USN 레코드와 달리

시간 값을 기록하지 않는다. 그래서 어떤 행위가 발생한 사실 자체는 확인할 수 있지만, 발생 시점을 정확히 특정하기는 어렵다.

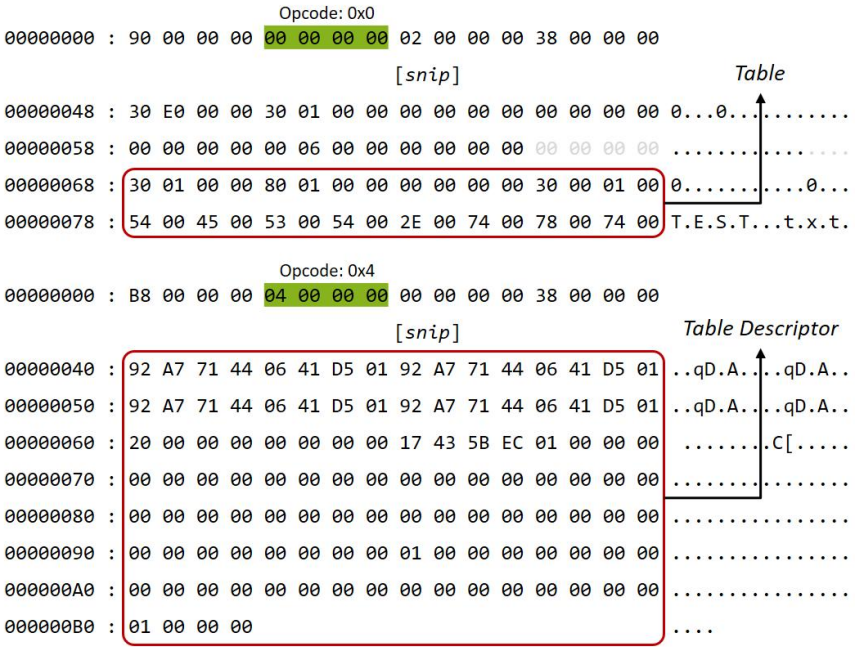

그림 12. File Record 생성과 업데이트에 대한 Redo Record
Fig. 12. Redo Record for creating and updating File Record

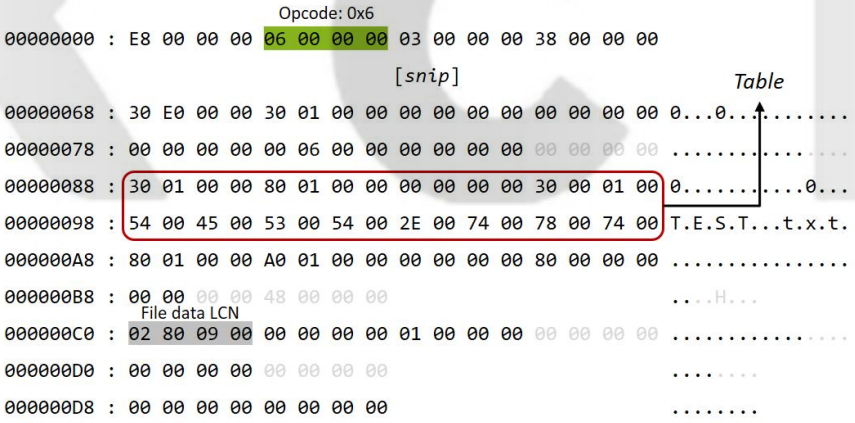

그림 13. 파일 할당에 대한 Redo Record
Fig. 13. Redo Record for File Allocation

표 3. 파일 생성 트랜잭션 로그
Table 3. File creation transaction log

Operation (Opcode)	Description	Remarks
Insert (0x1)	디렉터리 테이블 내 File Index 추가	생성한 파일명 확인 가능
Update Data with Root (0x4)	디렉터리 테이블 업데이트	
Value as Key (0x10)	디렉터리 테이블 내 File Index 키 값 설정	
Open Table (0x0)	File Record 생성	생성한 파일명 확인 가능
Update Data with Root (0x4)	File Record 테이블 업데이트	파일 메타데이터 확인 가능
Insert (0x1)	File Record 테이블 내 속성 추가	
Open Table (0x0)	File Record 테이블 접근	생성한 파일명 확인 가능
Allocate (0x6)	파일 할당(파일 할당 주소 추가)	파일 할당 주소 확인 가능

〔그림 13〕과 같이 파일의 할당 정보가 담긴 메타데이터까지 File Record 내 테이블을 업데이트하는 것으로 파일 생성 작업은 끝이 난다. 파일 생성에 대한 Logfile의 로그 레코드를 분석한 결과, 파일 생성에 따른 트랜잭션 로그는 〔표 3〕과 같이 정리할 수 있다. 로그 레코드를 분석하여 파일 생성 시 나타나는 트랜잭션 로그를 찾으면 특정 파일이 생성된 사실을 알 수 있다.

5.2 파일 삭제

파일을 삭제하는 경우는 볼륨 내 휴지통으로 이동하는 경우와 완전삭제하는 경우로 구분되어 로그가 다르게 생성된다. 휴지통으로 파일을 삭제하는 작업은 $I 파일을 생성하고 TEST.txt 파일을 $R 파일로 변경하는 것으로 진행된다. $I 파일이 생성되는 과정은 일반 파일을 생성하는 과정과 동일하다. 반면에 $R 파일은 원본 파일인 TEST.txt를 $RYQBOT0.txt로 이름을 설정하고 루트 디렉터리에서 휴지통으로 부모 디렉터리를 재설정(파일 이동)하는 것으로 생성된다. 이 작업에 대한 Redo 레코드는 〔그림 14〕와 같이 나타난다.

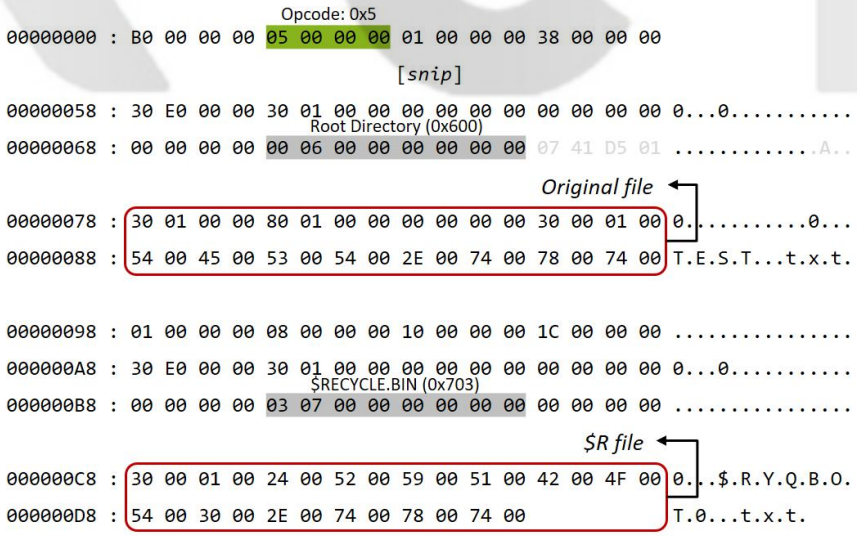

그림 14. 휴지통을 통한 파일 삭제 시 나타나는 Redo Record
Fig. 14. Redo Record appears when deleting files through the Recycle Bin

파일을 완전삭제하는 경우에는 파일의 할당을 해제하고 File Record 테이블과 File Index를 삭제한다. 이 경우에는 로그 레코드 내 Redo 레코드들의 Opcode 패턴은 0x7(Redo Free), 0xF(Redo Delete Table), 0x2(Redo Delete Row)로 나타난다. 〔그림 15〕는 File Record 테이블을 삭제하고 File Record의 Row를 삭제하는 Redo 레코드를 보인 것이다.

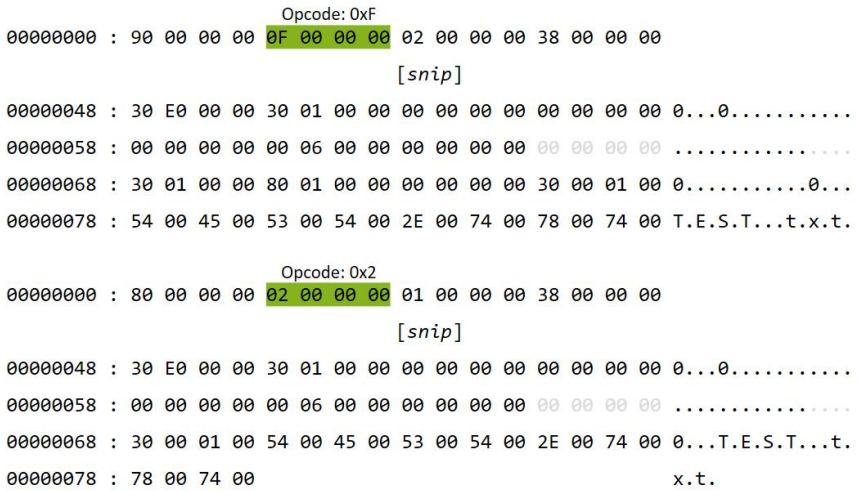

그림 15. 파일 완전 삭제 시 나타나는 Redo Record
Fig. 15. Redo Record when deleting files

표 4. 파일 삭제 트랜잭션 로그
Table 4. File delete transaction log

Operation (Opcode)	Description	Remarks
Free (0x7)	파일 할당 해제	
Update Data with Root (0x4)	디렉터리 테이블 업데이트	
Delete Table (0xF)	File Record 테이블 삭제	삭제한 파일명 확인 가능
Delete Row (0x2)	디렉터리 테이블 내 File Index 키값 삭제	
Delete Table (0xF)	File Record 삭제	
Delete Row (0x2)	디렉터리 테이블 내 File Record 삭제	

파일 완전삭제에 대한 Logfile의 로그 레코드를 분석한 결과, 파일 삭제에 따른 트랜잭션 로그는 〔표 4〕와 같이 정리할 수 있다. 로그 레코드를 분석하여 파일 완전삭제 시 나타나는 트랜잭션 로그를 찾으면 특정 파일이 삭제된 사실을 알 수 있다.

5.3 Logfile 시간 값 분석

Logfile의 로그 레코드에서 트랜잭션이 발생한 시점은 볼륨에서 일어난 행위를 분석하는데 중요한 요소이다. 그러나 로그 레코드는 트랜잭션이 발생한 시간을 별도로 저장하지 않는다. 하지만 파일이나 디렉터리의 시간 값을 업데이트하는 트랜잭션의 경우에는 그 트랜잭션이 발생한 시점을 추정할 수 있다. 〔표 5〕는 로그 레코드에서 트랜잭션의 발생 시점을 추정할 수 있는 경우와 트랜잭션 시간에 대해 정리한 것이다.

파일이나 디렉터리가 생성되는 경우에는 업데이트되는 File Record나 Directory Record 내 생성시간을 트랜잭션 발생 시간으로 취급한다. 파일 내용 수정에 대한 트랜잭션의 발생 시간도 File Record 내 수정시간으로 추정할 수 있다. 파일·디렉터리 생성, 파일 내용 수정의 경우 직접적으로 해당 트랜잭션에 존재하는 시간 값을 활용하여 시점을 특정할 수 있다. 하지만 파일이나 디렉터리를 삭제·이름 변경하는 경우는 부모 디렉터리의 시간 값을 업데이트하는 트랜잭션을 활용하여 간접적으로 트랜잭션 발생 시간을 추측할 수 있다.

표 5. 트랜잭션 시간 값이 기록되는 경우
Table 5. A case which has been logged transaction time

Case	Transaction Time	Description
파일 생성	파일 생성시간	File Record 내 타임스탬프 중 생성시간
파일 삭제	부모 디렉터리 메타데이터 수정시간	삭제된 파일의 부모 디렉터리의 메타데이터 수정시간
파일 내용 수정	파일 수정시간	File Record 내 타임스탬프 중 수정시간
파일 이름 변경	부모 디렉터리 메타데이터 수정시간	이름이 변경된 파일의 부모 디렉터리의 메타데이터 수정시간
디렉터리 생성	디렉터리 생성시간	Directory Record 내 타임스탬프 중 생성시간
디렉터리 삭제	부모 디렉터리 메타데이터 수정시간	삭제된 디렉터리의 부모 디렉터리의 메타데이터 수정시간
디렉터리 이름 변경	부모 디렉터리 메타데이터 수정시간	이름이 변경된 디렉터리의 부모 디렉터리의 메타데이터 수정시간

5.4 포렌식 조사 활용 방안

ReFS 저널링 파일에서 식별한 파일 작업은 파일의 삭제 시점 특정, 사용자의 마지막 행위 추적과 같은 포렌식 조사에 활용할 수 있다. Logfile은 파일시스템이 파일 작업을 처리하면서 발생한 일련의 트랜잭션을 로그 레코드로 기록한다. 이러한 Logfile에서 포렌식 조사에 필요한 정보를 얻기 위해서는 일련의 트랜잭션 로그를 ReFS 파일 작업 단위로 구성해야 한다. 예를 들어 조사를 위해 수집한 Logfile에서 〔표 4〕와 같이 나타나는 트랜잭션 로그를 식별하면 파일 삭제 작업이 있었음을 알 수 있다. 파일 삭제 작업을 분석하면 삭제된 파일의 정보와 파일이 삭제된 시점을 특정할 수 있다. ReFS에서는 파일을 완전삭제하는 작업에서 File Record 테이블을 삭제하는 트랜잭션을 처리한다. 이 트랜잭션의 데이터는 삭제한 파일의 File Record 정보로 구성되며 삭제한 파일의 이름을 확인할 수 있다. 한편, 확인한 파일을 삭제한 시점을 특정하기 위해서는 파일 삭제가 발생한 디렉터리의 테이블을 업데이트하는 트랜잭션 데이터도 함께 살펴봐야 한다. 이처럼 Logfile에서 하나의 파일 작업을 분석하기 위해서는 트랜잭션 로그 간에 연결이 필요하다. 따라서 Logfile에서 파일 작업에 따른 트랜잭션 로그를 식별하고 파일 작업 단위로 구성한다면 포렌식 조사에 유용한 정보를 얻을 수 있다. 추가적으로 Change Journal의 USN 레코드 정보를 함께 활용하면 ReFS에서 파일 작업이 발생한 시점, 작업 내용과 같은 정보를 보강할 수 있다.

Ⅵ. 결 론

본 논문에서는 기존의 ReFS 연구에서 부족했던 데이터 구조에 대해 추가로 분석하였다. 그리고 ReFS 저널링 파일인 Change Journal과 Logfile에 대한 상세 구조 분석을 진행하였으며, 포렌식 조사에 활용할 수 있는 방안을 제안하였다.

ReFS는 NTFS와 구조적으로 차이가 있다. 페이지 내 테이블 구조로 데이터를 관리하며, 파일이나 디렉터리에 대한 정보를 NTFS의 $MFT와 같이 중앙집중화된 메타데이터 파일로 저장하지 않는다.

Change Journal은 USN 레코드를 이용하여 볼륨 내 파일이나 디렉터리의 변경 사항을 기록한다. ReFS에서는 Object ID 표현에 128bit를 사용하기 때문에 File Reference Number에 128bit를 사용하는 USN_RECORD_V3를 사용한다. Change Journal의 USN 레코드에는 시간 값이 들어가기 때문에 USN 레코드 내 이벤트의 발생 시점을 특정할 수 있다. 따라서 Change Journal을 분석하면 ReFS 볼륨에서 특정 시점에 발생한 파일이나 디렉터리에 대한 행위를 추적할 수 있다.

Logfile은 NTFS의 $LogFile과 같이 트랜잭션을 로깅하며 재실행해야 하는 작업에 대해서만 로깅을 한다. 또한, Logfile에서는 다른 메타데이터 파일에서 데이터 저장을 위해 페이지 구조를 사용하는 것과 다르게 로그 엔트리 구조로 트랜잭션 데이터를 저장한다. 로그 엔트리 내부의 로그 레코드에서 재실행 작업들에 대한 데이터가 저장되어 있다. 로그 레코드를 분석하면 ReFS 볼륨에서 처리된 작업과 데이터를 알 수 있다. 이 정보를 이용하면 과거에 ReFS 볼륨에서 어떤 작업이 발생하였는지 조사할 수 있다. Logfile은 Change Journal과 달리 작업의 발생 시점을 특정할 수 있는 시간 값을 기록하지 않는다. 다만, 몇 가지 경우에는 저장된 트랜잭션 데이터를 통해 시점을 추정할 수 있음을 확인하였다.

ReFS의 Change Journal과 Logfile에 대한 분석 결과는 ReFS 볼륨을 대상으로 한 포렌식 조사에 유용할 것으로 기대된다. 특히 Logfile을 분석한 결과는 포렌식 조사의 목적뿐만 아니라 ReFS 동작을 이해할 수 있는 정보를 습득하는데 사용될 수 있다.

참 고 문 헌 (References)

〔1〕 Microsoft, "Building the next generation file system for Windows: ReFS", visited 2019-08-30, Available: https://blogs.msdn.microsoft.com/b8/2012/01/16/building-the-next-generation-file-system-for-windows-refs/, 2012

〔2〕 Microsoft, "Resilient File System (ReFS) overview", visited 2019-08-30, Available: https://docs.microsoft.com/ko-kr/windows-server/storage/refs/refs-overview, 2019

〔3〕 Nordvik R, Toolan F, Axelsson S, "Using the object ID index as an investigative approach for NTFS file systems", Digital Investigation. 28 Supplement, pp. S30-S39. 2019

〔4〕 Head A., "Forensic Investigation of Microsoft's Resilient File System (ReFS)", visited 2019-08-30, Available: http://resilientfilesystem.co.uk/, 2015

〔5〕 Santwana G., Radha D., Pallavi D, "Windows ReFS File System: A Study", International Journal of Advanced Research in Computer and Communication Engineering. 4, pp. 596-599, 2015

〔6〕 Kim S. H., "ReFS 데이터 저장 원리 분석 및 메타데이터를 이용한 삭제된 데이터 복구 기술 연구", Available: http://www.riss.kr/link?id=T15064519, 2018

〔7〕 Georges H., "Resilient Filesystem", NTNU, Available: http://hdl.handle.net/11250/2502565, 2018

〔8〕 Nordvik R., Georges H., Toolan F., Axelsson S., "Reverse engineering of ReFS", Digital Investigation. 30, pp. 127-147, 2019

〔9〕 Metz J. "Resilient File System (ReFS)", visited 2019-08-30, Available: https://github.com/libyal/libfsrefs/blob/master/documentation/Resilient%20File%20System%20(ReFS).pdf, 2013

〔10〕 Lees C., "Determining removal of forensic artefacts using the USN change journal", Digital Investigation. 10, pp. 300-310, 2013

〔11〕 Oh J., "NTFS Log Tracker", visited 2019-08-30, Available: http://forensicinsight.org/wp-content/uploads/2013/06/F-INSIGHT-NTFS-Log-TrackerEnglish.pdf, 2013

〔12〕 Carrier B., File System Forensic Analysis, Addison-Wesley, pp. 392-395, 2005

〔13〕 Microsoft, USN_RECORD_V2 structure, visited 2019-08-30, https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-usn_record_v2, 2018

〔14〕 Microsoft, USN_RECORD_V3 structure, visited 2019-08-30, Available: https://docs.microsoft.com/en-us/windows/win32/api/winioctl/ns-winioctl-usn_record_v3, 2018

저 자 소 개

이 선 호 (Seonho Lee)

2018년 2월 : 조선대학교 컴퓨터공학과 졸업
2018년 3월~현재 : 고려대학교 정보보호학과 석사과정
관심분야 : 디지털 포렌식

최 호 용 (Hoyong Choi)

2014년 2월 : 고려대학교 컴퓨터 통신공학과 졸업
2014년 3월~현재 : 고려대학교 정보보호학과 석박통합과정
관심분야 : 파일시스템 포렌식, 데이터베이스 포렌식, 문서 포렌식

박 정 흠 (Jungheum Park)

2014년 2월 : 고려대학교 정보보호대학원 공학박사
2014년 3월~2014년 12월 : 고려대학교 정보보호연구원 연구교수
2015년 1월~2019년 2월 : 미국 국립표준기술연구원(NIST), 방문연구원
2019년 3월~현재 : 고려대학교 정보보호연구원 디지털포렌식연구센터 연구교수
관심분야 : 사이버 보안, 개인정보보호, 디지털 포렌식

이 상 진 (Sangjin Lee)

1989년 10월~1999년 2월: ETRI 선임 연구원
1999년 3월~2001년 8월: 고려대학교 자연과학대학 조교수
2001년 9월~현재: 고려대학교 정보보호대학원 교수
2008년 3월~현재: 고려대학교 디지털포렌식연구센터 센터장
관심분야 : 디지털 포렌식, 심층암호, 해쉬함수