Journal of The Korea Institute of Information Security & Cryptology
VOL.24, NO.5, Oct. 2014
ISSN 1598-3986(Print)
ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.5.861

파일 오브젝트 분석 기반 개선된 물리 메모리 실행 파일 추출 방법

강 영 복,[1†] 황 현 욱,[2‡] 김 기 범,[2] 노 봉 남[1]
[1]전남대학교 시스템보안연구센터, [2]ETRI 부설연구소

An improved extraction technique of executable file from physical memory by analyzing file object

Youngbok Kang,[1†] Hyunuk Hwang,[2‡] Kibom Kim,[2] Bongnam Noh[1]
[1]Chonnam National University, [2]The Attached Institute of ETRI

요 약

악성코드의 지능화에 따라 물리 메모리에서 실행 파일을 추출하는 것이 중요한 연구 이슈로 부각되고 있다. 물리 메모리에서 파일 데이터를 추출하는 경우 일반적으로 프로그램 실행과정에서 사용 중인 파일 데이터를 추출하기 때문에 원본 파일 데이터가 추출되지 않는 문제점이 있다. 따라서 물리 메모리에 저장되는 파일 정보를 분석하고 이를 기반으로 디스크에 저장된 파일과 동일하게 추출하는 방법이 요구된다.

본 논문에서는 윈도우 파일 오브젝트 커널 정보 분석을 통한 실행 파일 데이터 추출 방법을 제시한다. 실험을 통해 물리 메모리에 저장되어있는 실행 파일 데이터 특징을 분석하고, 기존 방법과 비교하여 원본 파일 데이터를 효과적으로 추출함으로써 제안 방법의 우수함을 증명한다.

ABSTRACT

According to the intelligence of the malicious code to extract the executable file in physical memory is emerging as an import researh issue. In previous physical memory studies on executable file extraction which is targeting running files, they are not extracted as same as original file saved in disc. Therefore, we need a method that can extract files as same as original one saved in disc and also can analyze file-information loaded in physical memory.

In this paper, we provide a method that executable file extraction by analyzing information of Windows kernel file object. Also we analyze the characteristic of physical memory loaded file data from the experiment and we demonstrate superiority because the suggested method can effectively extract more of original file data than the existing method.

Keywords: Physical Memory Forensic, File Mapped Data, File Object

I. 서 론

최근 디지털 포렌식 연구에서는 하드디스크를 이용한 증거 수집 한계성을 극복하고자 물리 메모리를 이용한 다양한 증거 정보 수집 연구를 수행하고 있다. 기존 물리 메모리 덤프 데이터 수집 방법에서는 메모리 덤프 도구를 이용하여 실시간 메모리 덤프 데이터를 수집하였다. 하지만 최근 절전모드 파일을 이용한 메모리 정보 수집 방법[1]이 연구되면서 하드디스크를 이용한 물리 메모리 덤프 데이터 분석이 가

접수일(2014년 8월 6일), 수정일(2014년 9월 11일)
게재확정일(2014년 9월 12일)
† 주저자, k-dupe@nate.com
‡ 교신저자, hhu@ensec.re.kr(Corresponding author)

능하게 되었다.

물리 메모리에는 커널 데이터 정보뿐만 아니라 현재 실행중인 프로세스에서 참조하고 있는 파일 및 캐시 목적으로 저장된 파일 정보를 저장하고 있다. 물리 메모리에서 추출되는 파일 정보는 하드디스크에 저장된 파일 정보와 동일하며, 메모리 덤프 시점에서 사용 중인 파일 정보를 추출할 수 있기 때문에 증거 수집 및 분석에 있어 중요한 정보라고 할 수 있다. 또한 최근 64비트 OS가 보편적으로 사용되면서 물리 메모리 사용 용량도 점차 커지고 있으며, 물리 메모리에 저장되는 파일 정보 또한 증가하고 있다. 따라서 물리 메모리에서 저장된 파일 데이터 구조 및 특징을 분석하고 대용량 메모리에서도 효과적으로 추출할 수 있는 방법에 대한 연구가 필요하다.

본 논문에서는 윈도우 7 32/64비트 물리 메모리 데이터를 대상으로 파일 오브젝트 커널 구조 방법을 기반으로 한 실행 파일 데이터 추출 기법을 제안한다.

II. 관련 연구

윈도우 메모리 관리자는 파일 오브젝트 구조체와 VAD(Virtual Address Descriptor) 구조체를 이용하여 메모리에 저장된 모든 파일 데이터를 관리한다. 빠른 파일 실행을 위해 캐시목적으로 메모리에 저장되는 파일은 파일 오브젝트를 이용하여 관리된다. 현재 실행중인 프로세스에서 사용 중인 파일은 파일 오브젝트와 연결된 VAD 구조체를 이용하여 관리한다.

2.1 윈도우 파일 오브젝트 커널 구조체 구성

파일 오브젝트와 관련된 커널 구조체는 Fig 1과 같다. 파일 오브젝트와 관련된 커널 구조체는 Fig 1과 같다. 파일 경로는 _FILE_OBJECT 구조체에서 관리하며, _FSTRL_ADVANCED_FCB_HEADER 구조체 정보를 이용하여 파일 데이터 크기 정보를 관리한다. _FILE_OBJECT 구조체에는 파일 섹션 객체를 관리하는 _SECTION_OBJECT_POINTERS 구조체 포인터 값을 저장하고 있다.

_SECTION_OBJECT_POINTER 구조체 멤버 변수 DataSectionObject와 ImageSectionObject에서는 파일 데이터에 대한 총괄적인 정보를 관리하고 있는 _CONTROL_AREA 구조체 포인터를 저장하고 있다.

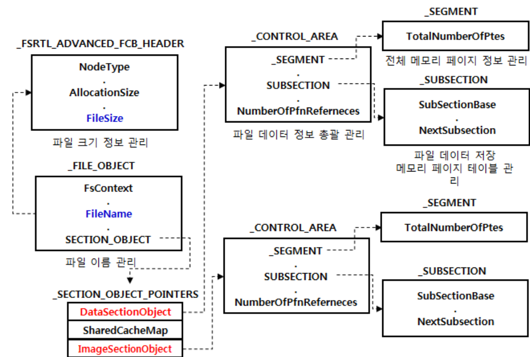

Fig. 1. File object Kernel Struct [2][3]

Fig. 1과 같이 _CONTROL_AREA 구조체에서는 파일 데이터가 저장된 메모리 페이지 주소를 관리하는 _SUBSECTION 구조체 포인터와 메모리 페이지 테이블에 저장 가능한 전체 페이지 크기 정보를 관리하는 _SEGMENT 구조체 포인터를 저장하고 있다. 또한 내부 멤버 변수 NumberOfPfnReferences에는 메모리 페이지 테이블에서 유효한 주소 값이 저장된 주소 개수가 저장되어 있다.

2.2 실행 프로세스에서 사용하는 파일 정보 관리

윈도우 시스템에서 실행중인 프로세스는 VAD 구조체를 이용하여 현재 사용 중인 메모리 영역을 관리한다[4]. Fig 2와 같이 VAD 구조체는 이진 벨런스 트리 형태로 구성되며, 최상위 루트 VAD에 대한 포인터는 프로세스 정보를 관리하는 _EPROCESS 구조체의 멤버 변수 VAD Root에 저장되어있다.

VAD 구조체에서는 관리 메모리 영역의 타입, 관리하는 가상 주소, 파일 오브젝트 관련 커널 구조체 포인터(control area) 정보를 저장하고 있다. 메모리 영역의 타입은 힙(heap)과 이미지(image) 타입으로

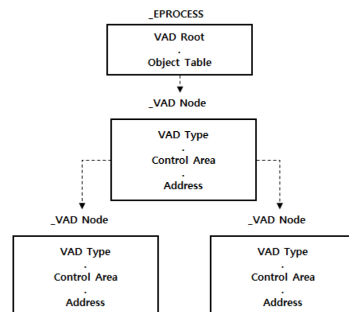

Fig. 2. Relation of _EPROCESS and VAD Node

구분된다. VAD 영역의 타입이 이미지인 경우, 관리하는 가상 주소에는 파일 데이터가 로드 되어있다.

2.3 물리 메모리 실행 파일 복구 방법 문제점

기존 연구에서의 물리 메모리 실행 파일 복구 방법들에는 몇 가지 문제점들이 존재한다. 첫째, VAD 구조체 분석 기반 실행 파일 추출방법[5][6]에서는 실행 파일에서 사용 중인 메모리 영역만을 이용하여 파일 데이터를 추출하기 때문에 실행 프로세스에서 사용하지 않는 파일 영역은 추출되지 않는 문제점이 있다. 둘째, 파일 오브젝트에서 관리하는 Image SectioObject 커널 구조 분석을 통한 실행 파일 추출 방법[7]에서는 실행 프로세스에서 현재 사용 중인 부분적인 파일 영역과 사용 대기 영역만을 추출하기 때문에 전체 파일 데이터 추출에 문제점이 있다. 셋째, 기존 실행 파일 데이터 추출 방법에서는 실행 중인 프로세스에서 사용하는 실행파일(EXE)과 모듈(DLL)만 추출되기 때문에 종료된 프로세스 파일이나 캐시 목적으로 메모리에 저장된 실행파일은 추출할 수 없는 문제점이 있다. 넷째, 기존 방법에서는 원본 파일 데이터 추출을 우선적으로 보장하지 않기 때문에 추출된 파일이 정상적으로 실행되지 않는 문제점이 있다. 따라서 물리 메모리에 저장된 파일의 특징을 분석하고 디스크에 저장된 파일과 동일하게 추출하는 방법이 필요하다.

III. 연구 내용

3.1 물리 메모리 저장 파일 데이터 관리

물리 메모리에 저장되는 모든 파일은 Fig3과 같이 VAD 구조체와 파일 오브젝트 구조체를 이용하여 관리된다. 실행파일에서 사용 중인 파일 데이터는 VAD 구조체와 파일 오브젝트의 ImageSectionObject에서 중복 관리한다.

VAD 구조체에서는 현재 사용 중인 파일 데이터만 관리하며, ImageSectionObject에서는 사용 중인 파일 데이터 이외에 사용대기 목적으로 메모리에 저장된 파일 데이터를 관리한다. DataSectionObject에서는 프로그램 최초 실행과정에서 캐시 목적으로 메모리에 저장된 원본 파일 데이터를 관리한다. DataSectionObject에서 관리하는 파일 데이터는 ImageSectionObject의 사용 대기 파일 데이터 영

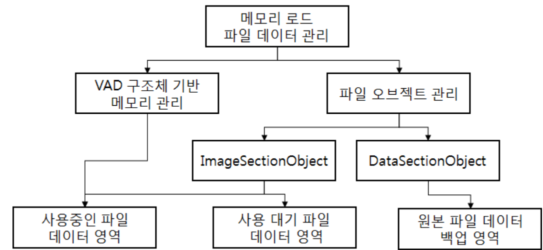

Fig. 3. Management of file data in physical memory

역과 동일하게 실행 프로세스에서 사용하지 않는 영역이다. 실행 중인 파일이 종료되는 경우 VAD 구조체에서 관리하는 파일 데이터 영역은 모두 초기화 되지만 파일 오브젝트에서 관리하는 파일 데이터 영역은 실행 파일이 종료되어도 초기화 되지 않는다. 하지만 파일 오브젝트에서 관리하는 파일이 디스크에서 삭제가 되는 경우 파일 오브젝트 정보와 파일 데이터 영역은 모두 초기화 된다.

물리 메모리에 저장된 실행 파일 데이터를 완벽하게 추출하기 위해서는 VAD 영역만을 추출하는 방법[5][6]이나 ImageSectionObject에서 관리하는 영역만을 추출하는 방법[7]이 아닌 VAD와 파일 오브젝트에서 관리하는 파일 데이터 영역을 통합하여 추출해야한다. 특히 파일 오브젝트에서 관리하는 파일 데이터 영역 중 실행 프로세스에서 사용하지 않는 영역은 원본 파일 데이터가 저장되어있기 때문에 해당 데이터는 실행 파일 복구에 필수적으로 요구된다.

3.2 파일 오브젝트 분석

3.2.1 ImageSectionObject 파일 데이터 관리

ImageSectionObject에서 관리하는 실행 파일 데이터는 PE 파일 포맷 구조 기반 4KB 단위로 분할되어 관리되기 때문에 메모리 페이지 테이블의 주소 인덱스와 파일 데이터 오프셋이 일대일 매칭 된다. ImageSectionObject에서 관리하는 파일 데이터 영역이 사용 중이거나 대기 중인 파일 데이터인 경우 해당파일 데이터가 저장된 페이지 테이블에는 유효한 주소 값이 저장되며, 사용 중이지 않는 경우 유효하지 않는 주소 값이 저장된다.

파일 데이터가 저장되어있는 시작 페이지 배열의 주소는 Fig 1의 DataSectionObject 포인터를 따라 연결된 _SUBSECTION 구조체의 멤버 변수

SubSectionBase 값에 명시된다. Fig 4 와 같이 SubSectionBase에 저장된 주소는 파일 데이터가 저장된 페이지 값의 첫 번째 배열의 위치를 나타낸다.

ImsageSectionObject에서 관리하는 메모리 페이지 테이블은 Fig 5와 같다. 페이지 테이블의 0 번째 주소(Page 0)에는 실행 파일 PE 구조에서 마지막 IMAGE_SECTION_HEADER 영역까지의 데이터가 저장된 메모리 주소가 저장된다.

PE 구조에서 SECTION 영역에 저장된 파일 데이터는 4KB 단위로 분할되어 저장되며, 파일 데이터가 저장된 주소 값은 메모리 페이지 테이블에 순차적으로 저장된다. PE 구조에서 2개 이상의 SECTION 영역이 존재하는 경우, 다음 SECTION 영역의 데이터는 새로 할당된 페이지에 저장된다. 그러므로 4KB 단위로 분할된 SECTION 영역의 마지막 데이터 크기가 4KB 미만인 경우 나머지 영역은 널 값으로 채워진다.

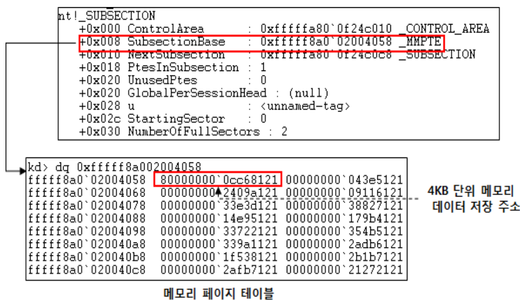

Fig. 4. _SUBSECTION Struct

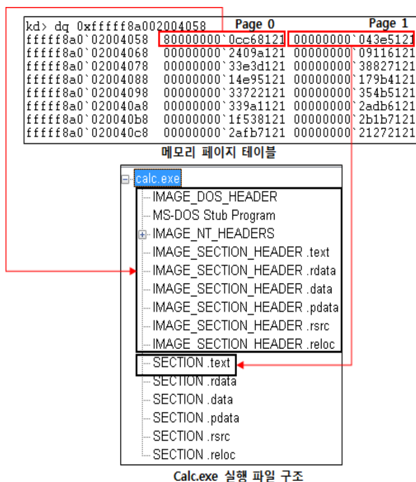

Fig. 5. ImageSectionObject memory page table

3.2.2 DataSectionObject 파일 데이터 관리

DataSectionObjet에서 관리하는 파일 데이터는 실행 파일 전체 데이터를 4KB 단위로 분할하여 관리하기 때문에 메모리 페이지 테이블의 주소 인덱스와 파일 데이터 오프셋이 일대일 매칭 된다. DataSectionObjet에서 관리하는 메모리 페이지 테이블에 유효한 주소 값이 저장된 경우 해당 파일 데이터는 메모리에 저장되어 있으며, 유효하지 않은 주소 값이 저장된 경우 해당 파일 데이터는 메모리에 저장되어있지 않은 상태이다.

파일 데이터가 저장되어있는 시작 페이지 배열의 주소는 Fig 1의 DataSectionObject 포인터를 따라 연결된 _SUBSECTION 구조체의 멤버 변수 SubSectionBase 값에 명시된다.

DataSectionObject에서 관리하는 메모리 페이지 테이블은 Fig 6과 같다. 메모리 페이지 테이블 0번째 주소(Page 0)에는 유효한 주소 값이 저장되어 있으므로 해당 파일 데이터 영역은 메모리에 저장되어있다. 하지만 메모리 페이지 테이블 1번째 주소(Page 1)에는 유효하지 않은 주소 값이 저장되어 있으므로 해당파일 데이터 영역은 메모리에 저장되어 있지 않는 것을 알 수 있다.

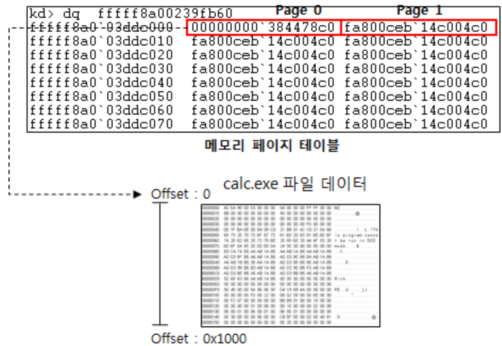

Fig. 6. DataSectionObject memory page table

3.3 메모리 저장 파일 데이터 분석

파일 오브젝트에서 관리하는 파일 데이터를 분석한 결과 ImageSectionObject에서 관리하는 파일 데이터는 프로그램 실행 과정에서 사용 중인 데이터와 사용 대기 중인 파일 데이터(원본 파일 데이터)가 혼합되어 관리되는 특징이 있다.

DataSectionObject에서 관리하는 파일 데이터의 경우 실행 초기 캐시 목적으로 파일 데이터를 관리하기 때문에 원본 파일 데이터만 관리하는 특징이 있다. 하지만 VAD에서 관리하는 파일 데이터는 프로그램 실행과정에서 사용 중인 파일 데이터이므로 프로그램 실행 과정에서 파일 데이터가 변경되어있을 가능성이 가장 높다고 할 수 있다.

물리 메모리에서 원본 파일 데이터를 추출하기 위해서는 원본 파일 데이터를 가장 많이 소유하고 있는 영역을 순차적으로 참조하여 추출해야한다. 따라서 DataSectionObject, ImageSectionObject, VAD에서 관리하는 파일 데이터를 순차적으로 참조하여 파일 데이터를 추출해야한다.

IV. 제안하는 실행 파일 추출 방법

물리 메모리에 저장된 실행 파일을 추출하기 위해 파일 오브젝트 분석 기반 실행 파일 (EXE, DLL) 데이터 추출 방법이 필요하다.

제안하는 방법에서는 물리 메모리에서 파일 오브젝트 관련 커널 구조체를 카빙 한다. 그러므로 실행 중인 프로세스에서 사용 중인 파일 뿐만 아니라 기존 실행 파일 추출 방법에서 추출할 수 없었던 종료된 실행 파일이나 캐시 목적으로 저장된 파일 데이터까지 추출할 수 있다.

4.1 파일 오브젝트 분석 기반 실행 파일 복구과정

파일 오브젝트 분석 기반 실행 파일 복구 과정은 Fig 7와 같다. 현재 실행중인 프로세스 정보 및 커널 구조체의 가상주소 변환을 위해 Volatility의 psscan과 동일한 방법인 풀 헤더 시그니처를 이용하여 _EPROCESS 구조체를 카빙 한다. 메모리에 저장된 파일 오브젝트 정보를 추출하기 위해 Volatility의 filescan과 동일한 방법인 풀 헤더 시그니처를 이용하여 _FILE_OBJECT 구조체를 카빙 한다. 카빙 된 _EPROCESS 구조체를 분석하여 VAD 정보를 추출하며, VAD 구조 정보 분석을 통해 실행 프로세스에서 사용 중인 실행 파일 정보를 추출한다. 카빙 된 _FILE_OBJECT 구조체를 이용하여 파일 경로 정보를 추출 하며, 내부 멤버 변수 FsContext를 이용하여 실제 파일 크기 정보가 저장된 _FSRTL_ADVANCED_FCB_HEADER 구조체 분석을 수행하여 파일 크기 정보를 추출한다.

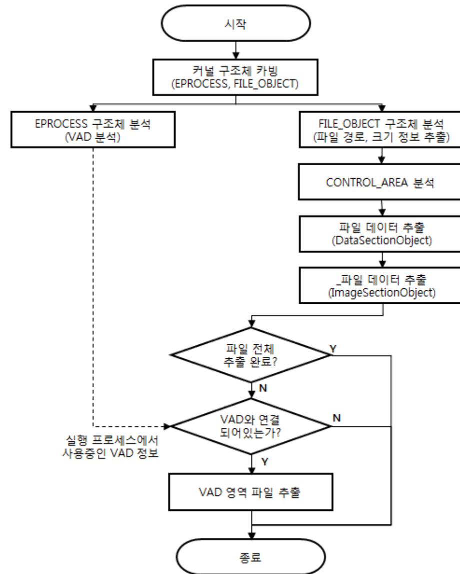

Fig. 7. Procedures for original file data extraction

FILE_OBJECT 구조체에 저장된 멤버 변수 SECTION_OBJECT 값을 참조하여 파일 데이터 섹션 객체 정보가 저장된 _CONTROL_AREA 구조체 정보를 추출한다.

_CONTROL_AREA 구조체 정보를 분석하여 ImageSectionObject, DataSectionObject에서 관리하는 메모리 페이지 테이블 정보를 추출 한다.

원본 파일 데이터를 추출하기 위해 가장 먼저 DataSectionObject에서 관리하는 메모리 페이지 테이블을 참조하여 추출가능한 모든 파일 데이터를 추출한다. DataSectionObject에서 추출하지 못한 파일 데이터 영역은 ImageSectionObject에서 관리하는 메모리 페이지 테이블을 참조하여 추출한다. 이후 파일 전체 데이터 추출이 완료된 경우 파일 추출 과정을 종료한다. 추출되지 않는 파일 데이터 영역이 있는 경우 최종적으로 VAD 구조체를 참조하여 해당파일 데이터 영역을 추출한다. VAD 구조체 분석과정에서도 파일 데이터를 추출하지 못한 경우 해당파일 영역은 메모리에 로드하지 않은 상태이므로 파일 추출과정에서 해당파일 영역을 널 값으로 채워 파일을 만든다.

V. 실험 및 결과

본 논문에서 제안하는 방법의 우수성을 검증하고자 Memory File Extractor(이하 MFExtractor)

를 개발하여 파일 추출 실험을 수행하였다.

실험 과정에서 추출된 파일과 원본 파일 데이터의 교는 MD5 해시 검사를 이용하였다. 실험은 윈도우 7 32/64비트 환경에서 진행하였으며, 기존 물리 메모리 실행 파일 추출 도구인 Volatility[5], Rekall[6] Memoryze[7]과 추출 결과를 비교하였다.

5.1 도구별 실행 파일 데이터 추출 결과

윈도우 7 32비트 시스템(x86)에서는 크기가 다른 3종의 물리 메모리를 사용하여 메모리 덤프를 수행하였으며, 64비트 시스템(x64)에서는 크기가 다른 5종의 물리 메모리를 사용하여 메모리 덤프를 수행하였다. 덤프 된 물리 메모리 데이터를 Volatility, Rekall, Memoryze, MFExtractor 도구를 이용하여 실행 파일 (EXE, DLL) 데이터를 추출하였다. 물리 메모리 크기별 도구에서 추출된 파일 결과는 Table 1과 같다. 실험 결과에서 전체 추출된 파일 개수는 도구를 이용하여 물리 메모리에 파일의 일부 데이터라도 추출 가능한 총 파일 개수이다. 원본 파일 추출 개수는 추출된 파일 중에서 실제 하드디스크에 저장된 파일과 동일한 파일 개수이다.

실행 파일 데이터 추출 결과 Volatility, Rekall 도구에서는 VAD 기반으로 추출하기 때문에 물리 메모리 크기에 따라 추출되는 파일 개수가 같은 것을 알 수 있다.

Memoyze 도구에서는 실행중인 파일을 대상으로 파일 오브젝트 내부 ImageSectionObject 정보로만 파일 정보를 추출하기 때문에 VAD 분석 방법으로 추출되는 결과와 거의 유사하게 추출되는 것을 알 수 있다. 하지만 ImageSectionObject에서는 실행 파일 데이터 추출에 중요한 PE 헤더 데이터를 관리하지 않는 경우가 있어 VAD 추출 결과에 비해 일부 실행 파일을 추출하지 못한 것을 알 수 있다.

MFExtractor 도구에서는 VAD와 파일 오브젝트에서 관리되는 파일 데이터를 이용하여 실행 프로세스에서 사용 중인 파일 데이터를 추출한다. 또한 종료되거나 캐시 목적으로 메모리에 저장된 파일 데이터의 경우 VAD 정보가 삭제가 되기 때문에 파일 오브젝트에서 관리하는 파일 정보만을 이용하여 파일 데이터를 추출한다. 따라서 MFExtractor 도구에서는 물리 메모리에서 기존 도구보다 더 많은 파일을 추출하며, 파일 데이터를 추출할 때 VAD에서 관리하는 파일 정보뿐만 아니라 파일 오브젝트 정보에서 관리 하는 정보까지 분석하여 추출하기 때문에 기존 도구에서 추출하지 못한 원본 파일도 추출할 수 있는 것을 알 수 있다.

Table 1 실험 결과에서 x64 시스템의 1GB 물리 메모리의 경우 기존 도구에서는 527개의 파일 데이터를 추출하였지만 제안한 방법을 이용하면 이보다 더 많은 835개의 파일 데이터를 추출하는 것을 알 수 있다. 추가로 추출된 308개의 파일은 물리 메모리에 캐시 목적으로 저장된 파일이거나 종료된 실행 파일이며, VAD 정보 없이 파일 오브젝트 정보만을 이용하여 관리되는 파일이기 때문에 Volatility나 Rekall 도구를 이용하여 추출할 수 없는 파일이다. Memoryze 도구에서는 파일 데이터를 추출하는 과정에서 VAD와 연결된 파일 오브젝트 내부 ImageSectionObject 값을 참조하여 파일 데이터를 추출하기 때문에 VAD 정보가 남아있지 않는 파일의 경우 파일 데이터를 추출할 수 없다. 그렇기 때문에 Memoryze 도구에서도 MFExtractor에서

Table 1. Result of the executable file data extraction in Window 7 32/64bit System

Memory Size		Volatility		Rekall		Memoryze		MFExtractor	
		Total file Count	Original file count	Total file Count	Original file count	Total file Count	Original file count	Total file Count	Original file count
x86	1G	521	0	521	0	519	0	550	2
	2G	549	0	549	0	540	0	841	5
	4G	541	0	541	0	539	0	843	6
x64	1G	527	0	527	0	527	0	835	2
	2G	526	0	526	0	525	0	947	6
	4G	527	0	527	0	527	0	1023	8
	8G	529	0	529	0	528	0	1037	9
	16G	534	0	534	0	534	0	1038	9

추가로 추출된 파일을 추출하지 못한 것을 알 수 있다.x86 시스템과 x64 시스템에서 물리 메모리 크기가 커질수록 MFExtractor를 이용하여 추출한 파일 개수는 점차 증가하는 것을 알 수 있다. 이를 통해 물리 메모리 크기가 커질 수로 메모리에 저장되는 파일 개수가 더 많아지는 것을 알 수 있다.

5.2 메모리 크기별 파일 데이터 저장 영역 분석

본 논문에서 제안한 방법을 기반으로 물리 메모리 크기에 따라 실행 파일 데이터 추출을 수행한 결과는 Table 1과 같다. 추출되는 전체 파일 개수에 비해 아주 적은 파일만 원본 파일 데이터가 추출되는 것을 알 수 있다. 그 원인을 분석하기 위해 물리 메모리의 크기에 따라서 추출되는 개별 파일에 대해 VAD, ImageSectionObejct, DataSectionObject에서 관리하는 파일 데이터 크기 정보를 추출하였고, 그 결과는 Table 2와 같다.

물리 메모리 크기가 1GB인에서 x86 시스템에서 추출된 전체 파일 개수는 550개이며, VAD에서 관리하는 파일 데이터는 크기는 58MB이다. VAD에서 관리 가능한 전체 파일 데이터 크기는 58MB이며, 전체 영역에서 약 18%만 관리되는 것을 알 수 있다. 또한 x86, x64 시스템 상관없이 물리 메모리 크기에 따라 VAD에서 관리하는 파일 데이터 크기 비율은 크게 변동되지 않는 것을 알 수 있다. 이를 통해 실행중인 프로그램 파일에서는 전체 파일 데이터의 약 19%만 사용하고 있는 것을 알 수 있다.

실행 파일에서 사용 중인 파일 데이터와 사용 대기 중인 파일 데이터를 관리하는 ImageSectionObject에서는 전체 파일 데이터 중에서 평균 24%정도만 관리하며 VAD와 동일하게 물리 메모리 크기에 따라 관리하는 파일 비율은 크게 변동되지 않는 것을 알 수 있다.

DataSectionObject에서는 원본 파일 데이터만을 관리하기 때문에 실행 파일 복구에 있어 중요한 정보 이다. 하지만 DataSectionObject에서는 관리 하는 물리 메모리 파일 데이터의 경우 물리 메모리 크기가 1GB인 x86 시스템에서는 최대 69MB 파일 데이터를 물리 메모리 크기가 1GB인 x64 시스템에서는 최대 30MB 파일 데이터만을 관리하는 것을 알 수 있다. 동일한 시스템에서 물리 메모리 크기가 커져도 DataSectionObject에서 관리 가능한 최대 파일 데이터 크기 증감은 크지 않는 것을 알 수 있다.

물리 메모리에서 파일 데이터 전체를 추출하기 위해서는 전체 파일 데이터가 메모리에 저장되어야한다. 하지만 실험 결과 Table2와 같이 VAD, ImageSectionObject에서는 물리 메모리 크기와 상관없이 관리 가능한 전체 파일 데이터에서 약 21% 정도의 파일 데이터만 메모리에 저장하고 있다. 또한 원본 파일 데이터를 관리하는 DataSectionObject에서는 VAD나 ImageSectionObject에서 관리하는 파일 데이터에 비해 관리 가능한 파일 데이터 크기가 작으며 실제로 저장되어있는 비중도 매우 적은 것을 알 수 있다. 따라서 일반적인 환경에서는 물리 메모리에 실행 파일 전체 데이터가 저장되어있지 않기 때문에 물리 메모리만을 이용해서는 전체 파일 데이터를 추출하는 것이 어렵다는 것을 알 수 있다.

5.3 추출된 파일 데이터 분석

본 논문에서 제안하는 방법으로 추출한 실행 파일 데이터와 기존 도구에서 추출된 파일 데이터에 대한

Table 2. Result of the file storage size in Window 7 32/64bit System

Memory Size		Total file count	VAD File data(MB)	ImageSectionObject File data (MB)	DataSectionObject File data (MB)
x86	1G	550	58/313 (18%)	104/331 (31%)	1.6/69 (2%)
	2G	871	80/392 (20%)	164/645 (25%)	2.8/87 (3%)
	4G	843	79/379 (20%)	180/579 (31%)	23/87 (26%)
x64	1G	835	75/407 (18%)	125/538 (23%)	1.57/30 (5%)
	2G	947	133/654 (20%)	211/930 (22%)	15.42/227 (6%)
	4G	1023	136/668 (20%)	203/991 (20%)	20/224 (8%)
	8G	1037	133/647 (20%)	201/996 (20%)	25/263 (9%)
	16G	1038	137/645 (21%)	209/993 (21%)	26/270 (9%)

비교 분석을 하였다. 비교 분석 대상 파일은 윈도우 운영체제에 기본적으로 설치되어있으며 일반적인 시스템에서 실행 가능한 calc.exe(윈도우 계산기) 프로세스를 대상으로 하였다.

5.3.1 변경된 파일 데이터 영역을 추출한 경우

Volatility와 Rekall 도구에서는 VAD를 이용한 동일한 방법으로 파일 데이터를 추출하기 때문에 본 분석에서는 Volatility에서 추출된 파일 데이터를 이용하여 비교 분석하였다. Fig 8은 Volatility 도구와 MFExtractor에서 추출한 파일 데이터를 비교한 것이다.

Volatility에서는 VAD 기반으로 현재 사용 중인 파일 데이터 영역을 추출하기 때문에 파일 데이터 오프셋 0x72400 영역을 원본 파일과 다르게 추출할 것을 알 수 있다.

calc.exe 파일 오프셋 0x72400은 .data 영역(데이터 영역)으로서 전역 변수 값이 저장되며 프로그램 실행과정에서 변경 될 수 있는 영역이다. 하지만 본 논문에서 제안한 파일 추출 도구에서는 VAD 뿐만 아니라 ImageSectionObject, DataSectionObject를 분석하여 파일 데이터를 추출하기 때문에 기존 도구와 다르게 해당 영역의 원본 파일 데이터를 추출한 것을 알 수 있다.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000723F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072400	0C	1A	BB	FA	FE	07	00	00	00	00	00	00	00	00	00	00
00072410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072420	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072430	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072440	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072450	01	00	00	01	2E	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF
00072460	03	00	00	00	20	00	00	00	0A	00	00	00	3D	01	00	00
00072470	3A	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Volatility 도구 : calc.exe 추출 바이너리

추출된 파일 데이터가 다른 부분

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000723F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072400	54	C0	01	00	01	00	00	00	00	00	00	00	00	00	00	00
00072410	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072420	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072430	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00072440	FF	FF	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00
00072450	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF
00072460	03	00	00	00	20	00	00	00	0A	00	00	00	3D	01	00	00
00072470	3A	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00

MFExtractor 도구 : calc.exe 추출 바이너리 (원본 파일 데이터)

Fig. 8. Extract file data comparison

5.3.2 파일 데이터 영역을 추출하지 못한 경우

Fig 9는 Memoryze 도구와 MFExtractor에서 추출한 파일 데이터를 비교한 것이다.

ImageSectionObject 기반 실행 파일 데이터를 추출하는 Memoryze에서는 파일 데이터 오프셋 0x26600 영역을 추출하지 못해 널 값을 채운 것을 알 수 있다. calc.exe 파일 오프셋 0x26600은 .text 영역(코드 영역)으로서 ImageSectionObject에서 위 파일 데이터 영역을 관리하지 않기 때문에 해당파일 데이터 영역을 추출 하지 못한 것을 알 수 있다. 하지만 본 논문에서 제안한 파일 추출 도구에서는 ImageSectionObject 뿐만 아니라 VAD, DataSectionObject를 분석하여 파일 데이터를 추출하기 때문에 기존 도구와 다르게 해당 영역의 원본 파일 데이터를 추출한 것을 알 수 있다.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000265E0	C4	20	41	5D	41	5C	5F	C3	90	90	90	90	90	90	90	90
000265F0	48	8B	C4	48	89	48	08	57	48	83	EC	40	48	C7	44	24
00026600	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00026610	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00026620	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00026630	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00026640	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00026650	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00026660	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Memoryze 도구 : calc.exe 추출 바이너리

파일 데이터를 추출하지 못한 영역

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000265E0	C4	20	41	5D	41	5C	5F	C3	90	90	90	90	90	90	90	90
000265F0	48	8B	C4	48	89	48	08	57	48	83	EC	40	48	C7	44	24
00026600	30	FE	FF	FF	FF	48	89	58	18	48	89	70	20	49	8B	F0
00026610	48	8B	FA	48	8B	D9	83	60	10	00	48	8B	02	48	8B	48
00026620	E8	48	85	C9	0F	84	9C	B7	00	00	48	8B	01	FF	50	20
00026630	4C	8B	D8	EB	00	4D	85	DB	0F	84	90	B7	00	00	33	C9
00026640	4D	85	DB	0F	95	C1	85	C9	0F	84	99	B7	00	00	49	8B
00026650	03	49	8B	CB	FF	50	18	48	83	C0	18	48	89	03	C7	44
00026660	24	58	01	00	00	00	4C	8B	0E	41	8B	41	F0	48	8B	17

MFExtractor 도구 : calc.exe 추출 바이너리 (원본 파일 데이터)

Fig. 9.Extract file data comparison

VI. 결 론

본 논문에서는 윈도우 물리 메모리에 저장되는 파일 정보를 관리하는 파일 오브젝트 커널 구조를 분석하고, 물리 메모리에 크기에 따른 파일 데이터 저장 용량 특징을 분석하였다. 또한 파일 오브젝트 분석을 기반으로 물리 메모리에 저장된 원본 파일 데이터를 효과적으로 추출하는 방법을 제안하였으며 이를 수행하는 프레임워크를 개발하였다.

제안하는 실행 파일 추출 방법을 이용하면 기존 방법보다 더 많은 파일 정보 수집이 가능하며, 원본

파일 데이터를 효율적으로 추출할 수 있다는 장점이 있다. 하지만 특정 파일 데이터가 메모리 페이징이 되거나 현재 실행 프로그램에서 사용하지 않아 메모리에 저장되지 않는 경우 해당파일 데이터를 추출할 수 없다는 문제점이 있다. 향후 연구에서는 윈도우 절전 모드 파일에 저장된 물리 메모리 데이터 정보를 분석 할 예정이다.

References

〔1〕 Matthieu Suiche, ″Windows Hibernation File for Fun ′N′ Profit,″ BlackHat USA, Aug. 2008.

〔2〕 James Butler, Justin Murdock, ″Physical Memory Forensics for Files and Cache,″ Defcon 19, July. 2011.

〔3〕 Mark E. Russinovich, David Solomon,Alex Ionescu ″Windows Internal 5th edition,″ Acorn, pp. 956-962, Jul. 2010.

〔4〕 Brendan Dolan-Gavitt, “The VAD tree: A process-eye view of physical memory,” DIGITAL INVESTIGATION 4S, pp. 62-64, Sep. 2007.

〔5〕 Volatility procexedump,https://code.google.com/p/volatility/wiki/CommandReference23#procexedump

〔6〕 Rekall, http://www.rekall-forensic.com/docs/Manual/Plugins/Windows/PEDump.html

〔7〕 Mandiant Memoryze, https://www.mandiant.com/resources/download/memoryze

〈저자소개〉

강 영 복 (Youngbok Kang) 학생회원
2012년 2월: 전남대학교 전자컴퓨터공학부(공학사)
2014년 2월: 전남대학교 정보보안협동과정 졸업 (이학석사)
2014년 2월~현재: 전남대학교 정보보안협동과정 박사과정
〈관심분야〉 디지털 포렌식, 악성코드 탐지, 취약점 분석

사 진

황 현 욱 (Hyunuk Hwang) 정회원
2000년 2월: 조선대학교 정보통신공학과 졸업(공학사)
2002년 2월: 조선대학교 전자공학과 졸업(공학석사)
2004년 8월: 전남대학교 정보보호협동과정 졸업(이학박사)
2004년 9월~현재: ETRI 부설연구소 선임연구원
〈관심분야〉 디지털 포렌식, 악성코드, 사이버보안

사 진

김 기 범 (Kibom Kim) 정회원
1994년 2월: 제주대학교 정보공학과 졸업(공학사)
1996년 8월: 고려대학교 전산과학과 졸업(이학석사)
2001년 2월: 고려대학교 전산과학과 졸업(이학박사)
2004년 1월~2004년 7월: (주) 이씨오 개발부장
2004년 8월~현재: ETRI 부설연구소 책임연구원
〈관심분야〉 디지털 포렌식, 사이버보안, 정보보호

노 봉 남 (Bongnam Noh) 종신회원
1987년: 전남대학교 수학교육과 (이학사)
1982년: KAIST 전산학과 (이학석사)
1994년: 전북대학교 전산과 (이학박사)
1983년~현재: 전남대학교 전자컴퓨터공학부 교수
2000년~현재: 시스템보안연구센터 소장
〈관심분야〉 디지털 포렌식, 시스템 및 네트워크 보안, 정보사회와 사이버 윤리