디지털산업정보학회 논문지 제11권 제2호-2015년 6월
http://dx.doi.org/10.17662/ksdim.2015.11.2.073

Windows 파일시스템의 디렉토리에 대한 디지털 포렌식 분석* **

조 규 상***

A Digital Forensic Analysis for Directory in Windows File System

Cho Gyusang

〈Abstract〉

When we apply file commands on files in a directory, the directory as well as the file suffer changes in timestamps of MFT entry. Based on understanding of these changes, this work provides a digital forensic analysis on the timestamp changes of the directory influenced by execution of file commands. NTFS utilizes B-tree indexing structure for managing efficient storage of a huge number of files and fast lookups, which changes an index tree of the directory index when files are operated by commands. From a digital forensic point of view, we try to understand behaviors of the B-tree indexes and are looking for traces of files to collect information. But it is not easy to analyze the directory index entry when the file commands are executed. And researches on a digital forensic about NTFS directory and B-tree indexing are comparatively rare. Focusing on the fact, we present, in this paper, directory timestamp changes after executing file commands including a creation, a copy, a deletion etc are analyzed and a method for finding forensic evidences of a deletion of directory containing files. With some cases, i.e. examples of file copy and file deletion command, analyses on the problem of timestamp changes of the directory are given and the problem of finding evidences of a deletion of directory containging files are shown.

Key Words : Timestamp, Direcotory, Digital Forensics, NTFS, Windows, B-tree

Ⅰ. 서론

NTFS는 Windows NT부터 시작하여 Windows 8에 이르기까지 안정된 파일시스템으로 오랜 기간동안 사용되고 있다[1]. NTFS 파일시스템에 대한 동작원리와 세부적인 저장방법 등에 대해서 많은 부분이 잘 알려져있고 문서화도 잘 되어 있다[2,4,8]. 대표적인 자료로써 Wikipedia[1]에 NTFS의 역사적인 사실부터 기본적인 구조에 대한 설명이 충실히 기술되어 있고, Microsoft의 Technet[4]에 중요한 정보들이 잘 기술되어 있다. Carrier[2]가 발간한 "File System

* 이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임. (NRF-2013R1A1A2064426)
** 이 논문은 2014년도 동양대학교 학술연구비의 지원으로 수행되었음.
*** 동양대학교 컴퓨터정보전학과 교수

Forensics"에는 상세한 자료구조가 소개되어 있어 NTFS에 관한 정보에 대해서 많은 내용을 알 수 있게 되었다.

NTFS의 포렌식에 대한 몇몇 연구 사례들이 있다. Sameer H. Mahant와 B. B. Meshram등은 삭제된 파일을 복원하는 방법에 대한 연구를 수행하였다[9]. 많은 도구들에서 적용하고 있는 방법들을 이론적인 연구논문으로 펴낸 것에 의미를 둘 수 있다. Ewa Huebner등은 포렌식 분석을 하기 위한 목적으로 NTFS파일 시스템에서 데이터를 감추는 방법과 그 방법을 이용한 감지와 복원에 대한 연구를 수행하였다[10]. Christopher Lees는 웹브라우저를 사용한 내역의 포렌식을 방해하기 위해 NTFS의 USN(Update Sequence Number) 저널 파일에서 안티포렌식 도구를 사용한 흔적의 특이패턴을 찾아내어 분석할 수 있는 방법을 제안하였다[11]. 김[12] 등의 연구에서는 NTFS의 저널파일에 대한 구조를 역공학적 분석으로 수행하여 디지털 포렌식에 활용하기 위한 방법을 연구하였다. Cho의 연구[13]는 파일의 타임스탬프에서의 변화가 생성, 복사, 삭제, 갱신, 이동, 이름 변경 등의 파일연산에 따라 서로 상이한 변화패턴을 갖는다는 것을 밝혔고, 타임스탬프 변경 툴을 사용하여 타임스탬프를 조작한 사례를 탐지할 수 있는 포렌식 방법을 소개하였다. 또한 후속연구[14]에서 폴더에 대한 생성, 복사, 삭제, 갱신, 이동, 파일명 변경 등의 파일연산에 대해서도 타임스탬프 변화패턴의 분석을 수행하였고 그것을 평가함수에 적용하여 어떤 파일연산이 수행되었는지 알 수 있는 방법을 제안하였다.

컴퓨터 포렌식에 관한 블로그들에서 파일시스템에 관련된 내용을 다루고 있는 것을 발견할 수 있다. 그 중에서 William Ballenthin의 블로그[5]에서는 디렉토리 관련 파이썬 프로그램인 INDXParser.py를 소개하고 있다. 이것은 Excel CSV 포맷으로 디렉토리 안의 파일명, 파일크기, 타임스탬프 등으로 디렉토리 목록들을 파싱하고 슬랙 영역에 기록된 내용에서 포렌식 정보를 구하는 기능을 구현한 프로그램이다. Chad Tilbury가 작성한 SANS DFIR 블로그[6]에서는 NTFS 인덱스에 관한 내용을 정보를 구할 수 있다. 여기서는 할당된 인덱스 영역에 남아 있는 엔트리 들의 흔적으로 어떤 파일들에 사용되었는지를 판단해보기 위해서 Access Data 사의 FTK 툴, TSK의 icat 툴, 그리고 EnCase의 EnScript를 사용하여 인덱스에 기록된 내용을 추출한 간단한 사례를 보였다. William Ballenthin과 Jeff Hamm[7]는 NTFS의 인덱스 구조의 구성요소 데이터를 분리해내기 위한 도구의 사용방법, 인덱스 구조에 대한 설명 등의 정보를 작성하였다.

NTFS의 디렉토리와 관련된 정보들이 포렌식에서 다뤄지는 많은 사례에 비해서 관련 문헌들에서는 NTFS의 디렉토리의 파일시스템 안에서의 내부적인 운영 방법과 포렌식 방법에 대해서는 잘 기술되어 있지 않고 연구논문에서도 이런 부분에 대해서 다루고 있는 경우도 찾아보기 힘들다. 이 점에 주안하여 이 연구에서는 Windows의 파일시스템인 NTFS에서 디렉토리에 관련된 정보를 분석하여 디지털 포렌식에 활용하기 위한 방법을 제안한다. NTFS에서는 B-tree 구조를 파일시스템에 어떻게 적용되고 있는지 2장에서 살펴보기로 한다. 또한 3장 1절에서는 디렉토리 안에 들어 있는 파일 명령 실행 후 디렉토리의 MFT 엔트리내의 타임스탬프에 변화가 있는지 알아보려고 한다. 3장 2절에서는 디렉토리 전체가 삭제되었다는 것을 밝힐 수 있는 포렌식 분석 방법을 소개한다. 4장에서는 디렉토리 내에서 복사와 삭제 명령을 실행하였을 때와 디렉토리 전체가 삭제된 경우에 대해서 사례를 통해 제안된 방법으로 설명한다. 5장에서는 이 연구의 의미와 활용에 대한 논의로 결론을 맺는다.

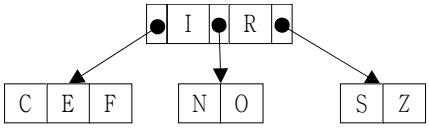
Ⅱ. 디렉토리 인덱스 구조

2.1 NTFS의 특징

NTFS는 디렉토리 안의 파일들을 관리하기 위하여 B-tree 방식의 자료구조를 사용하고 있다. 이 방식은 많은 파일을 다루는데 효율적이고 파일을 빠르게 찾을 수 있는 방식이다. Windows NT부터 기본 파일 시스템으로 장착되면서 Windows 8에 이르기까지 오랫동안 안정적인 파일시스템으로 자리잡고 있다. 시스템에 문제가 발생하거나 하드웨어적인 디스크에 손상이 있을 때 파일의 복구기능이 뛰어나며 암호화를 통한 보안 기능을 구현하였다. 권한이 있는 사용자만이 암호화를 할 수 있기 때문에 접근권한이 없는 외부 사용자로부터 파일을 보호할 수 있는 보안 기능을 갖추고 있다. 사용자 별로 특정 용량의 쿼터를 부여하고 관리할 수 있는 기능이 있고, 파일 압축, 분산 링크 트래킹, sparse 파일 기능, 다중 데이터 스트림 기능, Posix지원, NTFS 변경저널(change journal) 기능 등을 갖추고 있다. 무엇보다도 매우 큰 디스크 볼륨을 지원하는 기능이 장점이다. 이론상으로 최대 $(2^{64}-1)$개의 클러스터를 지원하고 파일의 크기는 최대 (16EB−1KB) 까지 지원하지만 실질적으로는 버전마다 지원하는 크기는 다양하다. 예를 들면 Windows 8에서는 최대 파일 크기는 (256TB−64KB)까지 지원된다[1,2].

2.2 NTFS에서의 B-tree 구현 방식

NTFS의 B-트리구조는 인덱스를 빠르게 검색하기 위한 구조이고 키 값의 갯수가 증가하여도 깊이 수준(depth level)을 증가시키는 것을 억제하기 위해 균형을 유지하는 균형트리(balanced tree) 방식이다[3]. 각 노드는 자식 노드에 대한 포인터와 키 값을 가지고 있고 키 값을 기준으로 왼쪽은 작은 자식 노드, 오른쪽은 큰 자식 노드를 가리키는 포인터를 가지고 있다. <그림 1>은 일반적인 B-tree의 간단한 예를 나타낸 것이다.



<그림 1> 일반적인 B-tree의 구조예

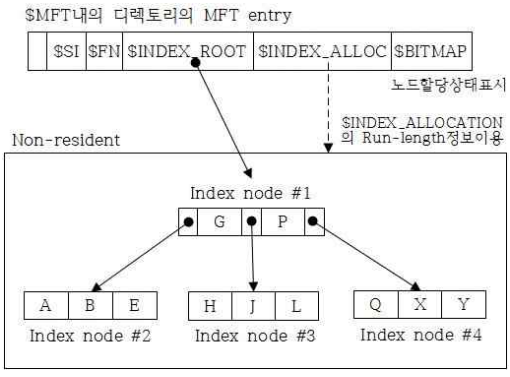
NTFS에서는 B-tree를 파일시스템에 적용하기 위해서 기본적인 구조와 운영방식은 따르고 있지만 이론과는 다른 구현을 위한 현실적인 방법이 몇 가지 적용되었다. 루트 노드 역할은 $INDEX_ROOT속성이 담당하고 있다. 하위노드에 대한 정보는 논레지던트(Non-resident)속성으로 $INDEX_ALLOCATION에 들어있는 Run-length 정보에 의해 유지된다. $BITMAP속성에는 할당된 노드들의 정보가 들어있다[2].

이론적으로 B-tree는 루트노드에는 반드시 최소 한 개의 키를 가지며 그것에 의해 두 개의 하위 노드를 연결하도록 구성되어야 하지만 NTFS에서는 루트 노드인 $INDEX_ROOT속성에 여러 개의 디렉토리 목록이 들어갈 수 있는 한 개의 노드형태로 변형하여 사용하고 있다. 이것은 MFT 엔트리의 레지던트 속성으로 사용되기 때문에 기록되는 목록의 수가 매우 적다. 이 공간의 크기는 최대 768바이트까지 가능하다. 파일명의 길이에 따라 다르지만 8.3형식의 파일명을 가지는 경우는 보통 5~6개 정도의 디렉토리 목록이 기록될 수 있다[15].

디렉토리의 목록(파일과 디렉토리들)이 늘어남에 따라 더 이상 저장할 공간(노드)이 없게 되면 새로운 노드를 만들어 공간을 확장한다. 이 때 4KB 크기의 노드안에는 많은 디렉토리 목록들이 저장되어 있고

항상 소팅되어 있어서 효율적이고 빠르게 검색이 가능하다. 파일명 소팅은 특수문자, 숫자, 영문자(대소문자 구분없음), 특수문자 순으로 소팅하게 된다. 즉, 아스키 코드순으로 소팅된다. 다만 대소문자의 구분을 하지 않는다. 파일의 길이가 8.3 포맷보다 긴 파일명은 8.3포맷의 짧은 파일명도 동시에 저장된다[4,8]. 동일 파일을 의미하는 두 긴 파일명과 짧은 파일명이 목록 내에 인접하게 저장하기 위해 대소문자를 구분하지 않는 방법을 적용한 것으로 생각된다.

각 노드들에 디렉토리 목록들이 증가하게 되면 여러 개의 노드들로 확장하게 된다. 참고문헌[15]에서 확장하는 방법에 대해서 기술하고 있다. 디렉토리안의 목록의 수가 많아지면 2층, 3층으로 트리의 높이를 키우게 된다. 이 때 중간 노드들에 들어 있는 디렉토리 목록들은 인덱스 키의 역할을 하며 디렉토리 목록이 하위 인덱스 노드나 리프 노드로 연결하기 위한 하위 노드의 번호가 저장되어 있다. 인덱스 노드들과 리프 노드들은 모두 4KB크기의 동일한 노드이다. 특이한 점은 목록들이 삭제되면 노드안에서 리스트들이 삭제되는데 이미 생성된 노드들은 목록이 들어 있지 않아도 노드를 삭제하지 않고 그대로 유지하게 된다. 그 안에 목록들의 흔적들이 남게 되며 이 정보들이 유용한 포렌식 정보가 된다.



<그림 2> NTFS의 B-tree 구성예

Ⅲ. 디렉토리 내에서 파일명령 실행 후 타임스탬프 변화에서 얻는 디지털 포렌식을 위한 정보

3.1 파일 명령 실행 후의 디렉토리에 기록된 타임스탬프에 대한 분석

이 절에서는 디렉토리 안에 들어 있는 파일에 대해서 생성, 삭제, 복사, 이동, 파일명 변경, 속성 변경 등의 파일 명령이 실행된 후에 MFT 엔트리에 들어 있는 디렉토리의 타임스탬프가 어떻게 변경되는지를 분석하고자 한다.

파일에 대한 명령을 실행하고 나면 생성시간, 수정시간, MFT수정시간, 접근시간 등의 4가지 타임스탬프에서 다양한 변화가 생긴다. 파일 명령에 대한 파일의 타임스탬프 변화에 대한 분석을 수행한 선행 연구[12,13]가 있지만 이 연구에서 이전의 연구 사례와는 달리 디렉토리 안에 들어있는 파일들에 대한 명령 실행한 후에 명령 실행 시간 값이 자신이 속한 디렉토리의 타임스탬프에 기록되는 것에 대한 분석을 수행하는 것이다.

3.1.1 파일 생성 명령

파일 생성 명령 실행 후에 디렉토리의 MFT엔트리 내의 두 타임스탬프 SIA($STANDARAD_INFORMATION attribute)와 FNA($FILNE_NAME attribute)는 변화가 생긴다. 디렉토리 안에서 새로운 어떤 파일 생성하는 경우에는 디렉토리의 MFT 엔트리에 있는 SIA의 수정시간(Modification time) t^M, MFT엔트리 변경시간(Entry modification time) t^E, 접근시간(Access time) t^A는 파일을 실행한 시간(Execution time) t_{exe}로 변경된다. 그러나 생성(Creation time) t^C는 변경되지 않는다. 또한 FNA의

<표 1> NTFS에서 구현된 B-tree의 요소들

항목	NTFS의 B-tree 특이 요소들
루트노드	$INDEX_ROOT속성 레지던트 속성
중간/리프노드	$INDEX_ALLOCATION속성의 Run-length정보로 표시되는 논레지던트 노드들
인덱스 레코드 크기	4KB
$INDEX_ROOT에 가능한 최대 목록 저장크기	768B
파일명 소팅 규칙	특수문자, 숫자, 영문자(대소문자 구분없음), 특수문자 순으로 소팅 -ASCII 코드 순
짧은 파일명/긴 파일명의 인덱스 목록	인덱스는 MFT엔트리와 마찬가지로 8.3포맷의 짧은 파일명과 255자까지 지원되는 긴 파일명으로 두 목록을 생성한다. 8.3포맷이하의 길이는 긴 파일명을 만들지 않는다[8].
인덱스 레코드 할당	인덱스 레코드 노드는 한번 생성되면 과련 파일이 모두 삭제되어도 인덱스 레코드는 빈 상태로 유지된다. 전체 디렉토리가 삭제되면 루트노드에서의 인덱스 링크 정보가 사라진다.

네가지 시간 FNA(t^C, t^M, t^E, t^A) 모두 변경되지 않는다. 식(1)은 이것을 기호로 나타낸 것이다.

$$SIA^{Dir}\ (t^C \rightarrow t^C,\ \ t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{Dir}(t^C \rightarrow t^C,\ \ t^M \rightarrow t^M,\ \ t^E \rightarrow \mathbf{t}^E,\ \ t^A \rightarrow t^A) \qquad (1)$$

$$SIA^{CreatedFile}\ (t^C \rightarrow t^M_{exe},\ t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{CreatedFile}(t^C \rightarrow t^C_{exe},\ t^M \rightarrow t^M_{exe},\ t^E \rightarrow \mathbf{t}^E_{exe},\ t^A \rightarrow t^A_{exe}) \qquad (2)$$

포렌식 분석 : 파일 생성 명령실행 후에는 당연히 디렉토리 안에 새로 생성된 파일이 존재하게 된다. 이 명령 실행 직후에 다른 명령이 실행된 적이 없다면 디렉토리의 타임스탬프는 생성된 파일내에 기록된 시간은 t_{exe}로 남아 있게 되므로 만일 디렉토리의 시간과 같은 값의 타임스탬프로 4개의 타임스탬프(생성, 수정, MFT변경, 접근)가 존재한다면 가장 최근에 파일 생성명령이 실행된 것으로 판단할 수 있다.

3.1.2 파일 삭제 명령

디렉토리 안에서 어떤 파일이 삭제되는 경우에는 디렉토리의 MFT 엔트리에 있는 SIA의 수정시간 t^M, MFT엔트리 변경시간 t^E, 접근시간 t^A는 각각 파일을 실행한 시간 t_{exe}로 변경된다. 그러나 생성시간 t^C는 변경되지 않는다. 또한 FNA의 네가지 시간 FNA(t^C, t^M, t^E, t^A) 모두 변경되지 않는다(식 3). 삭제된 파일의 타임스탬프는 원래 파일에 들어 있는 내용 그대로 유지된다. 파일을 삭제하면서 삭제되는 파일의 타임스탬프를 새로운 값으로 변경하지 않는다는 뜻이다(식 4).

$$SIA^{Dir}\ (t^C \rightarrow t^C,\ \ t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{Dir}(t^C \rightarrow t^C,\ \ t^M \rightarrow t^M,\ \ t^E \rightarrow \mathbf{t}^E,\ \ t^A \rightarrow t^A) \qquad (3)$$

$$SIA^{DeletedFile}\ (t^C \rightarrow t^M,\ t^M \rightarrow t^M,\ t^E \rightarrow t^E,\ t^A \rightarrow t^A)$$
$$FNA^{DeletedFile}(t^C \rightarrow t^C,\ t^M \rightarrow t^M,\ t^E \rightarrow \mathbf{t}^E,\ t^A \rightarrow t^A) \qquad (4)$$

포렌식 분석 : 파일 삭제 명령 후에는 디렉토리 안에 삭제된 파일이 존재하지 않는다. 디렉토리의 타임스탬프에는 식 (3)처럼 삭제가 실행된 시간 기록이 t_{exe}로 남는다. 이 시간과 동일한 타임스탬프를 가진 파일은 존재하지 않는다. 이런 상황이라면 디렉토리에 기록된 시간에 파일이 삭제되었다고 판단할 수 있다. 삭제된 파일은 MFT 엔트리에 남아 있는 정보를 이용하여 복구에 의하거나 디스크 툴을 이용하여 타임스탬프 정보를 얻을 수 있다(식 4). 또한 디렉토리 인덱스 노드에서 인덱스 목록의 흔적을 찾을 수 있다. 인덱스 노드는 항상 자동적으로 소팅을 하고 재

구성하여 기록하기 때문에 삭제된 파일이 들어 있는 위치에 따라 흔적이 지워질 수도 있고 그렇지 않을 수도 있다. 여러 인덱스 목록 중에서 마지막에 들어 있는 경우라면 언제나 흔적을 남기지만 중간에 들어 있는 경우라면 목록을 재구성하면서 지워진다.

3.1.3 파일 복사 명령

파일복사의 원본과 사본의 위치에 따라서 복사 명령 실행 후의 타임스탬프의 변화를 비교할 필요가 있다.타임스탬프의 관점에서는 같은 결과를 보이지만 복사의 원본과 사본이 놓인 곳의 차이점을 비교해서 포렌식 증거 수집의 대상을 적절히 정할 수 있다. 복사 명령에 의한 원본과 사본의 관계는 다음과 같은 5가지로 분류할 수 있다.

(1) 원본과 사본이 모두 해당 디렉토리 내에 존재하는 경우

(2) 원본은 다른 디렉토리에 있고 사본이 해당 디렉토리로 복사되는 경우

(3) 원본은 해당 디렉토리에 있고 사본은 다른 디렉토리로 복사되는 경우

(4) 원본이 들어 있지 않은 다른 외부 드라이브로 사본이 복사되는 경우

(5) 원본이 외부 드라이브에 있고 해당 디렉토리로 복사되는 경우

(1) 경우--(1)에서 디렉토리의 타임스탬프는 디렉토리의 MFT 엔트리에 있는 SIA의 수정시간 t^M, MFT엔트리 변경시간 t^E, 접근시간 t^A는 각각 파일을 실행한 시간 t_{exe}로 변경된다. 그러나 생성시간 t^C는 변경되지 않는다. 또한 FNA의 네가지 시간 FNA(t^C, t^M, t^E, t^A) 모두 변경되지 않는다(식 5).

(2) 경우-(2)는 경우 (1)과 동일한 결과를 보인다. 다만 원본이 사본과 다른 디렉토리에 들어 있다는 것이다.

(3) 경우--(3)은 원본이 들어 있는 해당 디렉토리의 타임스탬프와 복사 원본의 타임스탬프에는 전혀 변화가 없다. 사본이 생성되는 목적지의 디렉토리에 식 (5),(6)의 변화가 생긴다.

(4) 경우-(4)는 원본이 들어 있는 드라이브에 어떤 타임스탬프의 변화의 흔적은 없다. 파일이 복사된 외장 드라이브에 식 (5),(6)에 해당하는 결과가 기록된다.

(5) 경우-(5)는 복사가 실행된 후에 디렉토리의 타임스탬프도 식(5)와 식(6)로 타임스탬프가 변화된다.

$$SIA^{Dir}\ (t^C \rightarrow t^C,\quad t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{Dir}(t^C \rightarrow t^C,\quad t^M \rightarrow t^M,\quad t^E \rightarrow t^E,\quad t^A \rightarrow t^A) \qquad (5)$$

$$SIA^{CopiedFile}\ (t^C \rightarrow t^M_{exe},\ t^M \rightarrow t^M,\ \ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{CopiedFile}(t^C \rightarrow t^C_{exe},\ t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe}) \quad (6)$$

포렌식 분석 : 경우-(1)에서는 복사가 실행된 시간으로 디렉토리의 SIA의 t^M, t^E, t^A가 각각 복사된 시간 t_{exe}로 변경된다(식 5). 원본파일의 타임스탬프는 변경이 되지 않고 사본파일의 타임스탬프는 식(6)과 같이 변경이 된다. 이 시간과 동일한 타임스탬프를 가진 파일이 디렉토리에 존재한다. 그리고 SIA의 t^M이 동일한 두 파일이 한 디렉토리에 존재한다면 이것은 가장 최근에 이 디렉토리 안에서 파일의 복사가 t_{exe}시간에 일어 판단할 수 있다.

경우-(2)는 경우-(1)과 타임스탬프의 변화가 동일하다. 다만 원본이 들어 있는 곳과 사본이 만들어지는 곳이 다른 드라이브에 있다는 것이 차이점이다. 원본에 대한 검색을 디렉토리 인덱스 노드 내에서 할 수 없으므로 전체 디스크에 걸쳐서 원본 파일을 찾는다면 시간이 많이 소요될 것이다.

경우-(3)은 경우-(2)와는 역의 경우이다. 타임스탬프의 변화는 사본이 있는 곳에만 나타난다. 원본이 있는 곳의 타임스탬프의 변화는 없다.

경우-(4)는 원본이 들어 있는 드라이브에 어떤 흔적도 남지 않기 때문에 파일이 USB 또는 외장 디스크에 복사되어 유출되어도 알 수 없는 경우이다. 복사된 파일이 확보가 되지 않는다면 어떤 파일이 복사 유출되었는지 알 수 없기 때문에 다른 보조적인 기록을 통해서 포렌식을 위한 증거를 수집해야 한다.

경우-(5)는 경우-(2)의 유사 경우로 이해할 수 있다. 그 사본이 생성되는 디렉토리의 타임스탬프에 변화가 생기지만 원본이 있는 디렉토리에는 아무런 변화가 없다.

위의 5가지 복사의 사례를 정리하면 파일이 생성되는 곳의 디렉토리에는 식(5)와 (6)과 같은 타임스탬프의 변화가 생기지만 원본이 있는 곳에는 아무런 변화가 없다는 것이다. 다른 표현으로 설명하면 사본이 유입되는 디렉토리에는 타임스탬프의 변화가 있지만 유출되는 디렉토리에는 아무런 변화가 없다. 이런 기준으로 설명하면 복사에 의한 여러 가지 경우들을 단순화해서 설명할 수 있다.

3.1.4 파일 이동 명령

파일이동은 다음과 같이 4가지로 분류할 수 있다.

(1) 원본은 해당 디렉토리에 있고 다른 디렉토리로 이동되는 경우

(2) 원본이 다른 디렉토리에 있고 해당 디렉토리를 목적지로 하여 이동하는 경우

(3) 다른 드라이브로 이동하는 경우

(4) 다른 드라이브에서 해당 디렉토리로 이동의 경우

경우-(1)은 디렉토리의 타임스탬프는 디렉토리의 MFT 엔트리에 있는 SIA의 수정시간 t^M, MFT엔트리 변경시간 t^E, 접근시간 t^A는 각각 파일을 실행한 시간 t_{exe}로 변경되고 생성시간 t^C는 변경되지 않는다. 또한 FNA의 네가지 시간 FNA(t^C, t^M, t^E, t^A) 모두 변경되지 않는다(식 7). 이동한 파일은 원본이 놓여있는 곳에 기록되어 있던 SIA(t^C, t^M, t^E, t^A)가 이동한 파일의 FNA의 타임스탬프에 기록된다. 이것은 식(9)의 $FNA^{MovedFile}$ (t^C_{SI}, t^M_{SI}, t^E_{SI}, t^A_{SI})로 표현된다.

$$SIA^{Dir}\ (t^C \rightarrow t^C,\quad t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{Dir}(t^C \rightarrow t^C,\quad t^M \rightarrow t^M,\quad t^E \rightarrow t^E,\quad t^A \rightarrow t^A) \qquad (7)$$

$$SIA^{DestDir}\ (t^C \rightarrow t^C,\quad t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{DestDir}(t^C \rightarrow t^C,\quad t^M \rightarrow t^M,\quad t^E \rightarrow t^E,\quad t^A \rightarrow t^A) \qquad (8)$$

$$SIA^{MovedFile}\ (t^C \rightarrow t^C,\quad t^M \rightarrow t^M,\quad t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A)$$
$$FNA^{MovedFile}(t^C \rightarrow t^C_{SI},\ t^M \rightarrow t^M_{SI},\ t^E \rightarrow t^E_{SI},\quad t^A \rightarrow t^A_{SI}) \qquad (9)$$

경우-(2)에서 식(7)은 원본의 파일이 있던 디렉토리의 타임스탬프의 변화를 기록한 것이고 식(8)은 파일이 이동된 다른 디렉토리의 타임스탬프를 나타낸 것이다. 두 식에서 모두 SIA의 t^M, t^E, t^A가 각각 복사된 시간 t_{exe}로 변경된다. 이동된 파일의 타임스탬프는 식(9)와 같이 변경된다.

경우-(3)은 해당 두 디렉토리의 타임스탬프는 식(7)과 식(8)와 같이 t^M, t^E, t^A가 각각 복사된 시간 t_{exe}로 변경된다. 이것은 경우-(1)과 경우-(2)와 같은 결과이다. 다른 점은 이동된 파일의 타임스탬프의 변화가 식(10)과 같이 많이 다르게 나타난다는 점이다. 이동된 파일의 SIA의 t^C, t^M은 원본 파일의 타임스탬프 값이고 t^E와 t^A는 모두 파일 이동시간으로 설정된다. FNA의 시간도 모두 파일 이동시간으로 설정된다.

$SIA^{MovedFile}$ ($t^C \rightarrow t^C$, $t^M \rightarrow t^M$, $t^E \rightarrow t^E_{exe}$, $t^A \rightarrow t^A_{exe}$)(10)
$FNA^{MovedFile}$($t^C \rightarrow t^C_{exe}$, $t^M \rightarrow t^M_{exe}$, $t^E \rightarrow t^E_{exe}$, $t^A \rightarrow t^A_{exe}$)

경우-(4)는 외부의 드라이브에 원본 파일을 해당 드라이브로 복사하고 난 후에 원본 파일은 삭제되는 경우로 이해할 수 있다. 해당 두 디렉토리의 타임스탬프는 식(7)과 식(8)와 같이 t^M, t^E, t^A가 각각 복사된 시간 t_{exe}로 변경된다. 파일에서는 경우-(3)과 마찬가지로 식(10)의 결과로 이동된 파일의 타임스탬프를 얻을 수 있다.

포렌식 분석 : 경우-(1)과 경우-(2)는 출발지점과 목표지점이 서로 반대인 경우이다. 각각 원본의 디렉토리 시간과 이동된 파일들어 갈 디렉토리의 타임스탬프와 이동된 파일의 t^E가 세 가지 타임스탬프 t^M, t^E, t^A의 값이 모두 동일한 시간으로 변경된다는 점이 특이하다. 이런 경우는 한 디렉토리에서 다른 디렉토리로 이동되었다는 것을 알 수 있다.

한 드라이브 내에서 파일 이동의 경우는 실제로 파일이 이동하는 것이 아니라 MFT 엔트리에서 일부 파일 정보가 바뀌고 내용이 저장된 클러스터는 그대로 사용하고 있다. 실질적으로 파일이 이동한 것이 아니라 정보만 변경되었다는 의미다.

경우-(3)은 파일을 외장디스크로 복사한 후 삭제한 경우로 해석할 수 있다. 원본이 있던 디렉토리에는 유출된 시간이 디렉토리의 타임스탬프로 기록된다. 경우-(4)는 디렉토리 안에 이동된 파일은 복사의 경우와 유사하지만 타임스탬프의 결과가 다르다. 식 (6)에서 SIA는 $t^C \rightarrow t^M_{exe}$이지만 식(10)은 $t^C \rightarrow t^C$으로 다른 결과를 나타내서 복사와 이동을 서로 구분할 수 있다.

3.1.5 파일명 변경 명령

디렉토리 안에서 파일명이 변경되었을 때 디렉토리의 MFT 엔트리에 있는 SIA의 타임스탬프 중에서 생성시간 t^C를 제외하고 수정시간 t^M, MFT엔트리 변경시간 t^E, 접근시간 t^A는 각각 파일을 실행한 시간 t_{exe}로 변경된다. 또한 FNA의 네가지 시간 FNA(t^C, t^M, t^E, t^A) 모두 변경되지 않는다(식 11). 이 때 파일명이 변경된 파일은 SIA의 t^E가 파일명 변경시간으로 바뀐다. FNA의 4가지 타임스탬프는 파일명이 변경되기 전의 SIA의 타임스탬프 값으로 변경된다(식 12).

SIA^{Dir} ($t^C \rightarrow t^C$, $t^M \rightarrow t^M_{exe}$, $t^E \rightarrow t^E_{exe}$, $t^A \rightarrow t^A_{exe}$)
FNA^{Dir}($t^C \rightarrow t^C$, $t^M \rightarrow t^M$, $t^E \rightarrow t^E$, $t^A \rightarrow t^A$) (11)

$SIA^{F.NameChg}$ ($t^C \rightarrow t^M$, $t^M \rightarrow t^M$, $t^E \rightarrow t^E_{exe}$, $t^A \rightarrow t^A$)
$FNA^{F.NameChg}$($t^C \rightarrow t^C_{SI}$, $t^M \rightarrow t^M_{SI}$, $t^E \rightarrow t^E_{SI}$, $t^A \rightarrow t^A_{SI}$) (12)

포렌식 분석 : 파일명 변경 명령 후에는 파일의 타임스탬프에 특이한 변화가 생긴다. 변경되기 전의 SIA의 타임스탬프가 변경 후에 FNA로 복사되는 것이다. 이와 유사한 사례는 두 종류가 있다. 파일명의 변경과 파일이동의 경우-(2) "외부 디렉토리에서 해당 디렉토리로 이동"과 같은 형태의 타임스탬프 변화를 보인다. 두 명령이 같은 형태의 타임스탬프 변화를 보이는 이유는 현재의 파일이 들어 있는 위치 자체가 변하지 않고 MFT 엔트리 내에서의 값들만 변경되는 방식이 동일하기 때문이다. 파일시스템의 입장에서는 파일명을 변경하는 것이나 파일을 이동하는 것이나 MFT엔트리의 속성 정보만을 변경한다는 점에서 같은 동작이라는 의미로 해석된다.

3.1.6 파일내용의 갱신

디렉토리 안에 들어있는 파일의 내용이 수정되었을 때 디렉토리의 디렉토리의 MFT 엔트리에 있는 SIA의 타임스탬프 중에서 생성시간 t^C는 변경되지 않

고 수정시간 t^M, MFT엔트리 변경시간 t^E, 접근시간 t^A는 각각 파일을 실행한 시간 t_{exe}로 변경된다. 그리고 FNA의 네가지 시간 FNA(t^C, t^M, t^E, t^A)는 모두 변경되지 않는다(식 13). 이 때 내용이 수정된 파일은 SIA의 t^M과 t^E가 파일 수정후 저장한 시간으로 바뀐다. FNA의 4가지 타임스탬프 FNA(t^C, t^M, t^E, t^A)는 파일명이 변경되기 전의 SIA의 타임스탬프 값으로 변경된다(식 14).

$$SIA^{Dir}\ (t^C \rightarrow t^C,\ \ t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A_{exe})$$
$$FNA^{Dir}(t^C \rightarrow t^C,\ \ t^M \rightarrow t^M,\ \ t^E \rightarrow \mathbf{t}^E,\ \ t^A \rightarrow t^A) \qquad (13)$$

$$SIA^{Update}(t^C \rightarrow t^M,\ \ t^M \rightarrow t^M_{exe},\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A)$$
$$FNA^{Update}(t^C \rightarrow t^C,\ \ t^M \rightarrow t^M,\ \ t^E \rightarrow \mathbf{t}^E,\ \ t^A \rightarrow t^A) \qquad (14)$$

포렌식 분석 : 파일의 내용 수정 후에는 저장을 한 시간이 디렉토리의 SIA의 3개의 타임스탬프에 기록된다. 그것에 의하여 디렉토리 안에서 마지막으로 파일 갱신이 실행되었다는 것을 알 수 있다. 디렉토리의 타임스탬프만으로는 다른 명령과 구분되지 않기 때문에 같은 타임스탬프를 갖고 있는 파일을 찾아야 한다. 타임스탬프는 1/1,000초까지 세밀하게 저장된다. 파일내용 갱신의 경우는 MFT 엔트리 저장시간에서 미세한 지연이 있을 수 있다. 시분초 단위까지는 같은 타임스탬프를 가진 파일이 있고 파일의 타임스탬프 중 쓰기시간 t^M과 MFT 엔트리 수정시간 t^E가 같은 시간이라고 판단이 되면 그 시간에 파일내용을 갱신했다고 판단할 수 있다.

3.1.7 파일 속성의 변경

파일의 속성이 수정되었을 때는 이전의 다른 명령들과 다른 형태의 결과를 보인다. 디렉토리의 MFT 엔트리에 있는 SIA와 FNA의 각각 네가지 시간 FNA(t^C, t^M, t^E, t^A) 모두 변경되지 않는다(식 15). 이 때 파일의 타임스탬프는 오직 SIA의 t^E만이 속성 변경을 시도한 시간으로 기록된다(식 16).

$$SIA^{Dir}\ (t^C \rightarrow t^C,\ t^M \rightarrow t^M,\ t^E \rightarrow t^E,\ t^A \rightarrow t^A)$$
$$FNA^{Dir}(t^C \rightarrow t^C,\ t^M \rightarrow t^M,\ t^E \rightarrow \mathbf{t}^E,\ t^A \rightarrow t^A) \qquad (15)$$

$$SIA^{Chac}(t^C \rightarrow t^M,\ t^M \rightarrow t^M,\ t^E \rightarrow t^E_{exe},\ t^A \rightarrow t^A)$$
$$FNA^{Chac}(t^C \rightarrow t^C,\ t^M \rightarrow t^M,\ t^E \rightarrow \mathbf{t}^E,\ \ t^A \rightarrow t^A) \qquad (16)$$

포렌식 분석 : 식 (15)의 의미는 파일의 속성을 변경하는 것으로는 디렉토리의 타임스탬프가 변경되지 않는다는 의미이다. 다른 명령들은 항상 명령 후에 디렉토리의 타임스탬프가 변경되지만 이것은 변경되지 않기 때문에 디렉토리 내에서 마지막으로 실행된 연산을 추정하려면 다른 파일들의 타임스탬프를 종합적으로 분석해야 알 수 있게 된다. 파일의 속성을 변경했다면 타임스탬프 중에서 디렉토리의 타임스탬프보다 SIA의 t^E만이 한 개만이 유일하게 앞선 시간을 갖는다면 이것은 파일의 속성이 변경되었다고 판단할 수 있다.

3.1.8 하위 디렉토리의 타임스탬프 변경

하위 디렉토리에 대한 명령어는 일반 파일에 대한 명령과 동일하게 취급할 수 있다. 디렉토리의 생성, 복사, 삭제, 이동, 디렉토리명 변경, 속성 변경 등과 하위 디렉토리 내의 목록 변동 등은 파일에 대한 명령을 실행하는 것과 동일한 타임스탬프 결과를 만들어 낸다. NTFS에서는 MFT 엔트리에 기록된 파일에 대한 정보나 MFT 엔트리에 기록된 디렉토리에 대한 정보를 동일한 파일로 간주하기 때문이다.

이 장에서 다룬 디렉토리 내에 들어 있는 파일에 대한 명령을 실행하고 난 후에 디렉토리의 타임스탬

프의 변화가 생기는 것은 디렉토리 자체에 대해서 3.1.8에서 다룬 내용과는 다른 문제라는 것을 잘 이해해야 한다. 차이점은 디렉토리 안에 들어 있는 파일에 대한 명령이 실행되고 나면 디렉토리의 타임스탬프가 명령 실행 시간으로 변경되는 것과 디렉토리 자체에 대한 명령(생성, 삭제, 복사 등)을 실행하고 나면 파일에서 명령한 것과 같은 종류의 변화가 일어난다는 점에서 큰 차이가 있다.

3.2 디렉토리 삭제 시각 포착

디렉토리를 삭제하는 경우에는 삭제 명령 실행시간이삭제된 디렉토리의 $SI의 타임스탬프의 쓰기, MFT 수정, 접근 시간등의 3가지 타임스탬프의 값이 기록된다. 또한 디렉토리를 포함하고 있는 상위 디렉토리의 쓰기, MFT 수정, 접근 등의 3가지 타임스탬프의 값이 삭제된 시간으로 변경된다. 이 점은 포렌식 정보로써 중요한 의미를 부여할 수 있다.

디렉토리의 목록이 레지던트 속성으로 구성된 경우에 디렉토리를 삭제하고 나면 $INDEX_ROOT(0x90) 속성의 레지던트 데이터로 구성된 인덱스 엔트리가 삭제된다. 논레지던트로 구성된 $INDEX_ALLOCATION(0xA0)속성인 경우는 인덱스 레코드에 대한 할당정보는 그대로 유지되는 특징이 있다. $BITMAP속성은 인덱스 레코드를 할당한 정보를 갖고 있는데 이 항목은 존재하지만 해당되는 비트맵 정보를 0으로 수정해 버려서 할당하고 있는 페이지가 없음을 나타낸 정보를 갖게된다. 이 정보들을 이용하여 디렉토리 전체가 삭제된 경우를 찾을 수 있다.

논레지던트의 경우도 레지던트의 경우와 마찬가지로 인덱스 엔트리의 구성 요소에 대한 내용은 삭제하게 된다. 디렉토리가 삭제되기 전에 파일들이 먼저 삭제된다는 것을 알 수 있다.

Ⅳ. 디렉토리에서 디지털 포렌식을 위한 사례 분석

4.1 작업환경

디렉토리 안의 파일명령을 실행하기 위한 외장 usb 디스크는 다음과 같은 환경으로 구성되어 있다.

OS : Windows 7 Ultimate K Service Pack 1
디스크 포맷 : NTFS v3.1
저장매체 : SSD 외장 usb 드라이브
저장공간 : 1TB
디스크 할당 클러스터 크기 : 4,096 바이트
디렉토리 명 : CheckDir
사용된 파일들 : file-000.txt~file-006.txt

4.2 파일 명령어에 실행에 의한 디렉토리의 타임스탬프의 변화를 근거로한 포렌식 분석

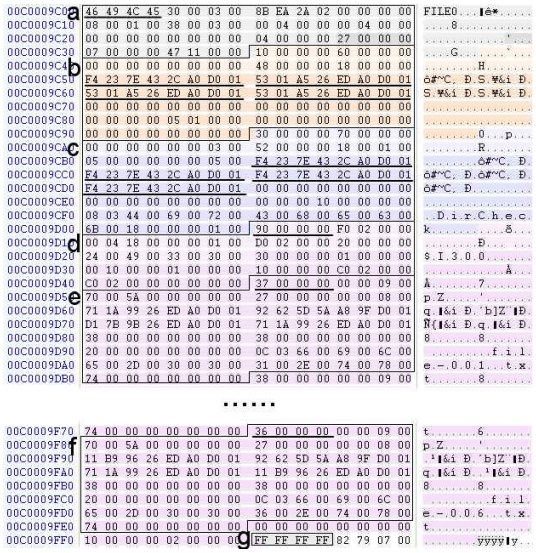
4.2.1 복사 명령의 사례 분석

아래의 <그림 3>은 DirCheck 디렉토리에 들어 있는 파일들의 인덱스 목록을 나타내고 있다. “a”는 파일헤더이고, “b”는 $STANDARD_INFORMATION(0X10) 속성이고 “c”는 $FILE_NAME(0X30)속성을 나타낸다. d”는 레지던트 속성(0x90)으로 저장되어 있는 $INDEX_ROOT속성이다. 디렉토리 안에 file-001.txt~file-006.txt까의 6개의 파일에 대한 인덱스 리스트를 나타낸다. 표 2의 “복사명령전” 타임스탬프는 5개의 파일이 DirCheck디렉토리로 복사된 후의 타임스탬프이다. “e”는 file-001.txt“(Ref. ID=0x37), “f”는 file-006.txt“(Ref. ID=0x36)을 나타내고, “g“는 MFT엔트리의 끝을 나타내는 표식이다.

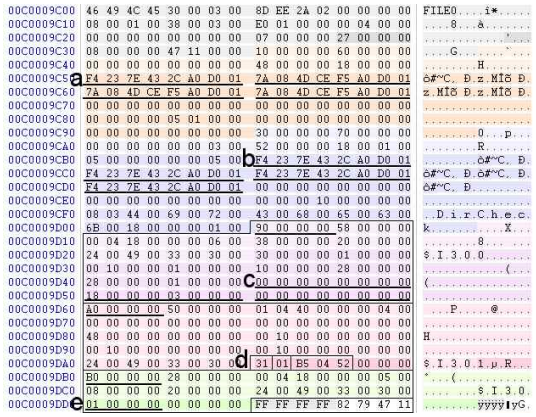
복사명령 “copy file-001.txt file-000.txt”를 명령프롬프트에서 실행하여 파일 file-001.txt를 file-000.txt로 복사한 경우이다. 이 경우는 3.1.3절의 경우-1에 해당하는 “원본과 사본이 모두 디렉토리 안에 있는 경우”이다.

표 2의 DirCheck의 2015-06-07 06:42:40은 file-006.txt파일의 복사시간이 기록된 것이다. 복사명령을 실행한 뒤에 DirCheck 디렉토리의 타임스탬프가 변경되었다. M, E, A의 타임스탬프가 모두 2015-06-07 07:44:38으로 변경되었다(<그림 4>의 “a”). 이것은 복사명령이 실행된 시간을 의미한다. <그림 5>는 DirCheck디렉토리의 MFT엔트리의 스크린샷이다. <그림 3>에서는 모든 디렉토리 목록들이 레지던트 속성에 기록되어있지만 복사 명령 후에는 레지던트 공간에 디렉토리 목록을 저장할 공간이 부족하기 때문에 외부의 인덱스 목록들은 <그림 5>와 같이 논레지던트($INDEX_ALLOCATION속성)로 확장된다. <그림 4>의 “b”는 $FILE_NAME 속성을 나타내고, “c”의 세 번째 8바이트“00 00 00 00 00 00 00 00”가 논레지던트 인덱스 레코드의 VCN을 나타낸다. 이것의 실제 할당된 블록은 “d”에 표시된다. 0x31에서 할당된 인덱스 레코드의 Run-length를 의미하는데 3은 뒤쪽에 기록된 주소가 3바이트 크기이고 의미고 1은 연속된 블록의 수를 표시하는 숫자가 1바이트 사용된다는 의미로 이 값이 1이므로 한 개만 할당되었다는 의미이다. 이것은 0x5204B5KB의 주소에는 즉 0x5204B5000(<그림 5>의 시작주소)에 $INDEX_ALLOCATION의 할당된 주소가 시작되고 연속 된 개수는 1개라는 의미로 해석된다.

데이터들(타임스탬프와 주소들)은 리틀엔디언(Little endian)방식으로 저장되고 있기 때문에 <그림 4>의 “d”의 주소 “B5 04 52”는 역으로 “52 04 B5”로 읽어야 한다. 또한 “b”의 첫 번째 타임스탬프의 경우 “F4 23 7E 43 2C A0 D0 01”은 “01 D0 A0 2C 43 7E 23 F4”로 읽어야 한다. 이것은 연월일-시분초와



<그림 3> DirCheck 디렉토리의 타임스탬프와 레지던트 인덱스 목록



<그림 4> DirCheck 디렉토리의 MFT엔트리 내용–타임스탬프와 $INDEX_ALLOCATION으로 확장된 인덱스 목록

1/1000ms까지 표시되는 정밀한 시간값이다.

복사 명령 실행 후에 얻은 포렌식 정보 : 파일을 DirCheck 디렉토리의 MFT 엔트리의 SIA속성(<그림 4>의 “b”의 3개(M, E, A)의 타임스탬프)에 기록된 타임스탬프와 동일한 타임스탬프를 갖는 파일이 명령

<표 2> DirCheck 디렉토리: 타임스탬프-복사명령 전후

	SIA	FNA
복사 명령전	C : 2015-06-06 07:41:56 M : 2015-06-07 06:42:40 E : 2015-06-07 06:42:40 A : 2015-06-07 06:42:40	C : 2015-06-06 07:41:56 M : 2015-06-06 07:41:56 E : 2015-06-06 07:41:56 A : 2015-06-06 07:41:56
복사 명령후	C : 2015-06-06 07:41:56 M : 2015-06-07 07:44:38 E : 2015-06-07 07:44:38 A : 2015-06-07 07:44:38	C : 2015-06-06 07:41:56 M : 2015-06-06 07:41:56 E : 2015-06-06 07:41:56 A : 2015-06-06 07:41:56

이 실행된 파일을 나타낸다. <그림 5>의 "b"의 3개(C, E, A)가 같은 타임스탬프를 나타낸다. 이 중에서 E는 코드값으로는 달리 보이지만 파일시스템에 기록되는 동안 수 ms정도의 지연시간이 포함되어있는 시간이다. 타임스탬프가 내부에 있으면 그 파일이 이 디렉토리 안에서 가장 마지막으로 명령이 실행된 파일이라는 것을 알 수 있다(식 5).

복사명령이 실행되고 난 후에 새로운 복사된 파일의 타임스탬프는 식(6)과 같은 형태로 기록된다. <그림 5>의 "b"가 새로 복사된 file-000.txt의 인덱스 목록이고 "c"가 원본인 file-001.txt이다. 둘의 타임스탬프의 연관성은 두 번째 타임스탬에서 찾을 수 있다. 이것은 수정시간(2015-06-05 15:57:41)이며 원본과 사본을 구분하는 방법은 MFT수정시간의 전후관계를 살피면 알 수 있다. file-000.txt는 (2015-06-07 07:44:38)이며 file-001.txt는 (2015-06-07 06:42:40)이므로 file-001.txt가 원본이다. "b"의 m1과 "c"의 m2는 파일의 고유번호(Ref. ID)이다.

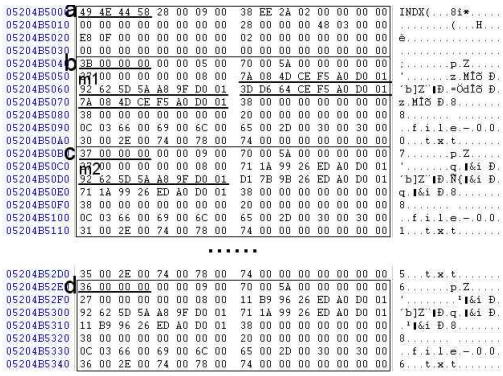
복사명령 "copy file-001.txt h:\"로 DirCheck 디렉토리 안의 파일을 다른 드라이브로 파일을 복사하는 경우이다. 3.1.3절의 경우-4에 해당한다. 이 경우는 DirCheck 디렉토리 안에 어떤 복사의 흔적도 남기지 않는다는데 문제가 있다. 많은 파일시스템 포렌식에서 이미 다룬 문제이지만 현재까지는 Windows 7의 NTFS에서 기본 설정으로는 복사여부를 파악할 수 없다는 한계가 있다. 참고문헌 [16]에서의 연구는 파일시스템의 접근시간이 기록가능한 경우에는 대량의 복사가 있었다면 집중적으로 같은 접근 시간을 나타내고 그렇지 않은 경우라면 분산적인 접근시간의 형태를 나타내는 특징으로 외부 복사 유출을 알 수 있는 방법을 제안하였지만 Windows 7의 NTFS의 디폴트 설정은 접근시간의 변경이 기록되지 않지만 *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\NtfsDisableLastAccessUpdate*를 1로 변경하면 가능하다.

4.2.2 삭제 명령의 사례 분석

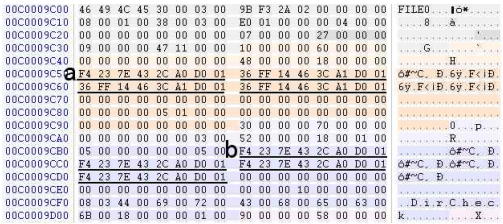
파일을 삭제한 후에 디렉토리의 타임스템프에 기록된 시간으로 삭제 명령이 실행된 시각을 알아내고 어떤 명령이 실행되었는지 분석하는 방법을 고려한다. 원본 파일이 삭제된 것을 복구하는 것 이외에도 삭제의 흔적을 디렉토리의 인덱스 엔트리에서 찾기로 한다.

삭제명령 "del file-000.txt"으로 파일을 삭제한 후에 DirCheck의 MFT에 기록된 타임스탬프는 표 3와 <그림 6>의 "a"- SIA속성, "b"-FNA속성 에 나타낸 바와 같다. 2015-06-07 16:09:03에 삭제 명령을 실행하였고 이 시간이 타임스탬프 "a"의 3가지 타임스탬프 W, E, A에 기록된 것이다.

<그림 7>은 인덱스 레코드의 디렉토리 목록을 나타내고 있다. 삭제된 파일 file-000.txt는 이 목록에서 찾아볼 수 없다. 언제나 목록에 변화가 생기면 자동적으로 알파벳순으로 소팅하여 저장하기 때문이다. <그림 7>의 "a"는 file-001.txt의 목록이 file-000.txt의 기록이 쓰여 있던 부분에 덮어쓰기 한 결과를 나타낸다. "b"는 이 디렉토리의 마지막 요소인 file-006.txt의 목록의 내용이다. 이 목록은 파일삭제 명령전에는 "c" 부분에 들어 있던 것이다. "c"부분은 유효하지 않은 데이터



<그림 5> DirCheck의 인덱스 목록 :
디렉토리의 타임스탬프와 레지던스 인덱스 목록



<그림 6> DirCheck 디렉토리의 타임스탬프-파일 삭제 후

<표 3> DirCheck 디렉토리: 타임스탬프-삭제명령 전후

	SIA	FNA
삭제 명령전	C : 2015-06-06 07:41:56 M : 2015-06-07 07:44:38 E : 2015-06-07 07:44:38 A : 2015-06-07 07:44:38	C : 2015-06-06 07:41:56 M : 2015-06-06 07:41:56 E : 2015-06-06 07:41:56 A : 2015-06-06 07:41:56
삭제 명령후	C : 2015-06-06 07:41:56 M : 2015-06-07 16:09:03 E : 2015-06-07 16:09:03 A : 2015-06-07 16:09:03	C : 2015-06-06 07:41:56 M : 2015-06-06 07:41:56 E : 2015-06-06 07:41:56 A : 2015-06-06 07:41:56

영역인 슬랙영역이다. 이 안에 기록된 정보는 지워지지 않고 남은 인덱스의 목록에 관한 정보이다.

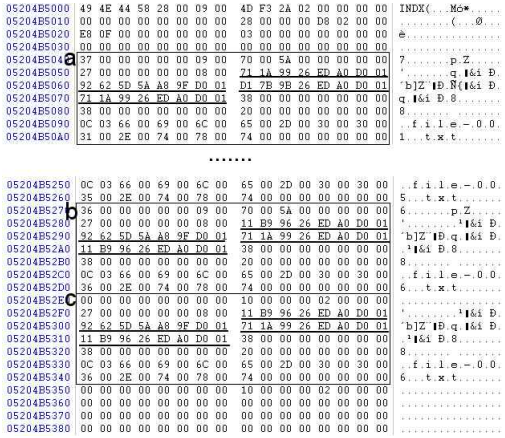
삭제명령 “del file-006.txt”으로 디렉토리의 마지막인 file-006.txt를 삭제한다. 파일을 삭제한 후의 슬랙영역에서의 흔적에 대해서 살펴보려고 한다.

<그림 8>의 “a” 부분은 인덱스 엔트리 헤더이다. 인덱스 헤더 16바이트 이후에는 FNA속성이 저장된다. 이 부분은 파일 file-005.txt의 인덱스 목록을 나타낸 것이다. 헤더의 첫 번째 8바이트는 MFT 파일 레퍼런스 번호(0x35, 53)를 나타낸다. 그 뒤의 2바이트는 이 엔트리 전체의 길이가 0x70(112)바이트이다는 의미이다. 세 번째 요소 2바이트는 FNA속서의 길이를 나타낸다. 0x5A(90)는 전체 엔트리 리스트 중에 FNA의 속성의 길이를 나타낸다. 맨 끝의 4바이트는 플래그 값을 나타낸다. 플로그 값이 0x00이면 자식노드가 존재하지 않는다는 의미, 0x01이면 자식노드가 존재한다는 의미이다. 0x02이면 리스트의 끝을 알리는 표식으로 사용된다. 이 경우 0x00이므로 자식노드가 없다는 뜻이다.

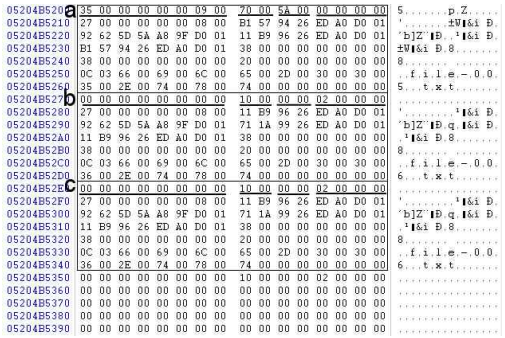
슬랙 영역에 남은 두 정보는 인덱스 헤더의 내용에서만 차이가 있고 나머지 영역은 모두 같은 값을 보인다.

“b“와 “c“의 첫줄에서 첫 8바이트는 모두 0으로 기록되어 있다. MFT 파일 레퍼런스 번호를 지운 것이다. 두번째 2바이트가 0x10으로 기록되어 있는데 항목의 길이의 원래 값은 0x70이지만 흔적을 남기면서 0x10으로 변경이 된 것이다. “b“와 “c“의 플래그 값은 모두 0x20으로 바뀌었다. 이 값은 인덱스 목록의 끝이라는 의미이다.

삭제 명령 실행 후에 얻은 포렌식 정보 : 삭제의 경우는 디렉토리 안의 파일들의 타임스탬프와 디렉토리 DirCheck의 MFT에 기록된 타임스탬프가 일치하는 파일을 찾을 수 없다. 왜냐하면 삭제 명령의 대상이 지워져서 없어졌기 때문이다. 이 삭제된 파일을 복구를 하여도 작업시간이 기록되지 않기 때문에 타임스탬프만으로 디렉토리에 기록된 시간에 해당하는 명령을 추론하기는 어렵다. 그러나 모든 명령들은 파일에 흔적을 남기지만 파일 삭제는 파일에는 타임스



<그림 7> DirCheck, test-000.txt 파일 삭제후의 인덱스 엔트리에 남은 흔적



<그림 8> DirCheck, test-006.txt 파일 삭제후의 인덱스 엔트리에 남은 흔적

탬프를 기록하지 않고 오직 디렉토리의 MFT 타임스탬프만 변경한다. 디렉토리의 MFT 타임스탬프와 동일한 시간을 갖는 디렉토리 내의 파일이 존재하지 않으면 디렉토리에 기록된 타임스탬프 시간에 파일삭제가 일어났다고 판단할 수 있다. 이 경우에는 2015-06-07 16:09:03에 파일 삭제명령이 실행되었다고 판단할 수 있다.

삭제명령에 의하여 포렌식 흔적이 남는 경우는 소팅된 인덱스 목록의 맨 끝에 있는 경우이다. 목록의 중간에 있는 항목의 경우는 덮어쓰기에 의하여 흔적을 남기지 않고 사라지게 된다.

4.3 디렉토리 삭제의 증거포착 사례 분석

이 절에서는 파일이 들어 있는 디렉토리 자체를 삭제하였을 때 삭제시간의 추정을 위한 사례를 소개한다. 4.2.2절에서 기술한 바와 같이 일반적으로 파일이 삭제되면 파일자체의 타임스탬프에는 기록을 남기지 않는다. 디렉토리도 일종의 파일이므로 같은 맥락으로 해석이 된다. 하지만 디렉토리가 하위 파일을 갖고 있는 경우는 결과가 다르게 나타난다.

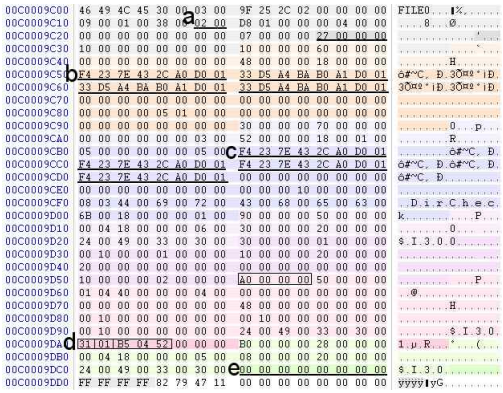
<표 4> DirCheck 디렉토리: 타임스탬프-디렉토리 전체 삭제 전후

	SIA	FNA
삭제 명령전	C : 2015-06-06 07:41:56 M : 2015-06-07 17:48:25 E : 2015-06-07 17:48:25 A : 2015-06-07 17:48:25	C : 2015-06-06 07:41:56 M : 2015-06-06 07:41:56 E : 2015-06-06 07:41:56 A : 2015-06-06 07:41:56
삭제 명령후	C : 2015-06-06 07:41:56 M : 2015-06-08 06:02:41 E : 2015-06-08 06:02:41 A : 2015-06-08 06:02:41	C : 2015-06-06 07:41:56 M : 2015-06-06 07:41:56 E : 2015-06-06 07:41:56 A : 2015-06-06 07:41:56

디렉토리 전체 삭제 명령이 실행된 시각은 2015-06-08 06:02:41이다. 이 시각이 SIA의 M, E, A에 기록된다. 이 시각은 엄밀히 말하면 디렉토리 안에 들어 있던 파일 중에서 제일 마지막으로 파일이 삭제

된 시각을 의미한다. 마지막 파일 file-005.txt가 삭제되었다. 이 사실은 <그림 10>에서 INDX파일의 근거로 찾을 수 있다.

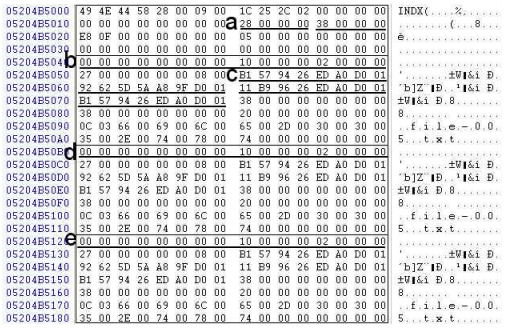
<그림 9>는 전체가 삭제된 CheckDir 디렉토리의 MFT엔트리를 나타낸 것이다. “a”는 삭제된 디렉토리라는 의미이다. 사용중인 디렉토리인 경우는 0x03이 기록되어 있다. “b”는 SIA의 타임스탬프이다. 마지막 파일이 삭제된 시각 정보를 갖고 있다(표 4). “c”는 FNA 속성이다. 이 부분은 변화가 없다. “d”는 $INDEX_ ALLOCATION속성(0xA0) 중에서 디렉토리 인덱스가 할당된 곳을 알 수 있는 정보가 들어 있다. 할당된 클러스터의 번호는 0x5204B5000이다. 이 곳에 가면 인덱스 레코드가 들어 있다. “e”는 $BTIMAT속성을 나타낸다. 밑 줄 부분에 기록되었던 비트값들은 할당된 클러스트의 사용여부를 알 수 있는 정보이다. 이 값들이 0으로 되어 있으면 할당된 클러스터가 없다는 뜻이다. 파일이 삭제되면서 인덱스 레코드들 중에서 파일을 저장하고 있는 인덱스 레코드는 없다는 의미이다. 인덱스 레코드는 할당될 때는 동적으로 할당되지만 파일이 삭제될 때는 해당 영역이 해제되지 않는다는 특징이 있다(표 1의 인덱스레코드 할당).



<그림 9> DirCheck의 MFT엔트리-디렉토리 전체 삭제

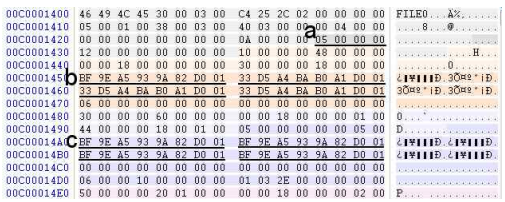
<그림 10>의 “a”의 0x28은 현재 위치에서 인덱스 엔트리가 시작되는 지점까지의 오프셋을 의미한다. 실제로 인덱스 엔트리는 0x40(0x18+0x28)위치에서 시작된다. 0x38은 인덱스 엔트리의 끝까지의 오프셋을 의미한다. 그러므로 실제 인덱스 엔트리에 할당된 것은 0x10(16)바이트 뿐이라는 의미이다. “b”의 16바이트가 삭제된 인덱스 엔트리를 나타내는 정보이다. 여기서 0x10은 할당된 크기를 나타내고 0x02는 삭제된 인덱스 엔트리를 나타내는 플래그 값이다. “c”는 삭제가 실행된 시간 정보를 담고 있다. “d”와 “e”는 이미 이전에 삭제된 인덱스 엔트리를 의미한다. 엔트리는 항상 소팅이 되어 있어서 리스트 중에서 file-005.txt가 맨 끝 파일이었고 인덱스 리스트에 흔적이 여러번 남아 있는 것은 전체 목록중에 가장 끝에 있고 가장 나중에 삭제되었다는 의미이다.

<그림 11>의 “a”의 0x05는 루트 디렉토리의 MFT Ref. 값이다. “b”의 타임스탬프에서 M, E, A의 3개의 타임스탬프가 <그림 9>의 것과 동일하다. 즉, DirCheck에 들어 있는 파일들이 삭제되면서 DirCheck의 3개의 타임스탬프를 변경하였고, DirCheck가 삭제되면서 그 상위 디렉토리인 루트의 M, E, A의 타임스탬프가 변경되었다는 것을 알 수 있다.



<그림 10> DirCheck의 인덱스엔트리-디렉토리 전체 삭제 후

전체 디렉토리의 삭제를 포착하기 위한 포렌식 정보 : 전체 디렉토리의 삭제는 디렉토리 정보 자체와 그 안에 들어 있는 파일들까지 같이 삭제되는 경우이다. 이 사실을 알기 위해서는 우선 DirCheck디렉토리의 MFT엔트리 정보를 찾는 것부터 시작한다. 0x27(39)가 MFT 엔트리 번호이다. 그곳에 가면 "b"위치에 SNA속성의 타임스탬프가 들어있다. 이 중에서 3개의 타임스탬프 M, E, A에 기록된 2015-06-08 06:02:41가 $INDEX_ALLOCATION속성(0xA0)을 찾을 수 있는데 여기에 기록된 정보는 디렉토리가 삭제되어도 그대로 남아있다. 주소가 0x5204B5000인 클러스터에 가면 디렉토리의 목록이 들어 있다. 이 안에 들어 있는 목록 중에서 제일 처음의 것이 <그림 10>의 "b"처럼 첫 16바이트가 인덱스의 끝을 알리는 정보로 채워져있으면 이 인덱스 레코드는 비어 있다는 것을 의미한다. 결론적으로 다음과 같은 절차로 디렉토리 전체가 삭제된 시간을 찾을 수 있다.



<그림 11> DirCheck의 상위디렉토리(루트)의 타임스탬프-디렉토리 전체 삭제 후

<그림 11> DirCheck의 상위디렉토리(루트)의 타임스탬프-디렉토리 전체 삭제 후

과정 1: <그림 9>의 "a"부분에서 0x02를 찾는다.

과정 2: <그림 10>의 "a" 부분에서 0x28, 0x38이 기록되어 있는 것을 찾는다.

과정 3: <그림 10>의 "b"와 같이 목록의 끝을 알리는 데이터가 기록을 확인한다.

과정 4: <그림 9>의 "b"의 타임스탬프 중 M, E, A에 기록된 시간 2015-06-08 06:02:41이 디렉토리 전체가 삭제된 시간이다.

Ⅴ. 결론

이 연구에서는 윈도우즈의 NTFS 파일시스템에서 디렉토리에 관련된 디지털 포렌식 정보를 얻기 위한 방법을 소개하였다. 디렉토리의 구조[1,2,3,4]에 대한 정보는 잘 문서화되어 있지만 B-tree방식으로 구현된 디렉토리의 목록을 유지하는 방법에 대해서는 자료구조를 포함한 기본적인 정보만 알려져 있다. 이 연구에서는 실질적으로 적용이 되고 있는 이 부분에 대해서 디지털 포렌식의 관점에서 접근하였다.

이 연구의 3.1절에서 디렉토리의 MFT 엔트리에 기록된 $STANDARD_INFORMATION의 타임스탬프는 디렉토리 안에 들어있는 파일들이 어떤 명령에 따라 변경되면 작업한 시간이 SIA의 타임스탬프 중 생성시간을 제외한 쓰기, MFT 수정, 접근시간에 모두 같은 시간을 기록한다는 사실을 밝혔다. 이 정보는 파일의 생성,삭제, 복사, 이동, 파일명 변경, 파일내용 갱신, 파일 속성변경 등의 명령에 대하여 디렉토리 안에서 언제, 어떤 명령이 실행되었는지 추정할 수 있는 근거로 사용된다. 4.2절에서 복사, 삭제에 대한 사례분석을 수행하였고 3.2절과 4.3절에서 디렉토리 전체가 삭제되었음을 판단하기 위한 기술적인 근거를 제시하였고 그것을 사례로 분석하였다.

파일의 타임스탬프에 대한 분석의 연구[9-14]는 많이 수행되었지만 디렉토리를 중심으로 다룬 연구는 찾기 힘들다. 이 연구에서 디렉토리에 대한 실제 사례를 중심으로 디지털 포렌식 관점에서 분석을 제공한 것에 의미를 부여한다. 디렉토리 전체가 삭제된 시각을 알아낼 수 있는 방법을 제시한 것도 디지털 포렌식연구에 도움이 된다고 자평한다. 또한, B-tree가 이론에서 실제로 구현되는 과정에 대해 기술한 부

분도 이 연구가 기여하고 있는 부분이라고 생각한다.

이 연구를 바탕으로 디렉토리와 파일들간의 정보를 상호 연계할 수 있는 포렌식 도구를 제작할 필요성이 있다. 기존의 포렌식 도구들에서 아직 구현이 되지 않은 기능이므로 개발한 후에는 디렉토리에 관한 디지털 포렌식 분석을 좀 더 논리적이고 조직적이며 자동화된 방식으로 수행할 수 있는 방법과 도구를 개발하는 것은 추후과제로 의미가 있다.

참고문헌

[1] Wikipedia. org, "NTFS - Features - Scalability," http://en.wikipedia.org/wiki/NTFS#Features

[2] B. Carrier, File System Forensic Analysis, Addison-Wesley, 2005, pp. 273-396.

[3] Wikipedia, "B-tree," http://en.wikipedia.org/wiki/B-tree.

[4] Microsoft TechNet, "How NTFS Works," https://technet.microsoft.com/en-us/library/cc781134(v=ws. 10).aspx.

[5] William Ballenthin, "NTFS INDX Attribute Parsing," http://www.williballenthin.com/forensics/indx/index. html.

[6] Chad Tilbury, "NTFS $I30 Index Attributes: Evidence of Deleted and Overwritten Files," SANS Digital Forensics and Incident Response Blog, http://digital-forensics. sans.org.

[7] William Ballenthin and Jeff Hamm, "Incident Response with NTFS INDX Buffers - Parts 1, 2, 3 and 4," https://www.mandiant.com/blog/author/willi-ballenthin/

[8] Microsoft MSDN, "Naming Files, Paths, and Namespace-Short vs. Long Names," http://msdn.microsoft.com.

[9] Sameer H. Mahant and B. B. Meshram, "NTFS Deleted Files Recovery: Forensics View," IRACST(- International Journal of Computer Science and Information Technology & Security (IJCSITS), Vol. 2, No. 3, 2012, pp. 491-497.

[10] Ewa Huebner, Derek Bem and Cheong Kai Wee, "Data hiding in the NTFS file system," Digital Investigation, Vol. 3, Issue 4, 2006, pp. 211-226.

[11] Christopher Lees, "Determining removal of forensic artefacts using the USN change journalOriginal," Digital Investigation, Vol. 10, Issue 4, 2013, pp. 300-310.

[12] 김태한, 조규상, "NTFS 파일 시스템의 저널 파일을 이용한 파일 생성에 대한 디지털 포렌식 방법," 디지털 산업정보학회 논문지, 6권, 2호, 2010, pp. 107-118.

[13] Gyu-Sang Cho, "A computer forensic method for detecting timestamp forgery in NTFS," Computers & Security, Vol. 34, 2013, pp. 36-46.

[14] 조규상, "타임스탬프 변화패턴을 근거로 한 평가함수에 의한 디지털 포렌식 방법," 디지털산업정보학회 논문지, 10권 2호, 2014, pp. 91-105.

[15] Gyu-Sang Cho, "NTFS Directory Index Analysis for Computer Forensics," IMIS 2015(the 9-th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing), July 8th-10th, Blumenau Brazil, 2015.

[16] Jonathan Grier, "Detecting data theft using stochastic forensics," Digital Investigation, Vol. 8, 2011, pp. S-71-77.

■ 저자소개 ■

조 규 상
Cho Gyusang

1996년 3월~현재
동양대학교 컴퓨터정보전학과
교수
2010년 9월~2011년 8월
미국 Purdue대학교, Dept. of
Computer Information
Technology, Cyber Forensic
Lab, Visiting scholar
1997년 2월 한양대학교 전자공학과 (공학박사)

관심분야 : 디지털 포렌식, 시스템보안
E-mail : cho@dyu.ac.kr

논문접수일: 2015년 6월 2일
수 정 일: 2015년 6월 11일
게재확정일: 2015년 6월 15일