NTFS - Part 4 (보고서)

날짜	@2024년 2월 13일
사람	ohnahee

Chapter 11. NTFS 개념
- 11.1. 소개
- 11.2. 모든 것이 파일
- 11.3. MFT 개념
  - MFT 엔트리 내용
  - MFT 엔트리 주소
  - 파일시스템 메타데이터 파일
- 11.4. MFT 엔트리 속성 개념
  - 속성헤더
  - 속성내용
  - 표준속성유형들
- 11.5. 다른 속성 개념
  - 기준 MFT 엔트리
  - Sparse 속성
  - 압축된 속성
  - 암호화 된 속성
- 11.6. 인덱스
- 11.7. 분석도구

Chapter 12. NTFS 분석
- 12.1. 파일 시스템 범주
  - $MFT 파일 개요
  - $MFTMirr 파일 개요
  - $Boot 파일 개요
  - $Volume 파일 개요
  - **$AttrDef 파일 개요**
  - 분석 기술
  - 분석 고려 사항
- 12.2. 내용범주
  - 클러스터
  - **$BITMAP 파일 개요**
  - $BadClus 파일 개요
  - 할당 알고리즘
  - 파일시스템 레이아웃
  - 분석 기술
  - **분석 고려사항**
- 12.3. 메타데이터 범주
  - $STANDARD_INFORMATION 속성
  - **$FILE_NAME 속성**
  - **$DATA 속성**
  - **$ATTRIBUTE_LIST 속성**
  - **$Secure 파일**
  - **할당 알고리즘**
- 12.4. 파일 이름 범주
  - 디렉토리 인덱스
  - 루트 디렉토리
  - 파일과 디렉토리 연결
  - 오브젝트 식별자

할당알고리즘
12.5. 응용프로그램 범주
디스크 할당
로깅 - 파일시스템 저널링
변경저널
12.6. 큰 그림
파일 할당 예제
파일 삭제 예제
12.7. 다른 주제
파일 복구
일관성 검사
Chapter 13. NTFS 데이터 구조
13.1. 기본 개념
Fixup 값
MFT 엔트리 (파일 레코드)
속성 헤더
13.2. 표준 파일 속성
$STANDARD_INFORMATION 속성
$FILE_NAME 속성
$DATA 속성
$ATTRIBUTE_LIST 속성
$OBJECT_ID 속성
$REPARSE_POINT 속성
13.3. 인덱스 속성과 데이터 구조
$INDEX_ROOT 속성
$INDEX_ALLOCATION
$BITMAP 속성
인덱스 노드 헤더 데이터 구조체
일반 인덱스 엔트리 데이터 구조체
디렉토리 인덱스 엔트리 데이터 구조체
13.4 파일시스템 메타데이터 파일
$MFT 파일
$Boot 파일
$AttrDef 파일
$BITMAP 파일
$VOLUME 파일
$ObjId 파일
$Quota 파일
$LogFile 파일
$UsrJrnl 파일

Chapter 11. NTFS 개념

- MS에 의해 설계되고 사용되는 기본 파일 시스템

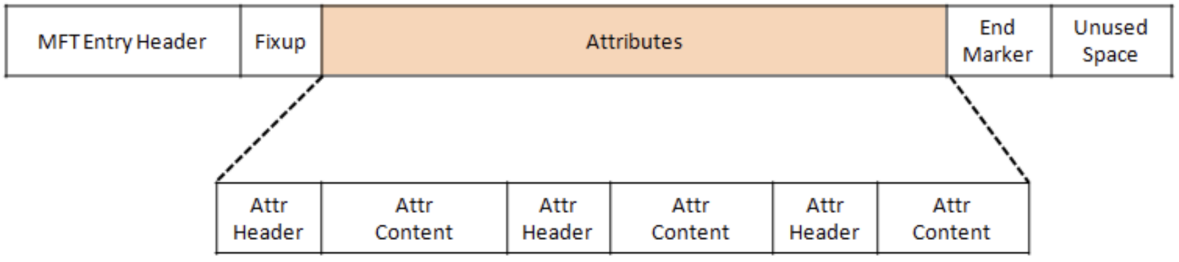
11.1. 소개

- 신뢰성, 보안, 대용량 장치를 지원하기 위해 설계
- 특정 내용으로 데이터 구조체를 감싸는 포괄적인 데이터 구조체를 사용해서 확장성 제공
- NFTS 파일 시스템 데이터의 모든 바이트가 한 파일에 할당 됨
- 굉장히 복잡한 시스템
    - MS에서 명세를 공개하지 않았기에 실제 구조와 다를 수 있음

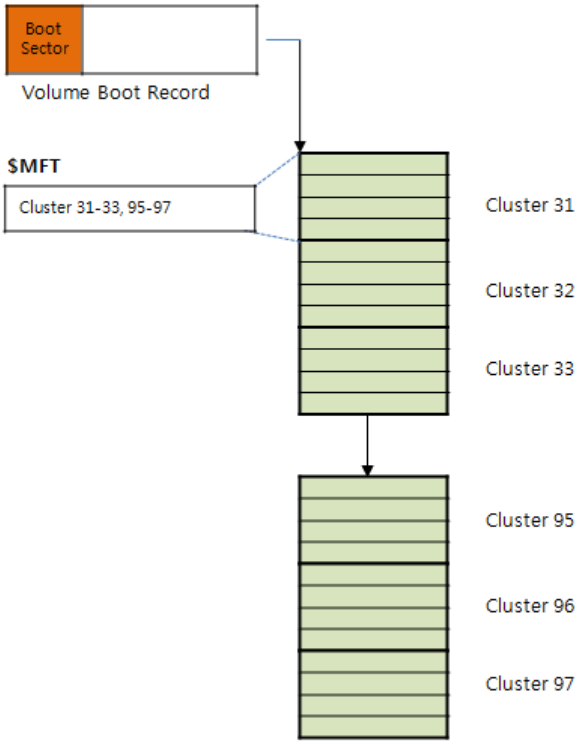
11.2. 모든 것이 파일

- 중요한 데이터가 파일로 할당됨
- NTFS 파일 시스템은 다른 파일시스템과 같이 특정 레이아웃을 갖지 않음
    - 전체 파일 시스템은 데이터 영역으로 간주되며 어떤 섹터도 파일에 할당될 수 있음
    - 볼륨의 첫 섹터에 부트섹터를 포함하고 그곳에 부트코드가 존재

11.3. MFT 개념



- 모든 파일들과 디렉토리에 대한 정보를 가지고 있음
- 모든 파일과 디렉토리는 테이블에 한 엔트리를 반드시 가지고 그 구조는 간단하다
- 엔트리 크기는 1KB, 첫 42바이트는 미리 정의된 목적을 가짐
    - 그 밖의 나머지 바이트들은 특정한 목적을 갖는 데이터 구조체로 이루어진 속성들을 저장함
- MS 에서는 각 엔트리를 파일레코드로 부름
    - 이 책에서는 MFT 엔트리
- 첫 엔트리는 $MFT 이름이며 MFT의 디스크 위치를 설명한다
    - MFT 시작 위치는 파일 시스템의 첫번째 섹터에 위치한 부트 섹터에서 주어진다

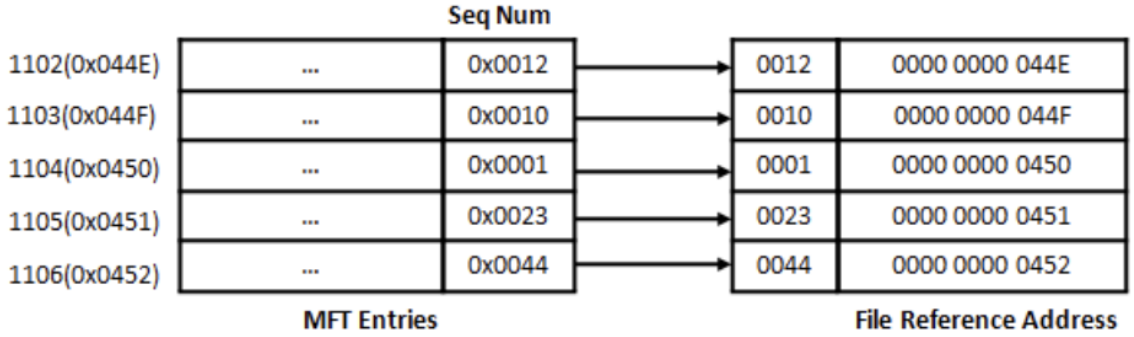


- 부트섹터를 이용해서 첫 번째 MFT 엔트리를 찾을 수 있음
- 첫 엔트리를 통해 MFT가 클러스터 32~34, 56~58에 단편화 되어 있음을 알 수 있음
- NTFS는 FAT처럼 연속적인 섹터 그룹인 클러스터를 이용한다
- NTFS 구현은 NTFS를 작게 시작하고 엔트리가 많이 필요해지면 크기를 확장하는 식으로 사용 됨
  - 한 운영 체제에서 파일 시스템 생성시 고정된 수의 엔트리를 생성하고 필요시 동적으로 크게 만드는 방식

MFT 엔트리 내용

- MFT의 각 엔트리 크기는 부트섹터에서 정의
- MS의 XP 버전까지는 1024바이트 크기 사용
- 데이터 구조체의 첫 42바이트는 12개의 필드를 포함하고 나머지 982 바이트는 비구조적으로 속성값을 가지고 있다
- 각 엔트리에 첫 필드는 시그니처이고 표준 엔트리는 아스키 문자열 FILE을 가지고 있다
  - 만약 그 엔트리에서 오류를 발견하면 BAAD 라는 문자열을 가짐
- 파일이 한 개의 엔트리 내에서 그 파일의 속성을 나타내기 적합하지 않으면 여러 엔트리를 사용할 수 있다
  - 이 같은 경우가 발생하면 첫 엔트리는 기준 파일 레코드나 기준 MFT 엔트리로 불리며 각 다음 엔트리들은 고정된 필드들 중 하나의 기준 엔트리 주소를 포함함

MFT 엔트리 주소



- 각 MFT 엔트리는 48 비트 값을 이용하여 순차적으로 주소 지정
- 첫 엔트리는 주소 0
- 파일 번호 : 최대 MFT 주소는 MFT가 증가 할 때 마다 변경되며 $MFT 크기를 각 엔트리 크기로 나누어서 결정
- 모든 MFT 엔트리는 할당 될 대 마다 증가하는 16비트 순서번호를 가짐
  - 순서 번호와 파일 번호를 결합해 파일 참조 주소를 만든다
  - 순서번호(16) + 파일번호(48) = 파일 참조주소 (64비트)

- NTFS는 순서 번호가 파일 시스템이 손상된 상태인지 아닌지를 판단하기 쉽게 해주기 때문에 MFT 엔트리를 참조하는 파일 참조 주소를 사용함
- 순서 번호는 데이터 구조체가 MFT 엔트리 주소를 포함하는지 판단한다
  - 순서번호는 이전 파일이 그 엔트리 주소를 사용했거나 새로운 파일의 부분임을 판단 가능
  - 지워진 파일을 복구할 대도 사용 가능
    - 참조번호를 갖는 구조체가 있다면 구조체 엔트리를 사용한 이후로 MFT 엔트리가 재할당되었는지 판단할 수 있다

파일시스템 메타데이터 파일

NTFS의 메타 데이터 파일 종류

Entry Num	File Name	설명
0	$MFT	MFT을 담고 있는 파일
1	$MFTMirr	MFT 파일의 백업본
2	$LogFile	트랜잭션 저널 기록 저장
3	$Volume	볼륨의 레이블, 버전, 볼륨 정보
4	$AttrDef	인자 값, 이름, 크기 등 속성 값
5	.	볼륨의 루트 디렉터리 저장
6	$Bitmap	볼륨의 클러스터 할당 관리 정보
7	$Boot	부트 레코드 영역의 정보
8	$BadClus	배드 클러스터에 대한 정보
9	$Secure	파일들의 보안과 접근권한 정보
10	$Upcase	모든 유니코드 문자의 대문자 정보
11	$Extend	추가적인 확장을 담고 있는 디렉터리, Windows는 일반적으로 어떠한 정보도 담지 않는다.
12~15	사용안함	사용 중 이라고 설정되어 있으니 비어있음
16~23	사용안함	미래를 위해 비어 있다.
상관없음	$Objid	파일 고유의 ID정보를 담고 있다.
상관없음	$Quota	사용량 정보를 담고 있다.(Win 2000 이상부터 존재)
상관없음	$Reparse	Reparse Point에 대한 정보를 담고 있다.(Win 2000 이상부터 존재)
상관없음	$UsnJrnl	파일이나 디렉터리에 변경이 있을 경우 그 기록을 담아 놓는 파일이다.
24~	일반 파일	일반적인 파일이나 디렉터리들이 여기서부터 저장된다.

- 파일 시스템 관리 데이터를 저장하는 별도의 파일이 있어야 함
  - MS는 이러한 파일들을 메타 데이터 파일이라고 칭함
  - 이 책에서는 파일 시스템 메타데이터 파일이라고 명함
- MS는 첫 16개 MFT 엔트리들을 파일 시스템 메타데이터 파일로 예약
- 이 파일 시스템 메타데이터파일은 사용자들에게 숨겨지지만 루트 디렉토리에 존재하게 됨
- 각 파일 시스템 메타 데이터 파일명은 $로 시작하고 첫 문자는 대문자로 씀

11.4. MFT 엔트리 속성 개념

- MFT 엔트리는 내부 구조체를 사용하고 대부분 특정 유형 데이터를 저장하는 속성을 저장하는 데 사용된다
- 많은 타입이 있고 각각은 자신의 내부 구조체를 갖는다

    - 다른 파일시스템과 확연히 다른 점
- 대부분의 파일 시스템은 파일 내용을 읽고 쓰기 위해 존재하지만 NTFS 파일시스템은 속성들을 읽고 쓰기 위해 존재한다
- 엔트리 속 각 속성은 헤더와 내용으로 이루어져있다
- MFT 엔트리 헤더는 모든 속성을 포괄하며 기준이 된다

속성헤더

- 속성의 타입, 크기, 이름을 구분한다
- 값들의 압축 여부, 암호화 여부를 식별하는 플래그 값이 있다
- 속성 타입은 데이터 타입에 기초하는 숫자 식별자
- 한 MFT 엔트리에는 동일 타입의 여러 속성들이 있을 수 있다
- 속성은 MFT 엔트리 내에서 고유한 식별자를 가지고 어떤 속성을 UTF-16인 이름에 할당될 수 있다

속성내용

- 어떤 형식과 크기가 될 수 있다
    - EX) 어떤 속성은 파일에 내용을 저장하기 위해 사용되고 그 크기가 수 MB에서 GB가 될 수 있다 하지만 1024바이트인 MFT 엔트리에 이러한 데이터를 저장하는 것은 실용성에서 많이 떨어짐 → 이러한 문제를 해결하기 위해 NTFS는 속성 내용이 저장되는 장소가 2개
- 거주 속성
    - 헤더가 있는 MFT 엔트리 내용 저장

      → 오직 작은 속성에서 사용
- 비거주 속성
    - 파일 시스템 외부 클러스터에 내용을 저장

표준속성유형들

속성 식별값	속성 이름	설명
16 (0x10)	$STANDARD_INFORMATION	파일의 최근 생성,접근,수정 시간, 소유자 등의 일반적인 정보
32 (0x20)	$ATTRIBUTE_LIST	속성들에 대한 리스트
48 (0x30)	$FILE_NAME	파일 이름(유니코드), 최근 생성,접근.수정 시간
64 (0x40)	$VOLUME_VERSION	볼륨 정보 (윈도우 NT 1.2 버전에만 존재)
64 (0x40)	$OBJECT_ID	파일 및 디렉터리의 16바이트 고유값 (윈도우 2000+)
80 (0x50)	$SECURITY_DESCRIPTOR	파일의 접근 제어와 보안 속성
96 (0x60)	$VOLUME_NAME	볼륨 이름
112 (0x70)	$VOLUME_INFORMATION	파일시스템 버전과 플래그 정보
128 (0x80)	$DATA	파일 내용
144 (0x90)	$INDEX_ROOT	인덱스 트리의 루트 노드 정보
160 (0xA0)	$INDEX_ALLOCATION	인덱스 트리의 루트와 연결된 하위 노드 정보
176 (0xB0)	$BITMAP	$MFT의 비트맵 정보
192 (0xC0)	$SYMBOLIC_LINK	심볼릭 링크 정보 (윈도우 2000+)
192 (0xC0)	$REPARSE_POINT	심볼릭 링크에서 사용하는 Reparse point 정보 (윈도우 2000+)
208 (0xD0)	$EA_INFORMATION	OS/2 응용프로그램과 호환성을 위해 존재 (HPFS)
224 (0xE0)	$EA	OS/2 응용프로그램과 호환성을 위해 존재 (HPFS)
256 (0xF0)	$LOGGED_UTILITY_STREAM	암호화된 속성의 정보와 키 값 (윈도우 2000+)

- 각 속성 타입은 번호로 정의되고 MS는 이러한 번호를 사용해서 엔트리 속성을 분류함
- 숫자 이외의 각 속성타입은 이름을 가지며 모두 대문자로 구성되며 앞에 $가 있다
- 비기준 MFT 엔트리를 제외하고 거의 모든 MFT 엔트리는 $FILE_NAME과 $STANDARD_INFOMATION 타입 속성을 가짐
  - $FILE_NAME 속성은 파일명, 크기, 임시정보가 들어있다
  - $STANDARD_INFOMATION 속성은 소유권, 보안, 임시정보가 들어있다
- 모든 파일은 파일 내용을 저장할 $DATA 속성을 갖는다
- 모든 디렉토리는 그 디렉토리내 위치한 파일들과 하위 디렉토리 정보들을 포함하는 $INDEX_ROOT 속성을 갖는다
  - 디렉토리가 크면 정보를 저장하기 위해 $INDEX_ALLOCATION과 $BITMAP 속성들을 사용한다

11.5. 다른 속성 개념

💡 1. 특별한 파일이 아주 많은 속성을 가질 때의 경우

2. 그 속성들의 내용을 압축하고 암호화 하는 방법

기준 MFT 엔트리

- 한 파일의 속성은 65,536개 (16비트 식별자) 가 있을 수 있고 속성 헤더를 모두 저장하기 위해 한 개 이상의 MFT 엔트리가 필요할 수 있다
- 추가적인 MFT 엔트리들이 한 파일에 할당될 때 그 원본은 기준 MFT 엔트리가 된다
  - 비기준 엔트리는 그것들의 필드에 기준 주소를 가짐
- 기준 MFT 엔트리에는 $ATTRIBUTE_LIST타입 속성이 있고 그 속성은 파일 속성과 MFT 주소가 있는 목록을 포함함
- 비지군 MFT 엔트리에서는 $FILE_NAME과 $STANDARD_INFOMATION 속성이 없음

Sparse 속성

- NTFS 는 비거주 $DATA 속성 값을 Sparse로 저정하여 한 파일에 필요한 공간을 줄일 수 있다
- 이 속성은 모두 0인 클러스터들이 디스크에 써지지 않도록 하는 것
- 대신 한 특별한 run이 0 클러스터를 위해 생성된다
- 일반 run은 시작 위치와 크기를 표현하지만 Sparse run은 단지 크기만을 표현한다

  그곳에는 Sparse 인지 아닌지를 나타내는 플래그도 존재

압축된 속성

- NTFS는 속성들을 압축된 형태로 사용 가능하다
  - 이는 파일 시스템 수준 압축으로 응용프로그램 수준이 아님
- MS는 $DATA 속성이 비거주일때만 압축할 수 있다고 전함
- NTFS는 필요한 저장공간을 줄이기 위해 Sparse runs와 압축된 데이터 2개를 사용
- 속성 헤더 프래그는 압축 여부를 구분하고 $FILE_NAME 와 $STANDARD_INFORMATION 플래그 또한 압축된 속성을 포함한 파일인지 보여줌
- 속성 내용은 압축되기 전 데이터를 압축 유닛 이라는 같은 크기의 데이터 묶음으로 나눈다
  - 압축 유닛 크기는 속성헤더에서 주어짐
  - 각 압축유닛이 발생할 수 있는 3가지 상황 존재

    1. Sparse 데이터 Run이 압축 유닛 크기로 만들어지는 경우, 모든 클러스터들은 0을 포함하고 디스크 공간에 할당하지 않는다

    2. 압축될 때 그 결과 데이터는 저장을 위해 같은 수의 클러스터가 필요하다 이 경우 압축유닛은 압축되지 않고 하나의 run이 원본데이터로 만들어진다

    3. 압축 시 결과데이터는 적은 수의 클러스터를 사용한다. 이 경우 데이터를 먼저 run으로 압축하고 디스크에 저장한다 Sparse run은 전체 run길이와 압축 유닛에 클러스터 수를 같도록 만들기 위해 압축된 run 뒤에 따른다

암호화 된 속성

- NTFS 속성 내용을 암호화 할 수 있는 기능을 제공
- 암호화 방법과 디스크에 존재하는 형식에 대한 개요
- 속성이 암호화 될 때 속성 내용만 암호화 되고 속성 헤더는 암호화 되지 않음
- 파일을 위해 **$LOGGED_UTILITY_STREAM**이 생성되고 그것은 데이터를 복호화하는데 필요한 키를 가짐
- 윈도우에서 사용자는 파일이나 디렉토리를 암호화할 수 있는 선택을 가짐
    - 이 때 **$STANDARD_INFORMATION 속성**에는 특별한 플래그를 가진다 또한 각 속성은 헤더에 특별한 플래그 설정이 있음

암호화 기초

- 암호화 : 평문 데이터를 암호 데이터로 변경하기 위해 암호 알고리즘과 키를 사용하는 과정
- 복호화 : 위와 반대로 복호 알고리즘과 키를 이용해 암호 데이터를 평문 데이터로 변경하는 과정
- 암호 알고리즘 두가지 종류
    - 대칭 : 암호화키와 복호화 키가 같은 것
    - 비대칭 : 암호화키와 복호화 키가 다른것
        - 공개키 : 누구에게나 공개되어 있는 키
        - 개인키 : 본인만 가지고 있는 키

NTFS 구현

- $DATA 속성을 암호화 할 대 DESX 라는 대칭 알고리즘 이용
- 한 개의 난수키는 암호화 된 데이터가 있는 각 MFT 엔트리를 위해 생성되고 FEK(File Encryption Key) 로 부른다
- MFT 엔트리에 다수의 $DATA 가 있으면 그것들은 모두 같은 FEK 로 암호화 됨
- $LOGGED_UTILTY_STREAM 속성은 DDF와 DRF의 목록을 포함
- DDF(Data Decryption Fields)는 그 파일에 접근하려는 모든 사용자를 위해 생성되고, 사용자의 SID(Security_ID), 암호화정보, 사용자의 공개키로 암호화된 FEK(즉, 데이터를 얻기 위해서는 사용자의 개인키와 이것을 이용)를 포함
- DRF(Data Recovery Fields)는 데이터 복구를 위해 필요 이것은 데이터 복구 공개키로 암호화된 FEK를 포함 이 공개키는 관리자나 인증된 사용자가 데이터 접근을 하기 위해 필요하다

11.6. 인덱스

- NTFS는 많은 상황에서 인덱스 데이터 구조체를 사용함
- NTFS 인덱스는 정렬된 순서로 저장된 속성들의 모임
    - 특히 디렉토리에서 흔히 사용되는데 $FILE_NAME 속성을 포함하기 때문

- NTFS 3.0 이상부터 $FILE_NAME 외에도 인덱스가 사용됨

11.7. 분석도구

- 1장에서 언급된 모든 도구들은 NTFS 이미지들을 지원
  - 하지만 자신의 윈도우 시스템을 보기 위해 마이크로소프트에서 제공하는 nfi.exe를 사용할 수 있다
  - 이 도구는 동적 시스템의 MFT내용, 속성, 이름, 클러스터를 보여줌
  - 그러나 정적 분석이 어울리는 포렌식에서는 유용한 도구는 아니다
- MFT 엔트리 각 속성은 고유의 식별자를 할당하고, 1개의 타입 값을 가짐
  - 이러한 2개의 값은 어떤 속성이던지 나타냄
- TSK istat을 이용해 속성들을 알 수 있다

Chapter 12. NTFS 분석

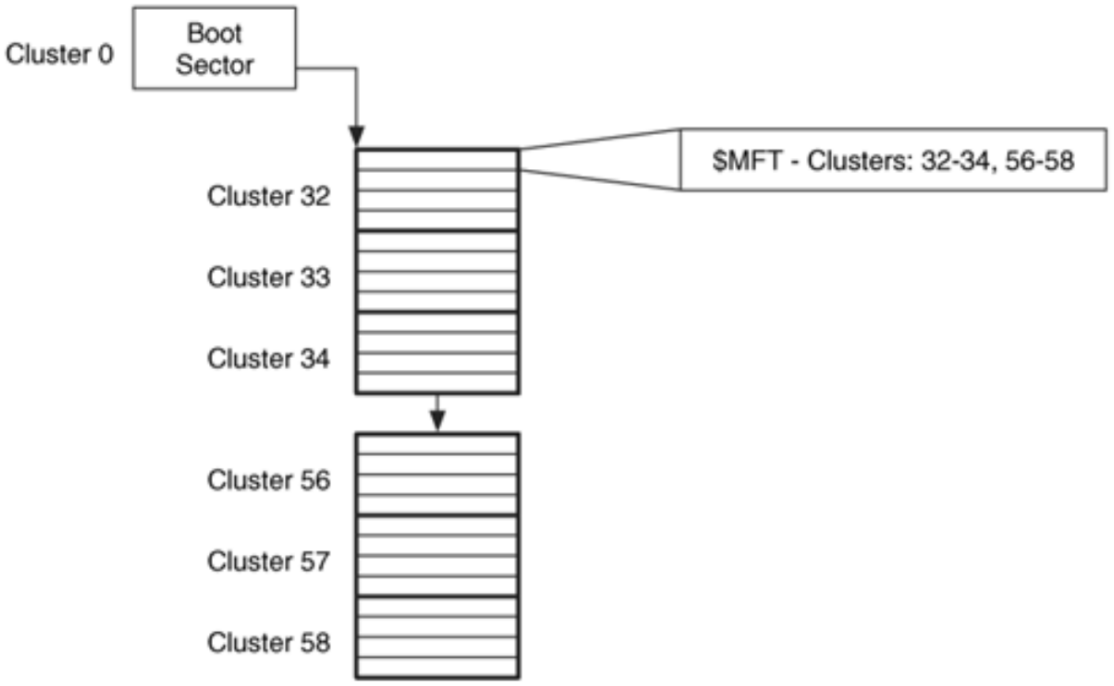
> 💡 모든것이 '한 개의 파일' 이기 때문에 까다롭다

12.1. 파일 시스템 범주

- 파일 시스템 : 파일 시스템 전반적인 설명 데이터 포함
- NTFS에서는 이러한 데이터를 페일 시스템 메타 데이터 파일에서 저장하고 루트 디렉토리에 이 파일이름이 존재한다
  - 이러한 데이터는 파일 시스템 어디든 존재가능
  - 부트코드 예외
  - 특징은 일반 파일들과 비슷하게 날짜, 시간 스탬프를 가지고 있음
    - 날짜와 시간 스탬프들은 파일 시스템이 생성될 때 시간을 설정하기 때문에 분석에 유용할 수 있다

$MFT 파일 개요

- 가장 중요한 파일시스템 메타데이터 **파일 중 하나**
- 모든 파일과 디렉토리 엔트리를 갖는 MFT(Master File Table)를 포함
- 일반 파일을 찾기 위해선 $MFT 파일 필요



- MFT의 첫 엔트리는 $MFT를 위한 것
  - 그것의 $DATA 속성은 MFT에 의해 사용되는 클러스터들을 포함
- $MFT에는 MFT엔트리들의 할당 상태를 관리하는 $BITMAP 속성 존재
- TSK istat 도구를 이용해서 파일 시스템 메타데이터 파일의 세부 내용을 볼 수 있음

$MFTMirr 파일 개요

- $MFT 파일은 모든 파일, 디렉토리를 찾는데 중요하다
- 부트섹터나 $MFT 엔트리들이 손상되면 파일을 찾는데 문제가 생긴다
  - 중요한 MFT 엔트리들은 백업본 존재
  - MFT 엔트리 1 → $MFTMirr을 위한 것

$Boot 파일 개요

- MFT 엔트리에서 7번째 위치
- 파일 시스템의 부트 섹터가 들어가있음
- 고정된 위치를 갖는 메타데이터 파일
  - $DATA 속성은 부팅시 필요하기 때문에 항상 파일시스템의 첫 섹터의 위치
- NTFS 부트섹터는 FAT의 많은 필드들과 유사
- 부트섹터는 각 클러스터 크기, 파일 시스템의 섹터 수, MFT 시작 클러스터 주소, 각 MFT 엔트리 크기의 기본 크기 정보 제공

$Volume 파일 개요

- MFT 엔트리에서 3번째에 위치
- 볼륨 레이블과 다른 버전 정보 포함
- 다른 파일에는 없는 2개의 고유 속성을 갖는다
  - $VOLUME_NAME 속성 : 볼륨의 유니코드 이름
  - **$**VOLUME_INFORMATION 속성 : NTFS 버전과 오류상태

$AttrDef 파일 개요

- MFT 엔트리 4번째에 위치
- 이 파일의 $DATA 속성은 각 속성 타입을 위해 이름과 타입 식별자를 정의함
- 이 파일은 각 파일 시스템들이 고유한 속성을 갖도록 하고 각 파일 시스템들이 표준 속성을 위해 식별자를 재정의 할 수 있도록 한다

분석 기술

- 파일 시스템의 설정과 레이아웃을 판단하기 위해 파일시스템 범주를 분석함
- NTFS 파일시스템의 첫 단계는 파일시스템 첫 섹터에 존재하는 부트섹터를 인지하는 것
  - 이 부트섹터는 $Boot파일의 일부분
  - 부트섹터는 각 MFT 엔트리의 존재하는 파일들의 시작 위치와 크기를 식별할 수 있도록 한다

분석 고려 사항

- 파일 시스템 범주에 사용자들이 이용하는 데이터는 많지 않음
- 부트섹터를 이용해서 기본적인 레이아웃을 확인 할 수 있다

- 헥사 편집기로 직접 확인에는 어려움이 있다
- 부트섹터 일부에는 반드시 0어야 한다
- 부트 섹터에 데이터를 숨길 수 있는 공간이 있음
    - $Boot 파일 끝에는 더 많은 공간 존재
- 볼륨과 파일 시스템의 슬렉공간 조사
- 파일 시스템 다음 영역 비사용 공간 확인

12.2. 내용범주

클러스터

- NTFS 파일은 속성의 집합이고 속성 중 일부는 거주로 NFT 엔트리에 저장하고 일부는 비거주
- 둘 다 클러스터에 내용을 저장한다
- 한 클러스터는 연속된 섹터의 그룹이며 클러스터 당 섹터 수는 2의 거듭제곱이다
- NTFS에서 클러스터들은 어떠한 파일이나 속성에 할당 될 수 있다
    - $Boot는 항상 첫 클러스터에만 할당

$BITMAP 파일 개요

- MFT 6번째 엔트리에 있는 $BITMAP 파일 시스템 메타 데이터 파일을 사용해서 결정한다
    - 파일 시스템 내의 모든 클러스터를 1개의 비트로 할당시키는 $DATA 속성

$BadClus 파일 개요

- NTFS는 불량 클러스터들을 MFT 엔트리 8인 $BadClus 에서 $DATA 속성에 할당해서 관리한다
- $Bsd 라고 부르는 $DATA 속성은 하나의 Sparse 파일이고 클러스터가 손상되었다고 보고될 때 속성이 더해진다

할당 알고리즘

- 할당정책은 운영체제에 기반하고, NTFS는 다른 할당정책을 이용할 수 있다
- 윈도우는 best-fit 을 사용해 자동맞춤을 하는데 여기서 처음이나 바로 다음에 공간이 있어도 가장 효율적인 공간에 데이터를 놓는다 때문에 적은 양의 데이터는 작은 클러스터 그룹에 들어갈 수 있다

파일시스템 레이아웃

- NTFS 모든 버전의 기본적인 한가지 개념은 MFT 구역
- MFT를 최소한으로 생성하고 더 많은 엔트리가 필요하면 확장한다
- 이렇게 하면 MFT가 한 파일에 할당한 이후에 단편화가 쉽게 될 수 있다는 위험이 있다
- 이것을 해결하기 위해 MFT를 위한 파일 시스템 일부분을 예약한다
- MFT 구역은 파일이나 디렉토리를 저장하지 않는 연속적인 클러스터들의 집합이다
- 기본적으로 윈도우는 파일 시스템의 12.5%를 MFT에 할당함

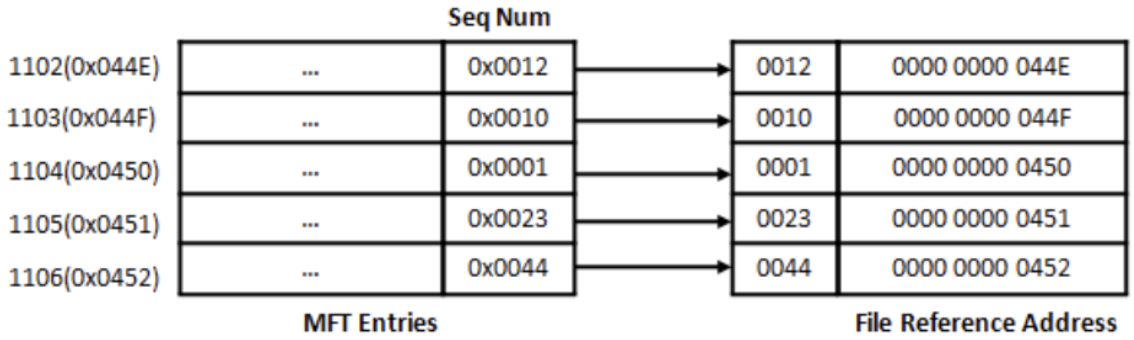
분석 기술

- 특정 클러스터 위치를 찾고, 할당상태를 확인하고, 내용을 처리하는 것을 포함
- 특정 클러스터 위치 찾기
  - 첫 클러스터는 파일 시스템의 시작
  - 그 클러스터 크기는 첫 섹터에서 주어지기 때문에 비교적 쉬움
- 할당 상태는 $Bitmap 위치를 찾고 그것의 $DATA 속성을 이용해 판단할 수 있다
- 삭제된 파일을 찾기위해 $Bitmap 상에서 0인 공간들을 추출하면 된다

분석 고려사항

- NTFS 는 오직 한가지 주소 지정 체계만 있으면 됨
- 파일 시스템의 모든 섹터는 클러스터에 할당
- FAT의 경우 FAT 영역과 예약된 영역에 할당 여부 확인이 명확하지 않음

12.3. 메타데이터 범주



$STANDARD_INFORMATION 속성

- 모든 파일, 디렉토리에 존재하고 핵심적인 메타데이터를 포함
- 시간, 날짜 스탬프 세트와 소유권, 보안, 할당 정책이 있음
  - 필수는 아니지만 MS가 제공하는 응용프로그램 수준에서 필요
- 속성의 4개 시간값은 아래와 같음
  - 생성시간 : 파일이 생성된 시간
  - 수정시간 : $DATA나 $INDEX 속성내용들이 마지막으로 수정된 시간
  - MFT 수정시간 : 파일 데이터가 마지막으로 수정된 시간, 파일의 속성을 선택했을 때 윈도우에서는 보여주지 않는다
  - 접근시간 : 파일의 내용에 마지막으로 접근된 시간
- NTFS 3.0+ 버전에서 이러한 속성은 보안정보와 응용프로그램 수준의 기능들을 위해 4개의 추가된 값들을 포함한다
  - 하나는 파일 소유자의 신원
  - 보안 ID 값은 $Secure 파일을 위한 인덱스로 사용되고 이 파일에 적용된 접근 제어 규칙을 결정하는데 사용
- USN (Update Seqence Number)를 포함

$FILE_NAME 속성

- 모든 파일과 디렉토리 MFT 엔트리 내에 최소 한개의 $FILE_NAME 속성이 있다
- 속성타입 식별자 48

- 파일 이름의 길이에 따라 유동적인 길이를 가짐
- 기본 길이는 66바이트
- $FILE_NAME 속성은 UFT-16유니코드로 인코딩된 파일 이름을 포함한다
- 반드시 8.3 도스 형식 win3, POSIX 형식
- $FILE_NAME 속성 부모 디렉토리를 위한 파일참조를 포함하고 이 파일참조는 그 속성이 MFT에 있을 때 가장 유용한 값 중 하나이다 → 도구가 MFT 엔트리 전체 경로를 더 쉽게 식별할 수 있도록 한다
- 또한 이 속성은 속성은 앞에 $STANDARD_INFOMATION 속성 절에서 설명한 4개의 비필수 값을 포함한다 윈도우는 이 값들을 파일 생성 이동 이름 변경시에만 변경
- $FILE_NAME 속성은 실제와 그리고 할당된 파일의 크기를 위한 필드들이 있지만 이런 값들은 사용자 파일들과 디렉토리에서는 0

$DATA 속성

- $DATA 속성은 정해진 값을 갖지 않음 어떠한 형태를 저장하기 위해 사용
- 속성타입 식별자는 128이고 0바이트 크기부터 어떠한 크기로도 가능
- $DATA속성은 각 파일에 할당되고 이름이 없음
- TSK를 포함한 도구에서 $DATA로 이름을 붙임
- 추가적으로 생성하는 $DATA 속성들은 MFT엔트리에 할당되고 그것들은 반드시 이름이 있음
- 추가적인 $DATA속성을 사용할 수 있음
- 윈도우 파일을 오른쪽 클릭해서 '속성'에 정보를 입력할 수 있고 그 정보는 $DATA속성에 저장
- 추가적인 $DATA속성은 데이터를 숨기는데 사용할 수 있고 디렉토리 목록 확인시 보이지 않음
- 대부분의 포렌식 도구들은 ADS(Alternate Data Streams)이라고 불리는 추가적인 $DATA 속성을 보여줌
- 사용자들은 명령 프롬프트에서 한개의 ADS를 쉽게 생성가능
    - 한개의 콜론 ':'을 이용해 ADS데이터와 속성이름을 저장한다

file.txt 파일에 foo라는 $DATA 속성을 생성하는 예

C:\>echo 'Hello There'>file.txt : foo

- $DATA는 데이터에 비인가된 접근을 막기위해 암호화하거나 공간을 절약하기 위해 압축할 수 있음
- 이러한 선택사항을 사용할 때 그 속성헤더는 해당 플래그를 설정
- 암호화를 사용할 때는 암호화 키를 저장하기 위해 $LOGGED_UTILITY_STREAM이 존재

$ATTRIBUTE_LIST 속성

- 모든 속성들을 저장하기 위해 MFT 엔트리 한 개 이상 필요로 할 때 사용된다

- 한개의 파일이나 디렉토리에는 65536개의 속성이 있을 수 있는데 이는 한 엔트리에 모두 넣을 수는 없음
- 최소한 한개의 MFT엔트리에 각 속성 헤더를 넣을 필요는 있다
- 내용은 비거주지만, 때로는 그 헤더들을 저장하기 위해 여러개의 MFT엔트리들이 필요
- $ATTRIBUTE_LIST 속성은 타입식별자가 32이고 두번째로 작음그
  - 항상 기준 MFT엔트리가 되며 자신을 제외한 파일의 모든 속성의 목록을 담는다. 목록의 각 엔트리는 속성타입과 어디에 위치하는지 알려주는 MFT엔트리 주소를 포함
- 속성은 단편화 될 가능성이 있고 runlist를 저장하기 위해 여러 MFT엔트리들을 요구할 수 있음
  - $DATA가 있는 경우에만 발생할 수 있음
  - 이것이 발생했을 때 각 비기준 MFT엔트리는 일반적인 $DATA 속성을 갖지만 그것은 파일 어디에 맞는지 식별함
  - 그 속성 헤더에는 논리적 파일주소인 run의 시작 VCN(Virtual Cluster Number)을 보여주는 한 필드가 있음

$Secure 파일

- 보안 식별자들은 파일이나 디렉토리 접근을 위한 접근 제어 정책을 정의하기 위해 사용된다
- NTFS버전 3.0이상에서 보안 식별자들은 MFT엔트리9에 위치하는 $Secure 파일시스템 메타데이터 파일에 저장
- 모든 파일과 디렉토리의 $STANDARD_INFOMATION 속성은 보안 ID(Security ID)라고 부르는 한개의 식별자를 포함하고 그 ID는 적절한 기술자를 확인하기 위해 $Secure 파일에 인덱스로 사용
  - 이러한 32비트 보안 ID들은 윈도우에서 사용자들에게 할당되는 보안 식별자(Security Identifiers, SID)와는 다름
  - SID는 전역으로 고유하고 보안 ID는 파일시스템에서 고유함
- $Secure 파일은 두개의 인덱스($SDH와 $SII)와 한개의 $DATA 속성($SDS)을 포함
- $DATA속성은 실제 보안기술자를 포함하고 두 인덱스들은 기술자를 참조하기 위해 사용됨
- $SII인덱스는 $STANDARD_INFOMATION 속성에 위치한 보안 ID로 정렬
- $SII인덱스는 그것의 보안 ID를 확인할 때 파일에 보안 기술자 위치를 확인하기 위해 사용된다
- $SDH인덱스는 보안 기술자의 해시에 의해 정렬된다. 운영체제는 새로운 보안 기술자가 파일이나 디렉토리에 적용될 때 이 인덱스를 사용
- 만약 새로운 기술자의 해시를 찾을 수 없다면 새로운 기술자와 보안 ID는 두 인덱스에 생성되고 더해짐

할당 알고리즘

- 메타데이터 할당 알고리즘에 대해 기술
- 응용프로그램에 종속적
- 메타데이터로 언급할 수 있는 3가지 전략
  - MFT 엔트리 할당
  - MFT 속성 할당

- 비필수 데이터를 업데이트

MFT 엔트리와 속성 할당

첫번째

- 윈도우는 엔트리 24를 기준으로 시작하고 '첫번째 적용' 알고리즘으로 MFT엔트리들을 할당한다는 것을 확인할 수 있었음
- 엔트리 0에서 15는 예약되어있고 그것들이 사용중이 아니어도 할당된 상태로 설정되며, 16~23은 전형적으로 할당되지 않음
- 사용자 파일들은 엔트리 24에서 시작하고 테이블 크기는 필요에 의해 증가됨
- 엔트리가 더이상 사용되지 않을때 사용중인지 구분하는 플래그를 제외하고는 변경되는 데이터는 없음
  - 비필수정보와 runlist는 복구가 가능
  - 엔트리가 다시 할당될 때 그것은 영구히 삭제되어 이전 파일 값들은 삭제
  - 그래서 MFT엔트리는 이전 파일의 슬랙데이터가 없음

두 번째

- MFT엔트리 속성에 공간을 할당하는 것
- 마이크로소프트는 속성타입으로 엔트리들을 정렬하고 계속해서 그것들을 정리함

시간 값 업데이트

- **생성시간**은 새로운 파일에 설정
  - 만약 새로운 파일을 생성하거나 복사를 하면 그 새로운 파일의 생성시간은 현재시간으로 설정된다
  - 파일을 이동하면 그 이동이 다른 볼륨으로 가더라도 생성시간은 원본 파일의 생성시간을 유지
- **마지막 수정시간**은 특정 $DATA, $INDEX_ROOT, 또는 $INDEX_ALLOCATION 속성 값이 수정될 때 설정
  - 파일을 이동하거나 복사하면 그 내용은 변하지 않음
  - 여러 $DATA가 있을 때 기본적이지 않은 속성이 수정되어도 시간은 업데이트 됨
  - 파일의 이름이나, 속성의 이름이 변경될 때는 업데이트 되지 않는다.
- **MFT수정시간**은 어떠한 속성이던지 변경될 때 설정
  - 한 파일의 내용을 변경하지 않고, 단지 열었을때도 설정 됨
  - 파일이름이 변경되면 $FILE_NAME 이 변경되므로 업데이트 됨
  - 파일이 다른 볼륨으로 이동시에는 업데이트 되지 않음
  - 같은 볼륨으로 이동할 때는 수정
- **마지막 접근시간**은 메타데이터 또는 내용을 봤을 때 설정

12.4. 파일 이름 범주

- 파일 이름과 내용을 연결하는데 사용되는 데이터를 말함

디렉토리 인덱스

- 헤더, $STANDARD_INFORMATIN, $FILE_NAME 속성에 특별한 플래그가 있는 일반적인 MFT엔트리를 가짐
- 디렉토리 인덱스의 인덱스 엔트리들은 파일 참조 주소와 $FILE_NAME 속성 포함
- $FILE_NAME 속성은 파일의 이름, 비필수 정보, 크기 , 기본 플래그들을 가지고 있음

루트 디렉토리

- 파일 시스템에서 전체 경로로 한 개의 파일을 찾으려면 루트 디렉토리의 위치를 아는 것이 중요
- 루트 디렉토리는 항상 MFT 엔트리 5에 위치
  - 이 엔트리는 표준 $INDEX_ROOT, $INDEX_ALLOCATION 그리고 $BITMAP 속성을 가짐
  - 모든 파일 시스템 메타데이터 파일들은 이 디렉토리에 위치

파일과 디렉토리 연결

- NTFS는 하드링크를 이용하여 파일이 한 개 이상의 이름을 갖도록 함
- 하드 링크는 원래의 파일 이름 연결 방식과 차이가 없음
- 원본과 동일한 엔트리를 가리키도록 부모 디렉토리 인덱스 엔트리에 할당
- MFT 헤더에 있는 링크 카운트는 하드 링크가 생성될 때 마다 하나씩 증가

→ 원본 파일 이름이 지워졌지만 하드 링크가 여전히 존재한다면 그 파일은 지워진 것이 아님

- NTFS 3.0+ 버전에서는 파일 디렉토리가 볼륨을 연결하는데 사용되는 재파싱 지점이라는 기능이 있음
- 재파싱 지점은 무엇을 연결하고 있는지에 대한 정보를 포함하는 특별한 파일이나 디렉토리
- 심볼릭 링크는 두 파일을 연결하는 재파싱 지점이고 접합은 두 디렉토리를 연결하는 것
- 마운트 지점은 디렉토리와 볼륨을 연결하는 것
- 윈도우 원격 저장 서버 기능은 파일이나 디렉토리의 서버위치를 설명하기 위해 재파싱 지점을 사용한다
  - 재파싱지점은 특별한 파일들이고 $STANDARD_INFORMATION과 $FILE_NAME속성들의 플래그 세트를 갖는다

오브젝트 식별자

- NTFS 3.0 버전에서 파일 이름, 디렉토리 이름, MFT 엔트리 주소를 사용하는 것 외 이것들을 주소화하는 두번째 방법
- 운영체제나 응용프로그램들은 각 파일의 고유한 128 비트 오브젝트 식별자를 할당하고 파일이름이 변경되거나 다른 볼륨으로 이동할 때 파일을 참조하기 위해 사용된다

할당알고리즘

- NTFS는 B-Tree 를 이용하고 할당하는 방법이 다양함
- 너무 많은 노드 존재시 트리를 나누고 새로운 레벨 생성
- 파일을 삭제할 때 엔트리는 트리에서 제거되며 노드에 남아있는 엔트리는 비워

12.5. 응용프로그램 범주

- NTFS는 많은 응용프로그램 수준의 기능을 지원하는 파일 시스템
- 파일 시스템에서 필요하지는 않지만 응용프로그램이나 운영체제들이 효율적으로 운영되기 위해서 필요함
- 디스크 할당, 로깅, 변경, 저널링에 대해 기술

디스크 할당

- NTFS는 디스크 공간 할당을 지원함
- 관리자는 사용자별로 사용할 수 있는 공간을 제한하는 할당을 설정할 수 있음
- 파일시스템 데이터로 저장되고 다른 데이터는 레지스트리 같은 응용프로그램 수준의 파일들에 저장

분석 고려사항

- 운영체제가 파일 시스템을 사용할 때 할당상태를 반드시 이용하는 것은 아님
- 할당상태는 부가적인 사항으로 봐야함
- 다른 운영체제에서 NTFS 파일 시스템을 마운트하면 사용자가 파일을 생성할 때 할당 상태를 업데이트 하지 못함
- 할당은 포렌식 조사동안 어떤 사용자들이 큰 데이터를 가지고 있었는지 판단할 때에 유용

로깅 - 파일시스템 저널링

- 파일 시스템의 신뢰성을 높이기 위해 MS 에 추가한 기능
- 충동이 일어나기 전 메타데이터 업데이트에 대한 정보를 기록하고 언제 업데이트들이 일어났는지 기록

분석 고려사항

- 파일 시스템에서 변경된 정보를 제공하지만 그것들이 덮어써지기 전에 어떤 엔트리들이 었었는지 확인 할 수 없고 그 저널 파일이 어떤 식으로 구성되어있는지 알 수 없음
- 증거를 찾더라도 설명하기 어려움

변경저널

- 파일과 디렉토리들이 변경될 때 기록하는 파일
- 변경로그 저널은 변경되었던 파일들을 목록화 하기 때문에 어떤 파일들이 변경되었는지 판단 과정을 더욱 쉽게 만들 수 있음
- ＼$Extend＼$UsrJrnl 파일에 저장되어 있음

분석 고려 사항

- 이 저널 기능이 활성화 되었다 보장할 수 없기 때문에 파일로부터 얼마나 많은 정보를 모을 수 있는지 정확하지 않음
- 비활성화시 윈도우는 내용을 모두 지워버림
- 내용들을 신뢰할 수 있다면 최근에 발생한 사건들을 재구성하는데 유용할 수 있음

12.6. 큰 그림

파일 할당 예제

- 파일 ＼dir1＼file1.dat을 생성하고, dir1 디렉토리는 루트 디렉토리에 이미 존재한다고 가정하자. 파일의 크기는 4000바이트이고, 각 클러스터는 2048 바이트이다.

1. 파일시스템의 첫 번째 섹터를 읽고, 부트섹터는 클러스터 크기, MFT시작 주소, 각 MFT 엔트리 크기를 결정한다.

2. $MFT 파일인 MFT의 첫 번재 엔트리를 읽어 $DATA 속성에 있는 MFT 나머지 레이아웃을 결정한다.

3. 먼저 새로운 파일을 위해 한 개의 MFT엔트리를 할당해야 한다. 비할당 엔트리를 찾기 위해 $MFT파일의 $BITMAP 속성을 해석한다. 첫 번째 free 엔트리, 엔트리 304는 새로운 파일에 할당되고, 비트맵 해당 비트는 1로 설정된다.

4. MFT내의 위치를 구하고, 내용을 지워 MFT엔트리 304를 초기화한다. $STANDARD_INFORMATION과 $FILE_NAME 속성이 생성되고, 시간 값들은 현재 시간으로 설정된다. '사용 중' 플래그가 MFT 엔트리 헤더에서 설정된다.

5. 다음은 MFT 엔트리 6인 $BITMAP 파일의 $DATA 속성을 이용해서 파일의 두 클러스터를 할당하는 것이다. 이 파일에는 2개의 클러스터가 필요하며, 두 개의 연속적인 클러스터 692, 693을 '자동 맞춤' 알고리즘을 이용해 찾는다. $BITMAP의 그 클러스터들에 해당 비트들은 1로 설정된다. 파일 내용은 클러스터에 써지며, 해당 $DATA속성은 그 클러스터 주소들로 업데이트된다. 그 MFT엔트리는 수정되고 파일 수정 시간들은 업데이트된다.

6. 다음 단계는 추가할 파일을 위해 파일 이름 엔트리를 더하는 것이다. MFT엔트리 5에 있는 루트 디렉토리는 dir1에 위치한다. $INDEX_ROOT와 $INDEX_ALLOCATION 속성들을 읽고, 정렬된 트리를 탐색한다. 그 dir1 인덱스 엔트리를 찾고, 그것의 MFT엔트리 주소는 200이다. 디렉토리의 마지막 접근 시간은 업데이트된다.

7. MFT엔트리 200을 구하고, file1.dat가 어디에 위치하는지 찾기 위해 $INDEX_ROOT 속성을 처리한다. 새로운 인덱스 엔트리가 그것을 위해 생성되고, 그 트리는 재정렬 된다. 이것은 그 노드 주위 인덱스 엔트리들에 결과이다. 새오룬 인덱스 엔트리는 파일 참조 주소에 MFT엔트리 304를 갖고, 시간들과 플래그들은 적절하게 설정된다. 그 디렉토리의 마지막 생성, 수정, 접근 시간들은 업데이트 된다.

8. 이전 각 단계들에서 $LogFile 파일시스템 저널, 그리고 ＼$Extend＼$UsrJrnl 변경 저널에 엔트리들을 만들어 왔다. 할당이 활성화되어 있었다면 새로운 파일의 크기가 ＼$Extend＼$Quota 사용자 할당에 더해졌을 것이다.

파일 삭제 예제

- \dir1\file1.dat 파일이 지워질 때 어떤 과정이 발생하는가

1. 파일시스템의 첫 번째 섹터를 읽고, 부트섹터는 클러스터 크기, MFT시작 주소, 각 MFT 엔트리 크기를 결정한다.

2. $MFT 파일인 MFT의 첫 번재 엔트리를 읽어 $DATA 속성에 있는 MFT 나머지 레이아웃을 결정한다.

3. dir1 디렉토리를 찾기 위해 루트 디렉토리, MFT 엔트리 5를 해석하고, $INDEX_ROOT와 $INDEX_ALLOCATION속성들에 인덱스를 탐색한다. dir1 엔트리를 찾고, MFT 엔트리 주소는 200이다. 디렉토리의 마지막 접근 시간은 업데이트된다.

4. MFT엔트리 200의 $INDEX_ROOT 속성을 처리하고, file1.dat 엔트리를 검색한다. 그 파일에 MFT 엔트리 주소가 엔트리 304라는 것을 확인할 수 있다.

5. 인덱스에서 그 엔트리를 제거하고, 그 노드에 엔트리들은 이동되고, 원래 엔트리로 덮어 써진다. 디렉토리에 마지막 생성, 수정, 접근 시간들이 업데이트 된다.

6. 사용중 플래그를 정리하는 것으로 MFT엔트리 304를 비할당한다. 또한 $BITMAP 파일의 $DATA속성을 처리하고, 이 엔트리에 해당하는 비트를 0으로 설정한다.

7. MFT 엔트리 304의 비거주 속성을 처리하고 ＼$BITMAP 파일에서 그 해당 클러스터들에 해당하는 비트를 비할당 상태로 설정한다. 클러스터 692, 693의 비트가 비할당된다.

8. 이전 각 단계들에서 $LogFile 파일시스템 저널, 그리고 ＼$Extend＼$UsrJrnl 변경 저널에 엔트리들을 만들어 왔다. 할당이 활성화 되어 있었다면 ＼$Extend＼$Quota 사용자 할당에서 파일 크기를 뺀다.

12.7. 다른 주제

- 특정 데이터 범주에 적용할 수 없는 지워진 파일 복구와 파일 시스템 일관성 검사에 대해 언급

파일 복구

- NTFS는 **파일이 지워지면 그 이름은 부모 디렉토리 인덱스에서 제거되고 그 클러스터들은 비할당 됨**
  - 마이크로소프트는 어떠한 포인터들도 삭제하지 않지만 향후에는 삭제할 수도 있다.
- NTFS의 단점은 파일 이름을 부모 인덱스에서 제거시 그 인덱스가 재정렬되고 이름정보를 잃어버릴 수 있음

일관성 검사

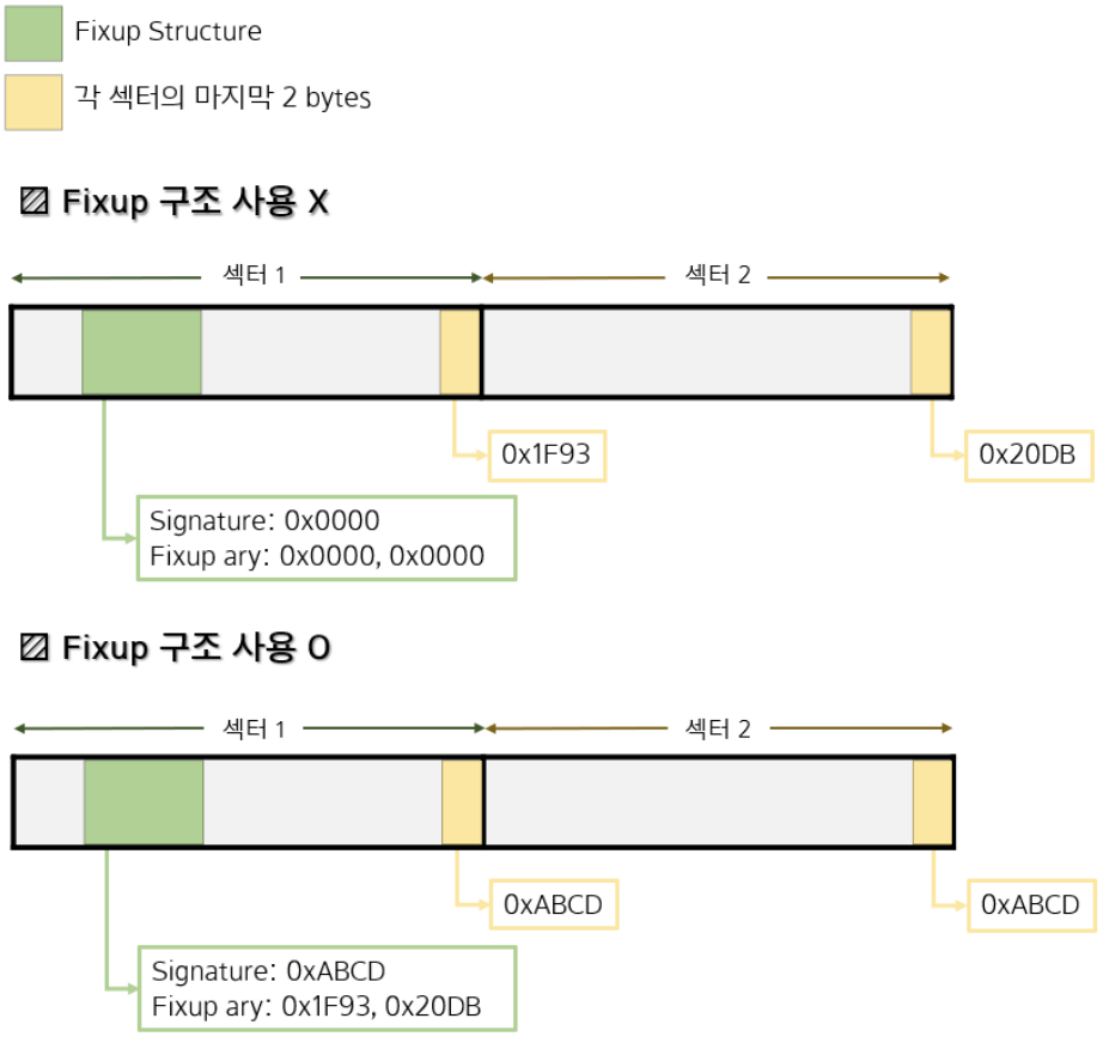
- 일관성 검사는 이미지가 손상되었거나, 변경여부를 탐지하는데 사용할 수 있음
- NTFS 부트섹터에 적은 양의 데이터가 있지만 마이크로소프트는 사용하지 않는 일부 값들을 0으로 만듬
- NTFS 파일시스템은 너무 유연하고, 많은 옵션들을 지원하기 때문에 분석이 쉽지 않음

Chapter 13. NTFS 데이터 구조

13.1. 기본 개념

- NTFS 기본 데이터 구조체 개념
- 데이터 구조체의 신뢰성을 제공하는 구조체의 설계특정 분석 후 MFT 엔트리와 속성 헤더의 데이터 구조체 설명

Fixup 값



- 신뢰성을 향상시키기 위해 사용하는 저장 기술 존재
- NTFS는 한 섹터가 넘는 길이를 갖는 데이터 구조체와 Fixup 결합함
- Fixup 값과 큰 데이터 구조체의 마지막 두 바이트는 데이터 구조체가 디스크에 써질 때 시그니처 값으로 교체 됨
  - 그 시그니터 값을 리용해 모든 섹터의 시그니처가 같은지 확인 후 무결성을 검증하는데 사용
- FIxup은 오직 데이터 구조체에만 사용
  - 파일 내용을 포함하는 섹터가 없는 경우에는 사용하지 않음
  - Fixup을 사용하는 데이터 구조체는 현재의 16비트 시그니처 값을 식별하는 헤더 필드와 원본 값들을 담고 있는 배열이 있음
- 데이터 구조체가 디스크에 써질 때 그 시그니처 값은 1씩 증가
- 각 섹터의 마지막 2바이트는 배열로 복사
- 시그니처의 값은 각 섹터의 마지막 2바이트에 써짐

MFT 엔트리 (파일 레코드)

Byte Range	Description	Essential
0 – 3	Signature ("FILE")	No
4 – 5	Offset to fixup array	Yes
6 – 7	Number of entries in fixup array	Yes
8 – 15	$LogFile Sequence Number (LSN)	No
16 – 17	Sequence Number	No
18 – 19	Link count	No
20 – 21	Offset to first attribute	Yes
22 – 23	Flags (in-use and directory)	Yes
24 – 27	Used size of MFT entry	Yes
28 – 31	Allocated size of MFT Entry	Yes
32 – 39	File reference to base record	No
40 – 41	Next attribute id	No
42 – 43	Align to 4B boundary	No
44 – 47	Number of this MFT Entry	No

- MFT는 NTFS의 핵심
- 파일과 디렉토리에 대한 엔트리를 가짐
- 크기가 일정하고 단지 몇개의 필드만 저장
- 보통 엔트리 크기는 1024 바이트
- 부트섹터에서 저장
- MFT 엔트리는 Fixup 값들을 사용하므로 윈도우 NTFS의 데이터 구조체는 Fixup 값으로 교체된 각 섹터의 마지막에 두 바이트(시그니처) 가 있다

속성 헤더

범위(10진수)	범위(16진수)	설명
0 – 3	0x00 – 0x03	속성 타입 식별자
4 – 7	0x04 – 0x07	속성 길이
8 – 8	0x08 – 0x08	Non-resident 플래그
9 – 9	0x09 – 0x09	속성 이름 길이 (N)
10 – 11	0x0A – 0x0B	속성 이름 시작 위치
12 – 13	0x0C – 0x0D	상태 플래그
14 – 15	0x0E – 0x0F	속성 식별자

- MFT 엔트리는 속성으로 채워지고 각 속성은 같은 헤더 데이터 구조체를 가짐
- 거주와 비거주 속성의 구조체는 다름
  - 비거주 속성에는 run 정보가 있다
- 이런 값들은 타입, 크기, 이름 위치 등의 속성에 대한 기본적인 정보를 제공한다
- 크기는 MFT 엔트리 다음 속성을 찾는데 사용되고 송성 다음에 0xffffffff가 존재하면 그것이 마지막이라고 인식한다
  - 속성이 비거주일때 플래그는 1
    - 그 플래그 값은 속성이 압축되었는지 (0x00001), 암호화 되었는지 (0x40000) 또는 Sparse 되었는지 (0x8000) 를 구분한다
  - 속성 식별자는 이 MFT 에서 속성에 대한 고유 번호이고 이름의 오프셋은 속성 시작에서 상대적

- 비거주 속성은 클러스터 runs 의 임의의 숫자를 설명해야하기 때문에 다른 데이터 구조체를 갖는다

범위(10진수)	범위(16진수)	설명
0 – 15	0x00 – 0x0F	공통된 헤더
16 – 23	0x10 – 0x17	런리스트 시작 VCN(Virtual Cluster Number)
24 – 31	0x18 – 0x1F	런리스트 끝 VCN
32 – 33	0x20 – 0x21	런리스트 시작 위치
34 – 35	0x22 – 0x23	압축 단위 크기
36 – 39	0x24 – 0x27	사용되지 않음
40 – 47	0x28 – 0x2F	속성 내용 할당 크기(클러스터 크기)
48 – 55	0x30 – 0x37	속성 내용 실제 크기
56 – 63	0x38 – 0x3F	속성 내용 초기화된 크기
64 – 64+2N	0x40 – 0x40+2N	속성 이름(유니코드) 단, 이름이 존재할 경우만 사용
64+2N+1 –	0x40+2N+1 –	속성 내용

- VCN 은 8장 파일시스템 분석에서 정의했던 논리적 파일 주소에 대한 다른 이름
  - 시작과 끝 숫자는 여러 MFT 엔트리 들이 한개의 속성을 설명할 필요가 있을 때 사용된다
  - 예를 들어 $DATA 속성이 매우 단편화 되어 있고 그것의 run이 MFT에 적합하지 않을 때 그것의 두번째 엔트리는 첫번째 엔트리의 마지막 VCN 다음 주소와 동일한 시작 VCN을 갖는 $DATA 속성을 포함한다

- 데이터 runlist 에서 오프셋은 속성 시작에서 상대적
  - Runlist 형식은 매우 효율적이지만 조금 혼란스러울 수 있다
  - 길이가 유동적이지만 최소 1바이트여야만 한다
  - 데이터 구조체 첫 바이트는 상위 4비트와 하위 4비트로 구성 (나블)
  - 4개의 최하위 비트들은 헤더 바이트 다음에 오는 run 길이 필드의 바이트 수를 저장한다
  - 최상위 4바이트 들은 길이 필드 다음에 오는 run 오프셋 필드에 있는 바이트 수를 저장한다

13.2. 표준 파일 속성

$STANDARD_INFORMATION 속성

범위(10진수)	범위(16진수)	설명
–	–	속성 헤더 (Attribute header)
0 – 7	0x00 – 0x07	생성 시간 (Creation time)
8 – 15	0x08 – 0x0F	수정 시간 (Modified time)
16 – 23	0x10 – 0x17	MFT 수정 시간 (MFT modified time)
24 – 31	0x18 – 0x1F	접근 시간 (Last accessed time)
32 – 35	0x20 – 0x23	속성 플래그 (Flags)
36 – 39	0x24 – 0x27	버전 최대값 (Maximum number of versions)
40 – 43	0x28 – 0x2B	버전 번호 (Version number)
44 – 47	0x2C – 0x2F	클래스 ID (Class ID)
48 – 51	0x30 – 0x33	소유자 ID (version 3.0+)
52 – 55	0x34 – 0x37	보안 ID (version 3.0+)
56 – 63	0x38 – 0x3F	Quota Charged (version 3.0+)
64 – 71	0x40 – 0x47	USN (Update Sequence Number) (version 3.0+)

- 타입 식별자 16
- 항상 거주속성
- 파일이나 디렉토리의 기본 메타데이터를 포함한다
- 모든 파일과 디렉토리에 존재하고 가장 낮은 타입 식별자를 갖기 때문에 일반적으로 첫번째 속성

- 4개의 시간값들은 1601년 1월 1일 UTC 부터 100나노 단위로 저장됨
- 같은 시간 필드들이 $FILE_NAME 속성에도 있지만 윈도우가 파일 속성에서 보여주는 시간값은 이 속성 정보이며 업데이트 되는 정보도 마찬가지임

플래그 값	설명
0x0001	읽기 전용 (Read Only)
0x0002	숨긴 파일 (Hidden)
0x0004	시스템 (System)
0x0020	아카이브 (Archive)
0x0040	장치 (Device)
0x0080	일반 (Normal)
0x0100	임시 (Temporary)
0x0200	Sparse 파일
0x0400	Reparse Point
0x0800	압축됨 (Compressed)
0x1000	오프라인 (Offline)
0x2000	빠른 검색을 위해 인덱스 되지 않은 내용
0x4000	암호화됨 (Encrypted)

속성 플래그 값에 대한 정보

- icat 도구를 이용하여 속성타입을 지정해서 속성을 볼 수 있다

$FILE_NAME 속성

범위(10진수)	범위(16진수)	설명
–	–	속성 헤더 (Attribute header)
0 – 7	0x00 – 0x07	부모 디렉터리의 파일 참조 주소 (File Reference of parent directory)
8 – 15	0x08 – 0x0F	생성 시간 (Creation time)
16 – 23	0x10 – 0x17	수정 시간 (Modified time)
24 – 31	0x18 – 0x1F	MFT 수정 시간 (MFT modified time)
32 – 39	0x20 – 0x27	접근 시간 (Last accessed time)
40 – 47	0x28 – 0x2F	파일 할당 크기 (Allocated size of file)
48 – 55	0x30 – 0x37	파일 실제 크기 (Real size of file)
56 – 59	0x38 – 0x3B	속성 플래그 (Flags)
60 – 63	0x3C – 0x3F	Reparse 값 (Reparse value)
64 – 64	0x40 – 0x40	이름 길이 (Length of name)
65 – 65	0x41 – 0x41	이름 형식 (Namespace)
66 –	0x42 –	이름 (Name)

- 타입 식별자 48
- 파일 이름과 부모 디렉토리 정보를 저장하기 위해 MFT 엔트리에 위치
- 디렉토리 인덱스에서 사용됨
- MFT 엔트리에서 사용될 때는 그 어떤 필수적인 정보도 포함하지 않지만 디렉토리 인덱스에서 사용할 때는 그렇지 않음
- 표준 파일이나 디렉토리에서 이것은 두번째 속성이고 항상 거주임
- 만약 한 파일에 여러 MFT 엔트리가 필요하면 $ATTRIBUTE_LIST 속성이$STANDARD_INFORMATION 과 이 속성($FILE_NAME) 사이에 존재

$FILE_NAME 속성 필드

분석 예제

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
044B7000	46	49	4C	45	30	00	03	00	18	C3	60	C1	02	00	00	00	FILE0.....`.....
044B7010	09	00	01	00	38	00	01	00	68	01	00	00	00	04	00	00	....8...h.......
044B7020	00	00	00	00	00	00	00	00	03	00	00	00	DC	12	01	00	....
044B7030	E5	00	00	00	00	00	00	00	10	00	00	00	60	00	00	00	............`...
044B7040	00	00	00	00	00	00	00	00	48	00	00	00	18	00	00	00	........H.......
044B7050	95	02	35	CC	F3	97	CB	01	3C	79	89	E0	3D	02	CC	01	..5.....<y..=...
044B7060	3C	79	89	E0	3D	02	CC	01	CA	A5	F3	59	A7	B8	CB	01	<y..=......Y....
044B7070	26	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	&...............
044B7080	00	00	00	00	CD	0C	00	00	00	00	00	00	00	00	00	00	....
044B7090	78	FD	71	8C	00	00	00	00	30	00	00	00	78	00	00	00	x.q.....0...x...
044B70A0	00	00	00	00	00	00	02	00	5A	00	00	00	18	00	01	00	........Z.......
044B70B0	05	00	00	00	00	00	05	00	95	02	35	CC	F3	97	CB	01	..........5.....
044B70C0	CA	A5	F3	59	A7	B8	CB	01	CA	A5	F3	59	A7	B8	CB	01	...Y.......Y....
044B70D0	CA	A5	F3	59	A7	B8	CB	01	00	D0	E0	FF	01	00	00	00	...Y............
044B70E0	00	00	00	00	00	00	00	00	26	00	00	00	00	00	00	00	........&.......
044B70F0	0C	03	70	00	61	00	67	00	65	00	66	00	69	00	6C	00	..p.a.g.e.f.i.l.
044B7100	65	00	2E	00	73	00	79	00	73	00	00	00	00	00	00	00	e...s.y.s.......
044B7110	80	00	00	00	50	00	00	00	01	00	00	00	00	00	01	00	....P...........
044B7120	00	00	00	00	00	00	00	00	0C	FE	1F	00	00	00	00	00	....
044B7130	40	00	00	00	00	00	00	00	00	D0	E0	FF	01	00	00	00	@...............
044B7140	00	D0	E0	FF	01	00	00	00	00	D0	E0	FF	01	00	00	00	....
044B7150	43	0D	FE	1F	44	DE	F7	00	00	00	00	00	00	00	00	00	C...D...........
044B7160	FF	FF	FF	FF	82	79	47	11	00	00	00	00	00	00	00	00	.....yG.........

- 부모 디렉터리의 파일 참조 주소
  - 순서 번호 : 0x0005
  - MFT 엔트리 주소 : 0x000000000005 (루트 디렉터리의 MFT 엔트리 번호)
- 생성 시간
  - 값 : 0x01CB97F3CC350295
  - 변환 시간 : Fri, 10 December 2010 07:52:46 KST
- 수정 시간
  - 값 : 0x01CBB8A759F3A5CA
  - 변환 시간 : Thu, 20 January 2011 22:38:41 KST
- MFT 수정 시간
  - 값 : 0x01CBB8A759F3A5CA
  - 변환 시간 : Thu, 20 January 2011 22:38:41 KST
- 접근 시간
  - 값 : 0x01CBB8A759F3A5CA
  - 변환 시간 : Thu, 20 January 2011 22:38:41 KST
- 파일 할당 크기
  - 값 : 0x01FFE0D000
  - 크기 : 8,587,890,688 (7.99 GB)
- 파일 실제 크기
  - 값 : 0x00
  - 크기 : 0 바이트
- 속성 플래그
  - 값 : 0x00000026
  - 적용 속성 : 아카이브, 시스템, 숨긴 파일

- Reparse 값 : 0x00000000
- 이름 길이 : 0x0C (12)
- 이름 형식
  - 값 : 0x03
  - 적용 형식 : Win32 & DOS
- 이름 : pagefile.sys

$DATA 속성

- 고유한 구조체가 없음
- 헤더 이후에는 파일 내용에 해당하는 미가공 데이터만 있음
- 타입 식벼라 128
- 최댓값이나 최솟값이 존재하지 않음
- 내용이 700바이트 이상이면 비거주 속성이 됨
- 대부분의 파일에서 MFT 마지막 속성
- 디렉토리는 인덱스 속성들 이외에 $DATA 속성들을 가질 수 있음

$ATTRIBUTE_LIST 속성

- MFT 엔트리 내에 있고 다른 속성들이 있을 수 있는 위치를 알려주는 역할
- 하나의 MFT 엔트리에 적합하지 않은 속성 헤더를 갖는 파일에 사용
- 파일이나 디렉토리에 있는 모든 속성의 엔트리 목록을 포함
- 속성 타입 식별자 32를 가짐

- 각 리스트 엔트리 필드

범위(10진수)	범위(16진수)	설명
–	–	속성 헤더
0 – 3	0x00 – 0x03	속성 타입
4 – 5	0x04 – 0x05	엔트리 길이
6 – 6	0x06 – 0x06	이름 길이
7 – 7	0x07 – 0x07	이름 시작 위치
8 – 15	0x08 – 0x0F	속성의 시작 VCN
16 – 23	0x10 – 0x17	속성 위치의 File Reference 주소
24 – 24	0x18 – 0x18	속성 ID

$OBJECT_ID 속성

- 64타입 식별자
- 이름 대신 파일을 나타내는 128비트 전역 오브젝트 식별자를 나타냄
- ＼$Extend＼$Objid 인덱스
  - 파일의 오브젝트 ID로 정렬
  - 각 파일에서 볼 수 있는 파일참조 주소를 포함
  - 그 속성에는 단지 4개의 필드만 존재

범위(Byte)	설명
0~15	오브젝트 ID
16~31	파일이 생성된 볼륨 ID
32~47	파일이 생성될 때 받은 오브젝트 ID
48~63	파일이 생성될 때 받은 도메인 ID

$REPARSE_POINT 속성

- 속성 식별자 192
- 재파싱 지점들의 파일들에 사용
- 재파싱 지점은 심볼릭 링크와 접합 그리고 볼륨을 위한 마운트에 사용

범위(Byte)	설명	범위(Byte)	설명
0~3	Reparse 타입 플래그	**10~11**	대상 이름 길이

4~5	Reparse 데이터 크기	**12~13**	대상의 출력 이름 오프셋
6~7	사용 x	**14~15**	출력 이름 길이
8~9	대상 이름 오프셋		

13.3. 인덱스 속성과 데이터 구조

- 인덱스 : 정리된 트리에 데이터 구조체가 있다는 것
  - 그 트리는 한 개 이상의 노드를 갖고 각 노드는 한 개 이상의 인덱스 엔트리를 갖는다
  - 트리는 루트 $INDEX_ROOT 속성에 위치
  - 다른 노드는 $INDEX_ALLOCATION 속성 인덱스 레코드에 위치
  - $BITMAP 속성은 인덱스 레코드들이 할당상태를 관리하기 위해 사용

$INDEX_ROOT 속성

- 거주 속성
- 144타입 식별자
- 오직 인덱스 엔트리의 작은 목록만 저장 가능

$INDEX_ALLOCATION

- 큰 디렉토리는 인덱스 엔트리가 거주 $INDEX_ROOT 속성에는 적합하지 않기 때문에 비거주 **$INDEX_ALLOCATION 속성 필요**
- 인덱스 레코드 크기는 $INDEX_ROOT 속성 헤더에 정의
  - 보통 4096바이트
- $INDEX_ALLOCATION 속성은 타입 식별자가 160이며 $INDEX_ROOT 속성 없이는 존재할 수 없음

$BITMAP 속성

- 많은 파일을 삭제한 후 또는 각 클러스터가 한 인덱스 레코드들보다 큰 경우 디렉토리는 불필요한 레코드들을 가질 수 있음
- 이 속성은 또한 각 MFT엔트리가 할당되었는지를 추적하기 위해 $MFT에 의해서 사용

- $BITMAP 속성은 176 타입 식별자를 가지고 있고 바이트들에 의해 구성되며 또 각 비트는 인덱스 레코드에 해당
- 이전 디렉토리를 위한 $BITMAP 속성을 icat을 사용해 볼 수 있음

인덱스 노드 헤더 데이터 구조체

- 인덱스 엔트리 목록이 시작하고 끝나는 위치를 설명하는 데 사용

일반 인덱스 엔트리 데이터 구조체

- 인덱스 엔트리 데이터에 대한 일반적인 구조체를 설명

디렉토리 인덱스 엔트리 데이터 구조체

- 파일이름을 사용하는 디렉토리 인덱스는 특정 인덱스 엔트리 데이터 구조체를 가짐
- 이전 절에서 개괄적으로 설명한 기본 템플릿을 사용하고 파일참조주소, $FILE_NAME 속성을 포함

13.4 파일시스템 메타데이터 파일

$MFT 파일

- MFT 엔트리 0에 위치
- 고유 속성은 MFT 엔트리 할당 상태를 관리하는 $BITMAP 속성임
  - 그 속성들은 바이트들로 구성되고 한 비트가 1로 설정될 때 그 엔트리는 할당된 것
  - 반대로 비트가 0이면 엔트리는 할당되지 않은 것
  - icat 도구에 속성타입 176을 지정해서 $BITMAP 속성 확인 가능

$Boot 파일

- MFT엔트리 7에 해당하고 부트섹터와 $DATA 속성이 있는 부트코드를 포함

- 이 속성은 항상 섹터 0에서 시작하고, 부트섹터 데이터 구조체가 위치
- 사용하지 않은 필드들은 FAT 부트섹터의 BIOS Parameter Block(BPB) 필드들에 해당
  - 윈도우는 이 값들이 0이아니면 디스크를 마운트 할 수 없음
- 부트섹터에서 가장 중요한 값은 각 섹터와 클러스터 크기
  - 이것들 없이는 어떠한 위치도 확인 불가능
- 두번째로 중요한 값은 MFT의 시작위치와 각 엔트리의 크기
  - MFT엔트리들은 항상 1024바이트였지만 앞으로는 그 크기를 쉽게 변경할 수 있도록 이 필드가 존재함
  - 또 $MFTMirr의 $DATA 속성의 주소가 주어짐
    - 이 주소는 복구도구들이 $MFT엔트리의 백업본이 어디에 있는지 결정하도록해서 MFT의 위치를 판단할 수 있도록 함

$AttrDef 파일

- $AttrDef 파일시스템 메타데이터 파일은 MFT엔트리 번호가 4
- 파일시스템 속성 이름과 식별자를 정의

$BITMAP 파일

- MFT엔트리 6에 위치한 $BITMAP파일은 클러스터 할당상태를 관리하고 $DATA 속성을 가짐
- 비트맵 데이터는 1바이트 값들로 구성되고, 각 바이트 최하위비트는 이전 바이트의 최상위 비트에 해당하는 클러스터 다음에 오는 클러스터에 해당
  - 예를들어 바이너리 값 00000001 과 00000011을 고려해본다면 첫 바이트는 클러스터 0에 해당하는 최하위 비트가 1
  - 그다음 7개는 모두 0이기 때문에 클러스터 0에 해당하는 최하위 비트가 1이다. 그 다음 7개는 모두 0이기 때문에 1~7은 할당되지 않았음을 알 수 있다
  - 두번째 바이트의 최하위 비트 2개가 1이므로 클러스터 8과 9는 할당이 되었음을 알 수 있다. 최하위 비트에서 이것들을 확인하기 위해 오른쪽에서 왼쪽으로 읽고, 그러고나서 오른쪽 바이트로 이동한다
- 주어진 클러스터의 할당상태를 결정하기 위해 그것이 위치한 비트맵의 바이트를 결정하는 것이 필요
  - 이것은 클러스터 주소를 8로 나누고, 나머지는 무시함
    - 예를들어 클러스터 5는 비트맵 바이트 0에 있고, 클러스터 18은 비트맵 바이트 2에 있다. 그 클러스터에 해당하는 바이트에 그 비트를 찾기 위해 그 나머지를 조사

$VOLUME 파일

- MFT엔트리 3에 존재
- 두 속성을 가짐

$VOLUME_NAME 속성

- 타입식별자 96을 가짐
- 오직 $VOLUME 파일을 할당한 경우만 지원
- UTF-16 유니코드 볼륨이름을 포함하는데 그 외에는 아무것도 없음

$VOLUME_INFORMATION 속성

- 식별자 112를 갖는 $VOLUME_INFORMATION 속성

$ObjId 파일

- 이 ID는 파일이름을 변경하더라도 여전히 그 파일을 찾을 수 있도록 함

$Quota 파일

- INDEX_ROOT와 $INDEX_ALLOCATION 속성사용
- $O 인덱스는 한 SID와 소유자 SID를 연결
- $Q 인덱스는 할당정보에 자산의 ID를 연결
- $O 인덱스를 위한 인덱스 엔트리는 아래에서 주어지는 필드들을 가짐

$LogFile 파일

- MFT 엔트리 2이고 NTFS 저널로 사용
- 표준파일 속성을 갖고, $DATA 속성에 로그데이터를 저장

$UsrJrnl 파일

- 변경 저널 프로그램은 응용프로그램 범주에 속함
- 파일에서 변경이 발생할 때 기록 변경은 예약된 MFT엔트리에 위치하지 않는 ＼$Extend＼$UsrJrnl 파일의 **$J**라는 이름의 $DATA 속성에 기록
- $J $DATA 속성은 Sparse
  - 다른 크기로 된 데이터 구조체 목록을 포함
  - 변경 저널 엔트리라 부름