KIPS Trans. Softw. and Data Eng.
Vol.9, No.4 pp.123~128 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2020.9.4.123

Detecting Methods of the Database Block Size for Digital Forensics

Sunkyung Kim† · Ji Su Park†† · Jin Gon Shon†††

ABSTRACT

As the use of digital devices is becoming more commonplace, digital forensics techniques recover data to collect physical evidence during the investigation. Among them, the file forensics technique recovers deleted files, therefore, it can recover the database by recovering all files which compose the database itself. However, if the record is deleted from the database, the modified record contents will not be restored even if the file is recovered. For this reason, the database forensics technique is required to recover deleted records. Database forensics obtains metadata from database configuration files and recovers deleted records from data files. However, record recovery is difficult if database metadata such as block size cannot be obtained from the database. In this paper, we propose three methods for obtaining block size, which is database metadata. The first method uses the maximum size of free space in the block, and the second method uses the location where the block appears. The third method improves the second method to find the block size faster. The experimental results show that three methods can correctly find the block size of three DBMSes.

Keywords : Digital Forensics, Database Forensics, Metadata, Block Size

디지털 포렌식을 위한 데이터베이스 블록 크기의 탐지 기법

김 선 경† · 박 지 수†† · 손 진 곤†††

요 약

디지털 기기 사용이 일반화되면서 수사 과정에서 물적 증거 수집을 위해 디지털 포렌식 기법을 사용한다. 이 중 파일 포렌식 기법은 삭제된 파일을 복구하는 것으로, 여러 개의 파일로 구성된 데이터베이스가 삭제되어도 복구할 수 있다. 그러나 데이터베이스에서 레코드가 삭제된 경우는 파일 복구를 하여도 수정된 레코드 내용이 복원되지 않는다. 이에 삭제된 레코드를 복구하는 기법인 데이터베이스 포렌식이 필요하다. 데이터베이스 포렌식은 데이터베이스 설정 파일로부터 메타데이터를 획득하고, 데이터 파일에서 삭제된 레코드를 복구한다. 그러나 데이터베이스에서 블록 크기와 같은 데이터베이스 메타데이터를 획득하지 못하면 레코드 복구가 어렵다. 본 논문에서는 데이터베이스 메타데이터인 블록 크기를 탐지하기 위한 세 가지 방법을 제안한다. 첫 번째 기법은 블록에 존재하는 빈공간의 최대 크기를 이용하며, 두 번째 기법은 블록이 나타나는 위치를 이용한다. 세 번째 기법은 두 번째 기법보다 더 빠르게 블록 크기를 찾을 수 있도록 개선한다. 실험 결과는 세 가지 탐지 기법 모두 세 종류의 DBMS의 블록 크기를 정확하게 찾을 수 있음을 보인다.

키워드 : 디지털 포렌식, 데이터베이스 포렌식, 메타데이터, 블록 크기

1. 서 론

최근 데이터를 다루는 컴퓨터 시스템에서는 데이터베이스의 사용이 늘고 있다[1]. 데이터베이스는 작게는 개인이 데이터를 처리하는 스마트폰에서부터 크게는 기업의 대량 데이터를 처리를 기업 정보시스템까지 다양한 곳에서 이용된다. 데이터베이스 시스템을 이용하는 기관이나 사람의 각종 주요 데이터가 데이터베이스에 저장되어있다. 예를 들면 스마트폰에 있는 데이터베이스에는 전화번호 목록, 전화 수·발신 기록, 인스턴스 메시지 대화 내용 등이 저장된다. 기업에서는 데이터베이스에 고객과의 거래 명세나 내부 회계 처리 명세 등이 저장된다. 즉, 사람에 의해서 발생한 행위가 지속해서 데이터베이스에 기록되고 있다.

그리고 범죄도 사람의 행위이며 따라서 범죄가 발생하였을 때 범죄자가 남긴 흔적들을 데이터베이스에 찾을 수 있다. 이런 이유는 범죄자는 이런 흔적을 삭제하거나 변조한다. 예를 들어 개인은 범죄 행위 과정에서 남긴 인스턴스 메시지를 삭제하거나 스마트폰을 훼손한다. 기업의 예로는 데이터베이스에 기록되어 있는 회계 데이터를 조작한다. 이런 경우 수사기

※ 이 논문은 2019년 한국방송통신대학교 학술연구비 지원을 받아 작성된 것임.
† 준 회 원 : 한국방송통신대학교 정보과학과 석사과정
†† 정 회 원 : 전주대학교 컴퓨터공학과 교수
††† 종신회원 : 한국방송통신대학교 컴퓨터과학과 교수
Manuscript Received : December 3, 2019
Accepted : January 20, 2020
* Corresponding Author : Jin Gon Shon(jgshon@knou.ac.kr)

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/ licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

관 입장에서는 데이터베이스를 압수하였다 하더라도 훼손된 데이터를 복구하지 못하면 물증을 확보하는 데 실패한다. 그러므로 수사기관은 물증 확보를 위하여 데이터베이스에서 훼손된 데이터를 복구하여야 한다[2].

데이터베이스의 데이터 파일은 메타데이터와 실제 데이터로 구성되어 있다[3]. 그리고 데이터베이스에서 레코드를 삭제하더라도 성능을 위하여 DBMS(Database Management System)는 레코드의 모든 값을 지우지 않고 삭제 여부만을 표시하는 것이 일반적이다. 따라서 데이터 파일을 분석하여 데이터베이스의 구성을 알 수 있는 메타데이터를 얻고, 삭제된 레코드 영역을 분석하여 삭제 이전 값을 얻을 수 있다. 이에 많은 데이터베이스 포렌식 연구에서는 메타데이터를 가지고 있는 파일에서 매개변수 파일을 획득하거나 중요 메타데이터를 미리 알고 있다는 가정하에 데이터베이스 포렌식 연구를 제안한다[4].

그러나 데이터베이스 파일이 저장된 디스크나 플래시 메모리와 같은 저장장치가 침수와 같은 물리적 훼손이 있으면 모든 데이터베이스 파일을 복구할 수 없다. 이런 물리적 훼손으로 인해 매개변수 파일을 획득하지 못했을 때는 데이터베이스 메타데이터를 직접적으로 찾을 수 없다. 특히, 데이터 파일을 구성하는 블록 크기를 알 수 없으면 데이터 파일을 기초 분석 단위인 블록 단위로 나눌 수 없기 때문에 데이터베이스 포렌식을 진행하기 어렵다.

본 논문에서는 일부의 데이터 파일만을 획득하고 필수 매개변수 파일을 획득하지 못했을 때 기본 데이터베이스 포렌식 기법을 적용하기 위해 필수적으로 필요한 블록 크기를 탐지하는 세 가지 기법을 제시한다. 첫 번째 기법은 블록에서 발견되는 최대 빈공간의 크기를 기반으로 블록의 크기를 계산하는 것이다. 기존 상용 또는 오픈소스 DBMS가 가지는 블록 크기의 규칙을 이용하여 계산할 수 있다. 두 번째 기법은 첫 번째 기법에서 가정한 블록 크기의 규칙을 따르지 않더라도 적용이 가능한 방법이다. 즉, 빈공간이 나타나는 패턴을 이용하여 블록의 크기를 계산하는 방법이다. 세 번째 기법은 두 번째 기법을 개선하여 좀 더 적은 수의 빈공간을 블록을 읽고 블록의 크기를 찾을 수 있는 방법이다.

본 논문에서 제시한 세 가지 블록 크기 탐지 방법은 실험을 통해서 유효성을 평가하였다. 실험 결과, 세 가지 기법 모두 블록의 크기를 정확히 계산할 수 있음을 확인하였다. 즉, 첫 번째 기법으로 세 가지 종류의 DBMS 파일에 대해서 실험한 결과 블록의 크기를 정확히 계산하였다. 또한, 가상의 데이터 파일을 만들어서 두 번째 기법으로 실험한 결과 259개의 빈공간의 위치를 이용하여 블록의 크기를 정확히 계산하였다. 그리고 두 번째 기법을 실험할 때 사용하였던 동일한 파일을 이용해서 세 번째 기법으로 실험한 결과 79개의 빈공간의 위치를 이용하여 블록의 크기를 정확히 계산하였다.

본 논문에서 제시한 세 가지 기법은 데이터베이스의 블록 크기를 정확히 탐지하는 방법으로서 어느 기법이나 매개 변수 파일에 의존하지 않고 데이터 파일만으로 블록 크기를 확인할 수 있어서 데이터베이스 포렌식을 진행할 수 있다는 장점이 있다. 즉, 기존 데이터베이스 포렌식 기법에서는 매개변수 파일을 확보하지 못했을 경우, 데이터 파일이 있음에도 포렌식을 수행할 수 없었으나, 제안하는 기법들은 데이터 파일만을 이용하여 포렌식을 수행할 수 있다는 점이 포렌식 분야에 대한 본 논문의 기여도라고 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 데이터베이스 포렌식 연구에 대해서 설명하고 데이터베이스 블록 구조에 대해서 설명한다. 3장에서는 블록 크기의 탐지 기법 두 가지에 대한 원리를 설명한다. 4장에서는 3장에서 제시된 기법을 실험하고 해당 실험 결과를 분석하여 개선된 기법을 추가로 제시하고 실험한다. 그리고 5장 결론으로 세 가지 탐지 기법을 정리하고 향후 연구과제를 다룬다.

2. 관련 연구

2.1 데이터베이스 포렌식

데이터베이스 포렌식이란 디지털 포렌식의 한 분야로 데이터베이스에서 어떤 일이 언제, 왜, 누구에 의해서 발생했는지를 찾아내는 작업이다[5]. 데이터베이스 포렌식의 일반적인 수행 절차는 데이터베이스를 구성하는 파일을 획득하고 설정 파일로부터 얻은 메타데이터를 이용하여 데이터 파일의 구성을 파악한 후 데이터 파일을 직접 탐색하여 삭제된 레코드를 찾아내는 과정이다. 따라서 데이터 파일에 기록되었다가 삭제된 레코드를 추출하는 작업인 데이터베이스 포렌식을 수행하기 위해서는 데이터베이스 데이터 파일을 탐색하기 전에 데이터 파일의 구성에 대해 알 수 있는 메타데이터를 정확히 파악해야 한다.

DBMS의 디스크 이미지 또는 램 스냅숏을 분석하여 총 8종의 DBMS를 대상으로 데이터베이스 포렌식 기법을 제시하고 도구를 제안한 연구가 James Wagner에 의해 수행되었다[3]. 이 연구에서 데이터베이스 저장 구조를 분석하는 방법을 제시하고 있다. 가장 먼저 블록 크기를 탐지한 후 블록 크기로 파일을 나누고 변경되었거나 삭제된 레코드를 포함한 블록 내의 레코드를 읽어내는 방법을 제시하고 실험하였다. James Wagner는 이후 데이터베이스 포렌식 결과를 기록하는 파일 포맷과 데이터베이스 포렌식 툴킷을 제시하였다[6]. 이 연구에서 데이터베이스 포렌식 결과 항목 중 하나로 블록 크기를 제시하였다.

오라클 데이터베이스, MySQL, 마이크로소프트 SQL Server 각각에 대한 단일 DBMS를 대상으로 한 다른 연구에서도 블록 혹은 블록 크기 등의 메타 정보를 파악하여 데이터베이스를 분석하였다. 오라클 데이터베이스 포렌식을 제안한 연구에서 블록 크기를 얻기 위해서 메타데이터를 데이터베이스 로그 파일을 분석하여 수행하였다[7]. MySQL 및 마이크로소프트 SQL Server 데이터베이스 포렌식을 제안한 연구에서는 블록 크기가 일정하다고 가정하고 연구를 하였다[8-10].

관련 연구를 통하여 데이터베이스 포렌식 과정 중 주요 메

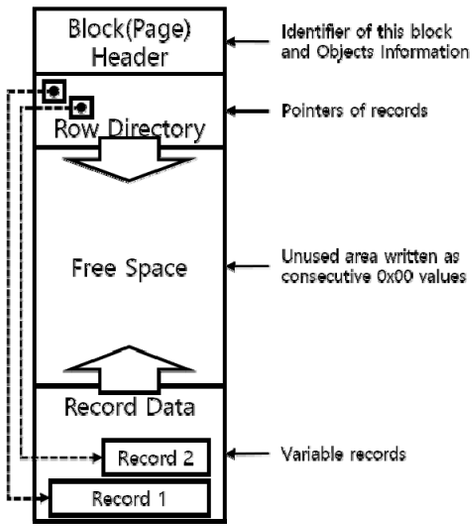


Fig. 1. Slotted Page Structure

타데이터인 블록 크기를 먼저 탐지하지 못하면 포렌식을 할 수 없음을 알 수 있다. 본 연구의 기법을 이용할 경우 파일을 일부만 획득한 경우에도 포렌식을 수행할 수 있다.

2.2 데이터베이스 블록 구조

데이터베이스는 블록 공간을 효율적으로 관리하기 위하여 Fig. 1과 같이 슬롯 페이지 구조(Slotted Page Structure)를 이용한다. 슬롯 페이지 구조에서 페이지는 데이터 블록을 의미한다. 슬롯 페이지 구조에서 가변길이레코드는 블록 가장 끝에서부터 입력되는 순으로 위쪽으로 채워진다. 레코드의 길이가 일정하지 않기 때문에 각 레코드의 시작 주소는 계산식으로 알 수 없다. 따라서 레코드 시작 주소를 알기 위하여 레코드의 시작 주소는 블록 헤더 바로 뒤에 기록한다. 이렇게 기록한 영역을 로우 디렉터리라 부르며 헤더 뒤에서부터 아래로 빈 곳을 채워나가는 구조를 가진다[11]. DBMS 종류에 따라 로우 디렉터리가 블록 아래서부터 채워지고 실제 레코드는 블록 헤더 바로 뒤에서부터 채워지는 반대의 구조를 가진다. 본 논문 실험 대상인 오라클 데이터베이스와 SQLite 데이터베이스는 레코드 데이터가 바닥에서 올라오는 구조이고, MySQL InnoDB는 블록 헤더 다음부터 아래로 채워지는 구조로 된다.

DBMS는 종류별 가질 수 있는 블록 크기가 다양하다. 동일한 DBMS를 사용한다고 하더라도 설정에 따라 다양한 블록 크기를 가진다. 따라서 DBMS의 종류를 알더라도 블록 크기를 탐지할 수는 없다. 그리고 현재 사용 중인 상용 또는 오픈소스 DBMS의 경우 한 데이터베이스 내에서는 모든 블록이 동일한 크기로 되어 있다[3].

3. 블록 크기의 탐지 기법

3.1 블록 크기의 규칙성을 이용한 기법

슬롯 페이지 구조에서는 블록 앞부분에 블록 헤더와 로우 디렉터리가 위치하고 블록 바닥부터 레코드가 채워진다. DBMS 종류에 따라 로우 디렉터리와 레코드 위치가 서로 바뀐다. 이런 특성으로 인하여 블록 중간에는 연속된 0x00 값으로 채워진 빈공간이 존재한다. 블록 시작과 끝에는 블록 헤더, 레코드, 로우 디렉터리 등으로 채워져 있음으로 빈공간 크기는 블록 크기보다 작다. 또한, 상용 또는 오픈 소스 DBMS에서 사용하는 블록 크기는 Table 1을 보면 디스크의 섹터 크기인 512 Bytes의 정수배이다. 그 이유는 디스크의 물리적 입출력 단위와 정수배가 되는 것이 입출력에 유리하기 때문이다. 따라서 블록 크기 S는 Equation (1)과 같다.

Table 1. Block Sizes of DBMSes

Block Size	Oracle	MySQL InnoDB	SQLite	DB2
0.5KB			O	
1KB			O	
2KB	O		O	
4KB	O	O	Default	Default
8KB	Default	O	O	O
16KB	O	Default	O	O
32KB	O	O	O	O
64KB		O	O	

$$S = K \times 2^n \tag{1}$$

여기서, K는 디스크 섹터 크기로 사용되는 512이며 n은 자연수이다.

데이터 파일의 조사된 블록 중 적어도 하나의 빈공간의 크기가 블록 크기의 50%를 초과하는 것으로 가정한다. 그러면 가장 큰 빈공간 크기도 50%를 초과하며 Equation (2)가 성립한다.

$$\frac{S}{2} < M < S \tag{2}$$

여기서, M은 가장 큰 빈공간 크기이다.

Equation (1)의 값을 Equation (2)에 대입하면 Equation (3)과 같이 n의 값을 구할 수 있다.

$$\begin{aligned} &\frac{K \times 2^n}{2} < M < K \times 2^n \\ &\Rightarrow 2^{n-1} < \frac{M}{K} < 2^n \\ &\Rightarrow n-1 < \log_2 \frac{M}{K} < n \\ &\Rightarrow n = \lceil \log_2 \frac{M}{K} \rceil \end{aligned} \tag{3}$$

Equation (3)을 Equation (1)에 대입하면 블록 크기 S를 Equation (4)와 같이 구할 수 있다.

$$S = K \times 2^{\lceil \log_2 \frac{M}{K} \rceil} \quad (4)$$

대부분의 운영 데이터베이스에서 데이터 파일은 데이터가 증가할 것을 대비하여 대부분 비어 있는 블록이 존재한다. 따라서 Equation (2)의 가정을 대부분의 데이터베이스에서 만족한다. 따라서 데이터 파일을 읽어나가면 최대 빈공간 크기를 검출한 후 Equation (4)를 이용하여 데이터베이스 블록 크기 탐지이 가능하다. 본 기법을 MFS(Max size of Free-Spaces)라고 하겠다. 그리고 4장에서 실험 결과를 보여 주겠다.

3.2 빈공간 위치를 이용하는 기법

블록 크기가 Equation (1)을 따르지 않는 DBMS를 만들 수 있다. Equation (1)을 따르지 않더라도 데이터베이스 내에서 블록 크기가 일정하고 슬롯 블록 구조를 가진다면 이럴 때에도 블록 크기를 찾을 수 있다. 블록 크기가 Equation (1)을 따르지 않더라도 블록 중간에 빈공간이 존재한다. 다수의 연속된 블록의 빈공간 위치를 추적하여 블록 크기를 찾을 수 있다. 블록 크기를 찾는 방법에 대해 수학적 문제로 정의하면 아래와 같다. 연속해서 문제의 해법을 제시하였다.

문제 정의: 일정한 개수의 숫자가 들어 있는 연속된 공간을 블록라고 한다. 블록의 크기는 들어 있는 숫자의 개수이다. 블록의 앞쪽과 뒤쪽 일정 영역을 0이 아닌 숫자로 채워져 있고 중앙에는 0으로 채워져 있다. 중앙에 0으로 채워진 영역의 크기는 일정하지 않다. 연속된 블록의 모임을 읽어서 블록의 크기를 찾는다.

해법: n번째 0으로 채워진 영역의 시작점 주소(offset: 첫 번째 블록의 시작점부터의 거리를 주소로 함)를 B_n이라 하고 종료점 주소를 E_n라 한다. E_n는 n번째 블록을 넘지 못하므로 블록 크기를 S이라고 할 때 nS 보다 작다. 또한 B_{n+1}는 n번째 블록 다음에 나오므로 nS 보다 크다. 따라서 Equation (5)과 같이 전개할 수 있다. 여기서 E_n, S, B_{n+1}, n은 자연수다.

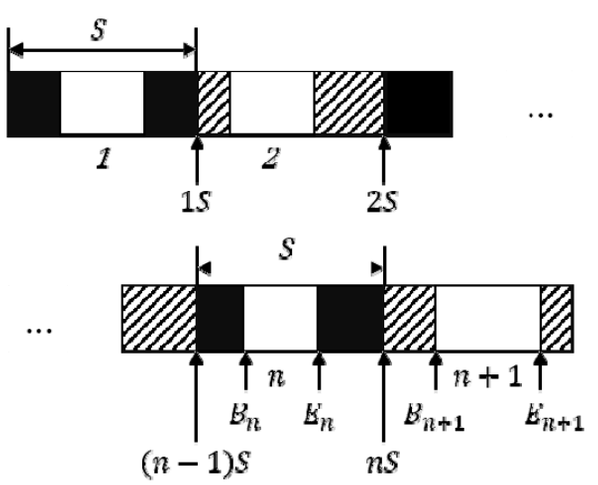


Fig. 2. Locations of Free-Spaces in Blocks

따라서 $\frac{B_{n+1}}{n} - \frac{E_n}{n} < 1$이면 자연수 S를 특정할 수 있다.

$$E_n \le nS < B_{n+1} \quad (5)$$
$$\Rightarrow \frac{E_n}{n} \le S < \frac{B_{n+1}}{n}$$

$$B_{n+1} - E_n < 2S \quad (6)$$
$$\Rightarrow \frac{B_{n+1} - E_n}{2S} < 1$$
$$\Rightarrow \frac{B_{n+1}}{n} - \frac{E_n}{n} < 1 \qquad (\text{if } n \ge 2S)$$

그리고 E_n과 B_{n+1}는 $E_n < B_{n+1}$이며 연속된 두 블록 안에 위치하므로 Equation (6)를 만족한다.

Equation (6)에서 보듯이 n이 $2S$보다 큰 데이터 파일이 확보된다면 빈공간 위치를 이용하여 블록 크기를 확인할 수 있다. Fig. 2를 보면 Equation (5), (6)에서 사용한 각 값들을 연속선상에서 확인할 수 있으며 각 식에 대해서 이해를 도울 수 있다. 본 기법을 PFS(Pattern of Free-Spaces)라고 하겠다. 그리고 4장에서 PFS를 실험하고 실험 결과를 바탕으로 좀 더 적은 블록으로 블록 크기를 찾을 수 있는 방법을 착안하고 실험한다. 그리고 PFS 실험을 통해 얻은 결과에서 좀 더 적은 수의 블록을 조사하여 블록 크기를 결정할 수 있는 방법을 제시하고 실험한다.

4. 블록 크기의 탐지 실험

4.1 블록 내 빈공간 존재 확인

3장에서 제시한 두 가지 기법 모두 블록 내에서 빈공간을 확인할 수 있어야 한다. 3.1절은 빈공간의 크기를 이용하며 3.2절은 빈공간의 위치를 이용한다. 따라서 본 실험에서는 블록 내의 빈공간을 시각화해서 확인하는 실험을 한다.

실험에 사용되는 데이터베이스는 스마트폰에서 사용되는 SQLite와 기업에서 주요 업무에 많이 사용되는 오라클 데이터베이스, MySQL이다. Fig. 3은 0x00이 아닌 값은 붉은색이나

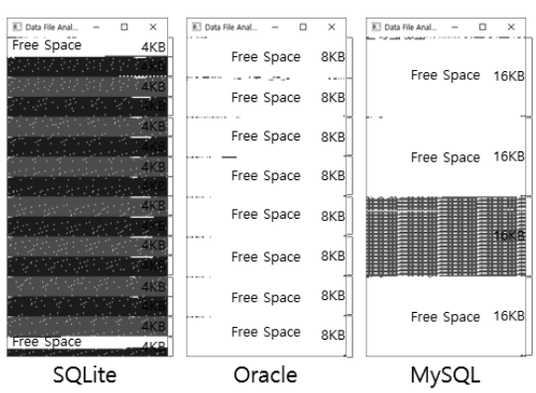


Fig. 3. Visualizing Free Space

Table 2. Result of MFS

DBMS	Actual Block Size (Bytes)	Max Size of Free Space (Bytes)	Result
SQLite	4,096	3,886	4,096
Oracle	8,192	8,160	8,192
MySQL	16,384	16,350	16,384

푸른색으로 표현했으면 0x00 값은 흰색으로 표현하였다. SQLite, 오라클 데이터베이스, MySQL 모두 블록 사이에 연속된 0x00 값이 나타나는 빈공간이 있는 것을 확인할 수 있다.

4.2 MFS를 이용한 블록 크기의 탐지 실험

MFS을 파이썬(Python)으로 구현하여 블록 크기가 4,096 Bytes로 설정된 SQLite, 8,192 Bytes로 설정된 오라클 데이터베이스, 16,384 Bytes로 설정된 MySQL 데이터 파일을 이용하여 실험하였다. 실험 결과 Table 2와 같다. 각 DBMS 별 최대 빈공간 크기는 3,886 Bytes, 8,160 Bytes, 16,350 Bytes임을 확인했다. 최대 빈공간 크기를 Equation (1)에 대입하면 각각의 블록 크기는 4,096, 8,192, 16,384로 계산되었다. 실제의 블록 크기와 MFS로 찾아낸 값이 일치함을 확인했다.

4.3 PFS를 이용한 블록 크기의 탐지 실험

PFS를 파이썬으로 구현하여 블록 크기가 5,678 Bytes이며 중앙에 빈공간이 존재하는 데이터 파일을 이용하여 실험했다. n이 점차 커지면서 블록 크기는 $\frac{E_n}{n}, \frac{B_{n+1}}{n}$ 사이의 값임을 Equation (5)에서 보았다. 우리는 Equation (5)에서 $\frac{B_{n+1}}{n}$을 블록 크기 상한이라고 하고 $\frac{E_n}{n}$을 블록 크기 하한이라 하겠다. 실험을 하여 n이 증가함에 따라 상한과 하한의 변화를 차트로 표시한 것이 Fig. 4이다. n이 증가함에 따라 상한과 하한이 불규칙하게 진동하지만 블록 크기로 수렴하는 것을 볼 수 있다. n이 259일 때 상한과 하한의 자연수는 5,678 하나만 존재하게 된다. 따라서 블록 크기가 5,678 Byte인 데이터 파일의 블록 크기를 탐지할 수 있었다.

4.4 개선된 PFS 제안 및 실험

Fig. 4에 중앙 수평선은 블록 크기를 나타낸다. 본 실험 결과를 보면 블록 크기의 상한과 하한이 데이터 파일을 읽어나가면서 블록 크기로 대체적으로 수렴은 하지만 값이 수렴하는 반대 방향으로 변하는 것도 볼 수 있다. 따라서 앞에서 찾은 상한과 하한보다 수렴하지 않는 값으로 변할 경우 그 값을 무시하고 앞에서 찾은 상·하한을 채택할 경우 좀 더 빨리 블록 크기를 찾을 수 있다. 따라서 이를 반영한 개선된 방법에 대한 Equation (7)과 같다.

$$\max\left(\frac{E_i}{i}\right) \le S < \min\left(\frac{B_{j+1}}{j}\right) \tag{7}$$

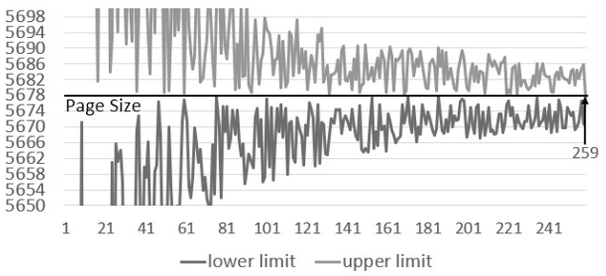


Fig. 4. Result of PFS

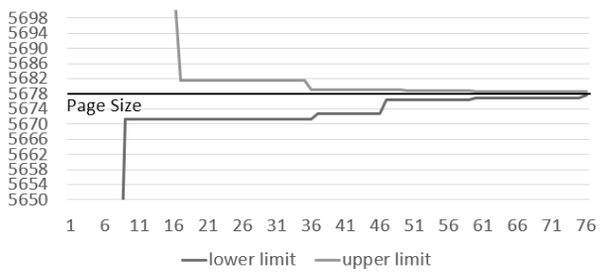


Fig. 5. Result of APFS

여기서 i,j는 자연수이다.

Equation (7)을 이용한 기법을 APFS(Advanced PFS)라 하겠다. 하한을 탐색하며 찾은 최댓값과 상한을 탐색하며 찾은 최솟값을 이용하여 동일한 파일로 실험한 결과가 Fig. 5이다. APFS를 이용했을 때 76개의 빈공간 탐색만으로 블록 크기를 탐지함을 확인할 수 있었다.

일반적으로 많이 사용되는 블록 크기인 Equation (1)을 만족하지 않는 블록에 대해서도 PFS로 블록 크기를 탐지할 수 있음을 확인했다. 또한 파일을 탐색이 이미 수렴한 상·하한의 최댓값과 최솟값을 이용한 APFS로 보다 효율적으로 블록 크기를 탐지할 수 있음을 확인했다.

5. 결 론

본 논문은 디지털 포렌식을 이용하기 위해 데이터베이스의 메타데이터를 직접 확보하지 못했을 때에도 포렌식을 할 수 있도록 데이터베이스의 블록 크기를 탐지하는 기법에 관한 것이다. 즉, 슬롯 페이지 구조를 가지는 데이터 파일 블록에 존재하는 빈공간의 크기와 위치를 분석함으로써 블록 크기를 탐지할 수 있는 기법을 제안하였고 실험을 통하여 기법의 유효성을 증명하였다.

현재 일반적으로 많이 사용되고 있는 상용 또는 오픈소스 데이터베이스의 블록 크기는 Equation (1)의 규칙을 따른다. Equation (1)의 규칙을 따르고 최대 빈공간 크기가 블록 크기의 절반 크기를 초과하는 것이 존재하는 경우 최대 블록 크기를 Equation (4)를 대입하여 세 가지 DBMS 파일에 대해서 블록 크기를 정확히 탐지하였다. 만약 Equation (1)의 규칙을 따르지 않더라도 충분한 블록과 빈공간 위치를 확인할 수 있다면

Equation (5)을 이용하여 블록 크기를 확인할 수 있음을 Equation (6)를 통하여 증명하였다. 또한 Equation (5)의 블록 크기의 발견되는 상·하한 최솟값과 최댓값이 수렴하는 것을 이용하여 좀 더 적은 수의 빈공간을 읽어 블록 크기를 정확히 탐지하였다. 개선된 기법은 Equation (7)로 정리하였다.

일반적인 경우 데이터 파일은 Equation (1)의 규칙을 따른다. 따라서 첫 번째 기법으로 블록 크기를 정확히 탐지할 수 있으며 연산도 가장 간단하다. Equation (1)의 규칙을 따르지 않는 DBMS의 경우 두 번째와 세 번째 기법 블록 크기를 탐지할 수 있으며 이중 세 번째 기법이 보다 효율적이다.

향후 연구 과제로서 블록 저장 구조를 가지는 파일에 대해 포렌식을 수행하고자 할 때에도 본 논문에서 제시한 블록 크기 탐지 기법을 이용하기 위한 연구가 필요하다. 또한, 매개변수를 탐지하지 못하여 저장 구조 파악을 못할 때 데이터의 빈공간을 이용하여 저장 구조를 파악하는 디지털 포렌식 기법으로 확장할 수 있을 것이다.

References

[1] Min Chen, Shiwen Mao, and Yunhao Liu, "Big Data: A Survey," *Mobile Networks and Applications*, Vol.19, No.2, pp.171-209, 2014.

[2] Supreme Proscutors' Office ROK, Scientific Investigation - Digital Investigation Support [Internet], https://www.spo.go.kr/site/spo/02/1020604000000 2018100812.jsp. [Accessed Oct. 24, 2018.

[3] James Wagner, Alexander Rasin, and Jonathan Grier, "Database Forensic Analysis Through Internal Structure Carving," *Digital Investigation*, Vol.14, pp.106-115, 2015.

[4] R. Chopade and V. K. Pachghare, "Ten Years of Critical Review on Database Forensics Research," *Digital Investigation*, Vol.29, pp.180-197, 2019.

[5] M. A. M. Guimaraes, R. Austin, and H. Said, "Database Forensics," in *Information Security Curriculum Development Conference*, pp.62-65, 2010.

[6] James Wanger, Alexander Rasin, Karen Hear, Rebecca Jacob, and Jonathan Grier, "DB3F & DB-Toolkit: The Database Forensic File Format and the Database Forensic Toolkit," *Digital Investigation*, Vol.29, pp.42-50, 2019.

[7] Jong-Hyun Choi, DooWoo Jeong, and Sangjin Lee, "The method of recovery for deleted record in Oracle Database," *Journal of The Korea Institue of Information Security & Cryptology*, Vol.23, No.5, pp.947-955, 2013.

[8] Jeewon Jang, Doowon Jeoung, and Sang Jin Lee, "The Recovery Method for MySQL InnoDB Using Feature of IBD Structure," *KIPS Tr. Comp. and Comm. Sys.* Vol.6, No.2, pp.59-66, 2017.

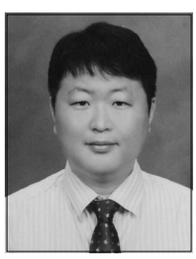
[9] Jiho Shin, "Comparison of Remaining Data According to Deletion Events on Microsoft SQL Server," *Journal of The KIIS&C*, Vol.27, No.2, pp.223-232, 2017.

[10] Jung Sung Kyun, Jee Won Jan, Doo Won Jeong, and Sang Jin Lee, "A Study on the Improvement Method of Delete Record Recovery in MySQL InnoDB," *KIPS Tr. Comp. and Comm. Sys.*, Vol.6, No.12, pp.487-496, 2017.

[11] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database System Concepts 6th Edition," New York: McGraw-Hill Education, pp.456-457, Jan. 2010.

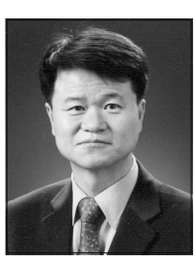
김 선 경

https://orcid.org/0000-0003-3160-4993
e-mail : skkim@wizbase.co.kr
1996년 고려대학교 전산학과(학사)
2017년 한국방송통신대학교 관광학과(학사)
2017년 ~ 현 재 한국방송통신대학교 정보과학과 석사과정
2012년 ~ 현 재 ㈜위즈베이스 기술이사
관심분야 : 유사도, 데이터베이스 포렌식, 데이터베이스 성능 최적화, 데이터 관리



박 지 수

https://orcid.org/0000-0001-9003-1131
e-mail : jisupark@jj.ac.kr
2013년 고려대학교 컴퓨터교육과(박사)
2015년 ~ 2018년 충남대학교 초빙교수
2018년 ~ 2019년 경기대학교 교양학부 조교수
2019년 ~ 2020년 동국대학교 융합교육원 교수
2020년 ~ 현 재 전주대학교 컴퓨터공학과 교수
2020년 ~ 현 재 한국정보처리학회 이사 및 JIPS 간사
관심분야 : 분산 시스템, 모바일 클라우드 컴퓨팅, e-Learning



손 진 곤

https://orcid.org/0000-0002-0540-4640
e-mail : jgshon@knou.ac.kr
1991년 고려대학교 전산학전공(박사)
1991년 ~ 현 재 한국방송통신대학교 컴퓨터과학과 교수
1997년 ~ 1998년 State University of New York (Stony Brook) Visiting Professor
2000년 ~ 현 재 ISO/IEC JTC1/SC36 Korea Delegate
2009년 ~ 현 재 이러닝학회 부회장
2010년 한국정보처리학회 부회장
관심분야 : 컴퓨터통신망, 분산시스템, e-Learning, 정보기술 표준화