情報保護學會論文誌
第 21 卷 第 2 號, 2011. 4

라이브 포렌식을 위한 윈도우즈 물리 메모리 분석 도구*

한 지 성† , 이 상 진‡
고려대학교 디지털포렌식연구센터

The Windows Physical Memory Dump Explorer for Live Forensics*

Jisung Han† , Sangjin Lee‡
Digital Forensic Research Center, Korea University

요 약

라이브 포렌식은 하드디스크 파일시스템 분석으로 획득할 수 없는 메모리 내의 활성 데이터를 얻을 수 있다는 장점으로 인해 최근의 포렌식 조사 시 활용되고 있다. 하지만 기존의 라이브 포렌식은 활성 시스템에서 시스템 정보를 획득하기 위한 명령어 기반의 도구를 사용함으로써, 악성코드에 의한 변조된 결과 획득 및 재분석이 용이하지 못한 단점을 가지고 있다. 따라서 본 논문은 시스템 조사 도구를 이용한 라이브 포렌식의 단점을 보완하기 위한 윈도우즈 커널 객체 구조 설명 및 분석 방법을 설명한다. 또한, 이를 활용하기 위한 도구를 설계 및 구현하였고, 실험 결과를 통해 그 효과를 입증한다.

ABSTRACT

Live data in physical memory can be acquired by live forensics but not by harddisk file-system analysis. Therefore, in case of forensic investigation, live forensics is widely used these days. But, existing live forensic methods, that use command line tools in live system, have many weaknesses; for instance, it is not easy to re-analyze and results can be modified by malicious code. For these reasons, in this paper we explain the Windows kernel architecture and how to analyze physical memory dump files to complement weaknesses of traditional live forensics. And then, we design and implement the Physical Memory Dump Explorer, and prove the effectiveness of our tool through test results.

Keywords: digital forensics, live forensics, live data, physical memory, windows memory analysis, kernel objects

I. 서 론

컴퓨터 기술이 발전함에 따라 대부분의 정보가 디지털 형태로 저장되고 있기 때문에, 디지털 정보에 쉽게 접근하기 위한 PC 및 스마트폰 등의 디지털 기기의 보급 및 사용이 증가하고 있다. 하지만 정보 접근의 용이성이 쉬워진 만큼 디지털 관련 범죄 역시 급속도로 증가하는 부작용을 낳고 있다. 따라서, 디지털 관련 범죄의 증가에 따른 디지털 증거의 중요성이 대두됨에 따라 디지털 포렌식(digital forensics) 분야의 발전이 가속화되고 있다.

전통적인 디지털 포렌식 방법론에 의하면, 포렌식 조사관은 우선 조사 대상 시스템의 전원을 내리고 디스크 이미징을 통해 디스크 사본을 획득한 후, 이를 사후 분석하여 디지털 증거를 찾는다[1][2]. 대부분의 경우 디지털 증거는 하드디스크나 USB 메모리에 저장되어 있는 비휘발성 데이터일 가능성이 크기 때문에, 디스크 분석만으로도 유효한 주요 증거를 찾을 수

접수일(2010년 8월 10일), 게재확정일(2010년 9월 28일)
* 본 연구는 한국연구재단을 통해 교육과학기술부의 바이오 연구개발사업으로부터 지원받아 수행되었습니다. (20100020634)
† 주저자, xnetblue@korea.ac.kr
‡ 교신저자, sangjin@korea.ac.kr

있다. 하지만 이러한 분석 방법 하에서는 시스템의 전원을 차단하는 순간 중요한 휘발성 증거들을 잃는다[3].

휘발성 데이터는 시스템의 전원을 차단하는 순간 사라지는 데이터를 의미하며, 이는 레지스터, 캐쉬, RAM 등의 휘발성 매체에 저장되어 있다. 이러한 정보를 얻기 위해서는 시스템의 전원을 차단하기 전에 시스템 정보를 얻기 위한 명령어 및 도구를 이용한다. 이렇게 디스크 분석으로는 획득할 수 없는 활성 데이터를 수집하고 이를 분석하는 것을 라이브 포렌식(live forensics)이라 한다.

라이브 포렌식은 기존의 디스크 포렌식으로는 획득할 수 없는 네트워크 정보, 실행 중인 프로세스, 열려있는 파일, 커널 상태 등의 정보들을 수집 및 분석할 수 있다. 그리고 이러한 데이터의 크기가 디스크에 담긴 데이터에 비해 상대적으로 작기 때문에 수집하는데 적은 시간이 소요된다. 또한 웹, 데이터베이스, 메일 서버 시스템 등의 침해사고 대응 시에도 사업 연속성을 보장하기 때문에[2] 비용적인 측면에서도 장점을 가진다.

본 논문은 라이브 포렌식 관점에서 윈도우즈 물리 메모리 덤프 파일을 분석하고, 이를 이용하여 시스템이 활성화 되어있을 때의 프로세스, DLL(동적 링크 라이브러리), 핸들 등의 상태를 보여주는 도구의 설계 및 그 결과물을 보여준다. 2장에서는 기존의 라이브 포렌식 방법과 물리 메모리 분석 방법의 비교를 통해, 물리 메모리 분석의 장점을 보여주고, 3장에서 윈도우즈 계열 OS의 커널 구조 및 물리 메모리 분석 기법을 상세히 설명한다. 4장에서는 물리 메모리 분석 도구의 설계 및 그 결과물을 통한 분석 결과를 설명한다. 마지막으로 5장에서 결론을 맺는다.

[표 1] 활성 데이터 조사 명령어

명령어	활성 데이터
date /t & time /t	시스템 날짜 및 시간
psfile.exe	열려있는 파일
netstat.exe	네트워크 연결
tasklist.exe pslist.exe	프로세스 정보
listdlls.exe	프로세스가 사용하는 모듈 및 DLL
handle.exe	프로세스가 사용하는 핸들
fport.exe	프로세스-포트 매핑

II. 관련 연구

2.1 물리 메모리 분석의 장점

기존의 일반적인 라이브 포렌식 조사는 [표 1][4]과 같은 명령어 기반 인터페이스 도구들을 개별적으로 실행하거나 쉘 스크립트를 이용하여 일괄적으로 실행하여 결과를 취합하는 방식을 이용하였다. 이는 일반적으로 시스템 관리를 위해 사용하는 도구를 이용하여 신속하게 휘발성 정보들을 수집할 수 있다는 장점이 있다.

하지만 최근에는 PCI 컨트롤러나 IEEE 1394 포트를 통한 하드웨어 기반 물리 메모리 수집[5][6], mdd[7], win32dd[8] 등을 이용한 소프트웨어 기반 물리 메모리 수집과 수집된 물리 메모리를 대상으로 하는 라이브 포렌식 분석에 관한 연구가 활발히 진행 중이다.

다음 절은 기존의 명령어 도구를 이용한 활성 데이터 분석에 비해 물리 메모리 분석이 가질 수 있는 우수한 장점들을 설명한다.

2.1.1 시스템 변화 최소화

조사자가 프로세스를 실행할 때 마다 새로 실행된 프로세스가 물리 메모리에 로드되면서 기존의 메모리 영역이 덮어 쓰여지거나 이미 실행 중인 프로세스가 페이지 파일로 페이지아웃 되는 등의 상당한 시스템 변화가 일어난다[9]. 즉, 라이브 포렌식은 조사자의 조사 행위를 최소화 하는 것이 매우 중요하다.

기존의 라이브 포렌식 조사 방법은 각각의 활성 데이터를 조사하기 위해서 다수의 프로세스 실행을 유발하기 때문에 조사 직전에 물리 메모리에 존재하던 활성 데이터들이 손상될 가능성이 크다. 따라서 활성 데이터의 손상을 최소화하기 위해서는 하드웨어 기반의 물리 메모리 수집을 하거나 최소한의 프로세스 실행만으로 수집할 수 있는 소프트웨어 기반의 물리 메모리 수집을 수행하여 이를 분석하는 것이 최선의 방법이다.

2.1.2 악성코드에 의해 변조된 시스템 정보 수집 우회

대부분의 악성코드는 DLL 인젝션, SSDT 후킹, 커널 객체 직접 수정(DKOM) 등을 통해 시스템 정보를 은닉/변조하여 조사자의 활성 데이터 수집을 방해한다[10]. 기존의 라이브 포렌식 조사 도구들은 OS

에서 제공하는 API 함수를 이용해 시스템 정보를 열람하는데, 만약 대상 시스템이 악성코드에 의해 침해되었을 경우 악성코드가 악의적으로 변조한 함수의 실행 결과를 출력할 가능성이 크기 때문에 정확한 라이브 포렌식 조사가 힘들다.

물리 메모리 분석은 함수의 흐름과 관계없이 실제로 존재하는 커널 객체들을 분석하고 추출하여 의미있는 정보를 도출해내는 방식이다. 따라서 악성코드에 의해 API 함수의 흐름이 변조되었다 하더라도, 분석 결과에는 영향을 끼치지 않는다.

2.1.3 재분석 용이

활성 데이터는 시간의 흐름에 따라 지속적으로 변화되고 전원이 차단되면 모두 소멸되기 때문에, 기존 도구를 통해 수집한 활성 데이터로부터 다시 새로운 결과를 도출해 내는 것은 불가능하다. 하지만 물리 메모리 덤프 파일은 수집 당시의 물리 메모리에 의한 사본이기 때문에 언제든지 재분석이 가능하여 추가적인 분석이 필요할 경우 지속적으로 활용될 수 있다.

2.2 기존의 물리 메모리 분석 도구

2004년에 발표된 Brian D. Carrier의 연구[11]에 의해 수집된 물리 메모리로부터 활성 데이터를 추출할 수 있음이 증명되었고, 그 다음 해에 개최된 Digital Forensic Research Workshop (DFRWS 2005)의 메모리 분석 챌린지부터 물리 메모리 분석 연구가 빠르게 진행되기 시작하였다. 그 이후 다수의 물리 메모리 분석 도구들이 개발되었는데, 다음은 그 도구들의 종류 및 특징을 설명한다.

- **MemParser**

MemParser[12]는 물리 메모리 덤프 파일로부터 프로세스 목록, DLL 목록, 프로세스 실행 환경, 문자열 추출 등의 기능을 가진 콘솔 도구이다. 윈도우즈 2000만을 대상으로 하기 때문에 지원 환경이 제한적이며, 콘솔 명령어 기반이기 때문에 사용하기 불편하다.

- **PtFinder**

PtFinder[13]는 물리 메모리 덤프 파일로부터 프로세스 및 쓰레드 정보를 탐색하는 도구이다. MemParser에 비해서 다양한 운영체제 버전(윈도우즈 2000, XP, 2003, Vista)을 지원한다. 하지만 perl 스크립트로 작성되어 있어 분석 시스템에 perl 인터프리터를 설치해야하며, 분석 대상 시스템에 설치된 서비스팩 버전에 따라 지원하지 않는 경우가 있다. 또한 프로세스 및 쓰레드 정보만을 출력하기 때문에 분석할 수 있는 정보에 상당한 제약이 따른다.

- **Volatility**

Volatility[14]는 python 스크립트로 작성된 물리 메모리 분석 도구이다. 프로세스, 쓰레드, DLL 목록, 프로세스가 사용 중인 파일, 네트워크 세션 등 가장 많은 정보를 제공한다. 또한 플러그인을 지원하여, 기존 코드를 재사용하여 새로운 기능을 추가할 수 있다. 하지만 실행을 위해서는 python 인터프리터가 필요하며, 콘솔 기반의 분석을 해야 하기 때문에 사용이 불편하다. 또한 윈도우즈 XP만을 지원하기 때문에 대상 범위가 좁다는 것이 단점이다.

III. 윈도우즈 물리 메모리 분석

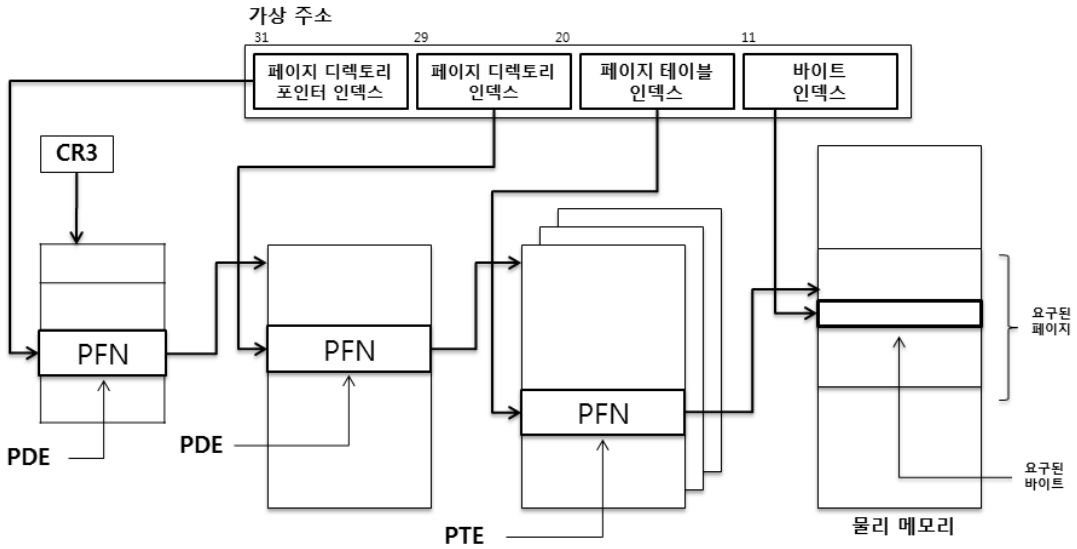
본 장은 윈도우즈 커널 구조를 설명하고, 이를 분석하기 위한 기법을 소개한다. 또한 의도적인 은닉 프로세스가 존재할 때, 이를 탐지하기 위한 기법도 설명한다. 분석 대상은 인텔 x86 CPU 기반의 윈도우즈 XP, Vista, 7 및 server 2003, 2008 등의 서버 제품군이다.

3.1 윈도우즈 커널 구조

3.1.1 커널 및 사용자 주소 공간

윈도우즈는 시스템 메모리를 관리하기 위해 가상 주소를 사용한다. 그리고 커널 및 사용자 프로세스가 접근할 수 있는 가상 주소의 집합을 가상 주소 공간이라고 한다. 가상 주소 공간은 두 개의 영역으로 나뉘는데 커널 및 커널 모드 드라이버에 의해 사용되는 커널 주소 공간과, 사용자 프로세스 및 DLL에 의해 사용되는 사용자 주소 공간이 있다.

커널 주소 공간에는 운영체제 커널 및 드라이버가 위치해 있으며, 커널 모드에 진입한 개체만이 접근 가능하다. 사용자 주소 공간에는 프로세스들이 위치해 있으며, 각각의 프로세스는 자신만의 주소 공간을 갖는다. 즉, 프로세스는 커널 주소 공간에 접근할 수 없으며, 다른 프로세스 주소 공간에 읽기 및 쓰기 역시 할 수 없다[15][16].



〔그림 1〕 가상-물리 주소 변환 과정 (PAE)

각 주소 공간은 〔그림 1〕과 같이 가상 주소를 비트 단위의 인덱스로 사용하여, CR3 레지스터로부터 참조 가능한 페이지 디렉토리, 페이지 테이블, 요청한 페이지 순으로 접근하여 물리 메모리 및 페이지 파일에서의 물리적 위치를 알 수 있다〔17〕. 모든 개체는 주소 공간을 표현함에 있어서 가상 주소를 사용하기 때문에, 물리 메모리를 분석하기 위해서는 가상-물리 주소 변환이 반드시 필요하다.

3.1.2 메모리 할당

윈도우즈는 시스템 메모리를 관리하기 위해 메모리 영역을 블록 단위로 나누어 관리하는데 이를 페이지(page)라 한다. 페이지는 활성화되어 물리 메모리에 존재하거나, 비활성화 되어 페이지 파일 등의 형태로 디스크에 저장되어 있을 수 있다. 각 페이지는 일반적으로 4096바이트의 크기를 가진다.

커널은 성능 및 용량의 효율성을 위해서 활성화 된 페이지를 미리 할당해놓고, 사용자 및 커널 영역에서 메모리 할당 요청이 있을 때마다 페이지 풀 내에서 요청한 용량만큼을 재할당 해주게 된다. 이때 할당된 메모리 풀마다 POOL_HEADER라는 8바이트 크기의 구조체가 존재하는데, 구조체 내에는 메모리 풀의 크기 및 태그를 나타내는 필드가 있다〔18〕. 풀 태그는 메모리의 사용 용도마다 4바이트 크기의 아스키 문자가 지정되는데, 예를 들어 프로세스 객체의 경우 "Proc", 스레드는 "Thre"를 나타낸다. POOL_HEADER 구조체의 태그는 메모리 분석을 위한 카빙 기법 적용 시, 커널 객체의 시그니처로써 활용될 수 있다.

3.1.3 커널 객체

커널은 운영체제 구성요소를 객체 단위로 관리한다. 즉, 프로세스, 스레드, DLL, 파일 등은 각자의 객체를 가지며, 커널 영역에 위치하게 된다. 각 객체는 특유의 구조체 형태를 가지며, DDK(Driver Development Kit)내의 헤더파일 및 WinDBG의 dt명령어로 확인해 볼 수 있다.

커널 객체는 커널 주소 공간에 존재하기 때문에 사용자 영역에서 직접 접근은 불가능하며, 핸들(handle)이라는 인터페이스를 통해서 접근 가능하다. 각 프로세스는 자신이 사용하는 객체들을 핸들 테이블(handle table)로 관리하는데, 이를 통해 자식 프로세스, 스레드 혹은 사용 중인 파일 등을 획득할 수 있다.

3.2 물리 메모리 분석 기법

앞 절에서 살펴본 커널의 구조를 토대로 커널의 물리 메모리를 분석하기 위한 기법들이 다수 존재한다. 물리 메모리 분석 도구를 설계 및 개발하기 위해 적용할 수 있는 물리 메모리 분석 기법을 설명한다.

3.2.1 문자열 검색 및 추출 기법

메모리에는 포렌식 조사 시 매우 유용하게 사용될 수 있는 데이터들이 다수 존재하는데, 사용자 ID, 패스워드, 메신저 대화 내역, 키보드 입력문자, 접근한 URL, 텍스트 문서 등의 민감한 정보가 발견될 수 있다. 즉, 시스템 사용자의 특징을 잘 표현할 수 있으며, 대부분이 텍스트 형태로 남아 조사관이 쉽게 식별할 수 있다는 점에서 매우 중요하다.

만약 하드디스크에서 암호화 된 문서 및 압축파일이 발견되고 패스워드 및 키를 전수조사하여 복호화해야 하는 경우, 패스워드 문자열 및 암호화 키의 길이가 길고 영문자, 숫자, 특수 문자 등이 조합되었을 때에는 허용된 시간 안에 찾아내는 것은 매우 힘들다.

하지만 기존 연구에 의하면, 768MB 크기의 페이지 파일에서 텍스트를 추출해 본 결과, 약 43%의 확률로 패스워드를 찾을 수 있었다[19]. 즉, 페이지 파일이 물리 메모리에서 비활성화 되어 페이지 아웃된 데이터의 집합임을 생각해 보았을 때, 물리 메모리에서도 매우 높은 확률로 사용자의 패스워드를 찾을 수 있을 것이다.

3.2.2 커널 객체 구조 분석 기법

▪ Directory Table Base 탐색

디렉토리 테이블 베이스는 3.1.2절의 [그림 1]과 같이 가상 주소를 물리 주소로 변환하기 위해서 반드시 필요하다. CR3 레지스터에는 현재 실행 중인 스레드에 대한 프로세스의 디렉토리 테이블 베이스가 저장되어있으며, 그 이외의 다른 프로세스들을 위해 각 프로세스 객체인 EPROCESS 구조체 안의 KPROCESS. DirectoryTableBase 필드에도 존재한다. 즉, 각 프로세스가 독립적인 주소 공간을 가지기 위해서 디렉토리 테이블 베이스의 값이 각자 다르게 저장되어 있으며, 문맥교환(context switching) 시 CR3 레지스터에 로드된다.

커널 객체들을 분석하기 위해서는 커널 객체들이 존재하는 커널 주소 공간을 물리 주소로 변환할 수 있어야 한다. 커널 주소 공간을 표현하기 위한 디렉토리 테이블 베이스는 "Idle" 가상 프로세스의 EPROCESS의 필드로서 존재한다. 즉, 디렉토리 테이블 베이스를 찾기 위해서는 EPROCESS 구조체를 탐색해야 하는데, 이를 위한 방법은 3.2.3절에서 상세히 설명한다.

▪ 커널 전역 변수

커널 전역 변수들은 문서화 되지 않은 KDDEBUGGER_DATA64 구조체에 존재한다. 이 구조체에는 EPROCESS의 원형 이중 연결 리스트의 시작 주소인 PsActiveProcessHead, 로드된 드라이버 모듈의 원형 이중 연결 리스트의 시작 주소인 PsLoadedModuleList 등의 중요한 커널 변수들이 존재한다. KDDEBUGGER_DATA64 구조체의 주소는 KPCR.KdVersionBlock → DBGKD_GET_VERSION64.DebuggerDataList 필드를 순서대로 참조함으로써 얻을 수 있다. 즉, KPCR을 찾아야 주요 커널 전역 변수들을 얻을 수 있다.

KPCR 구조체는 각 프로세서의 정보 및 커널 디버거에 의해 보이는 값들이 저장되어있다[20]. KPCR 구조체의 SelfPcr 필드에는 KPCR 구조체 자신의 주소가 저장되어 있으며, Prcb 필드에는 KPCRB 구조체의 주소가 저장되어 있다. KPCRB 구조체는 항상 KPCR 구조체의 시작 주소로부터 120바이트 뒤에 존재하기 때문에 (수식 1)의 조건을 만족한다. 즉, (수식 1)의 조건을 이용하여 KPCR 구조체를 찾으면

[표 2] PEB 구조체 주요 필드

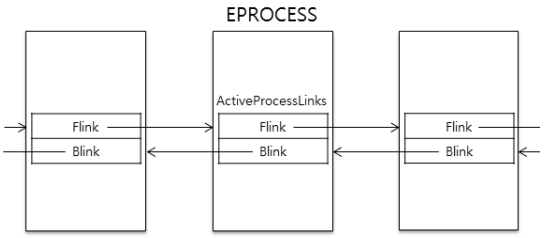
필드명	오프셋	자료형	의미
ImageBaseAddress	0x08	Ptr32 Void	실행 이미지의 베이스 주소
Ldr	0x0C	Ptr32 _PEB_LDR_DATA	로드된 DLL
ProcessParameters	0x10	Ptr32 _RTL_USER_PROCESS_PARAMETERS	실행 시 매개변수 (실행 경로, 명령어줄, 윈도우 이름, 현재 경로 등)
OSMajorVersion	0xA4	Uint4B	OS 메이저 버전
OSMinorVersion	0xA8	Uint4B	OS 마이너 버전
OSBuildNumber	0xAC	Uint4B	OS 빌드 넘버
OSCSDVersion	0xAE	Uint4B	OS 서비스팩 버전

커널 전역 변수들을 찾을 수 있다.

$$x \leftarrow \text{KPCR.SelfPcr}$$
$$y \leftarrow \text{KPCR.Pcrb}$$
$$y = x + 120_{(16)}$$

(수식 1) KPCR 필드의 필수 조건

참고로, 윈도우즈 XP에서 KPCR 구조체는 가상 주소 0xFFDFF000에 항상 존재하기 때문에 해당 주소로 직접 접근이 가능하지만, 비스타 이상의 버전에서는 재부팅 시마다 다른 위치에 로드되기 때문에 반드시 탐색 과정을 거쳐야 한다.



〔그림 2〕 이중 연결 리스트로 연결 된 EPROCESS 구조체

▪ 실행 프로세스

윈도우즈 커널은 프로세스를 관리하기 위해 〔그림 2〕와 같이 원형 이중 연결 리스트로 EPROCESS 구조체를 관리하고 있다. 현재 실행 중인 프로세스의 목록은 커널 변수인 PsActiveProcessHead로부터 링크를 따라가면서 EPROCESS를 나열하면 모두 얻을 수 있다〔21〕.

▪ 프로세스 환경 블록

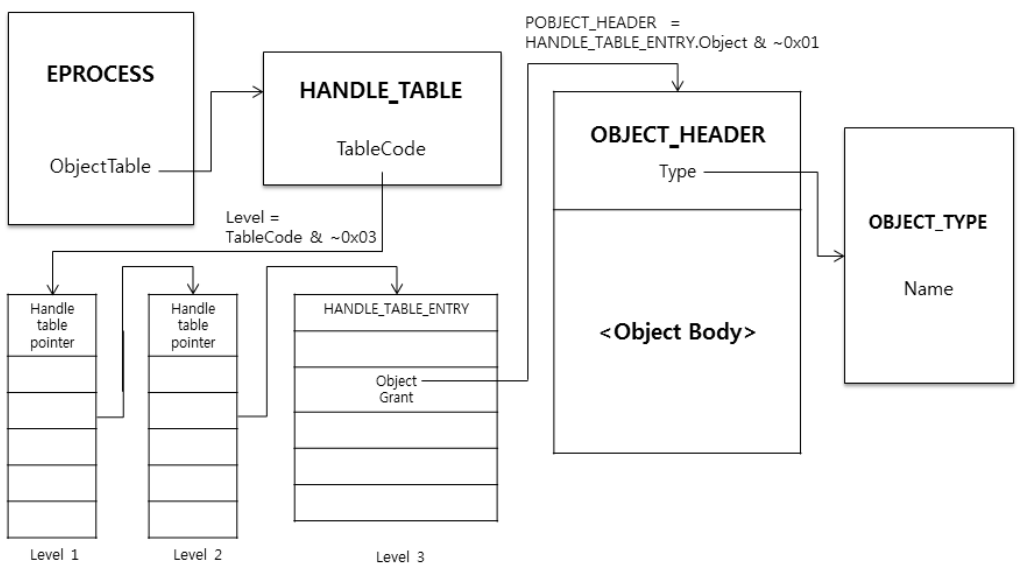
프로세스 환경 블록(PEB)은 각 프로세스의 환경을 나타내는 구조체이다. 각 프로세스의 PEB 주소는 EPROCESS.Peb 필드에 저장되어 있으며 이 주소는 프로세스 주소 공간 내의 주소를 가리키기 때문에 KPROCESS.DirectoryTableBase를 이용하여 프로세스 주소 공간을 복원해야 한다. PEB로부터 얻을 수 있는 주요 환경 변수는 〔표 2〕와 같다.

▪ 로드된 모듈 목록

로드된 모듈이란 커널 영역에서는 커널 드라이버, 사용자 영역에서는 DLL을 의미한다. 로드된 모듈의 정보는 LDR_DATA_TABLE_ENTRY 구조체 안에 정의되어 있다.

커널 드라이버의 목록을 얻기 위해서는 커널 변수 중 PsLoadedModuleList를 참조하면, 원형 이중 연결 리스트로 구성된 LDR_DATA_TABLE_ENTRY 구조체들을 얻을 수 있다.

사용자 영역에서의 DLL은 각 프로세스의 PEB로부터 Ldr 필드의 주소를 참조하여 PEB_LDR_DATA 구조체에 접근한 후, InLoadOrderMo-



〔그림 4〕 3단계 핸들 테이블 구조

duleList, InMemoryOrder-ModuleList, In-InitializationOrderModuleList의 세 개의 링크 중 하나를 참조하면 LDR_DATA_TABLE_ENTRY 구조체들을 얻을 수 있다.

▪ **핸들 테이블**

〔그림 3〕 핸들의 구조

핸들은 프로세스가 참조하는 시스템 객체의 인터페이스이다. 프로세스가 핸들로 참조할 수 있는 객체는 프로세스, 스레드, 파일, 레지스트리 키 등이 있다. 핸들 테이블을 분석함으로써 현재 프로세스가 소유하고 있는 프로세스, 스레드 및 현재 열고 있는 파일 등을 얻을 수 있다.

핸들은 비트 단위로 나뉘어 〔그림 3〕과 같이 핸들 테이블의 인덱스로써 사용된다. 핸들 테이블의 구조는 최대 3단계까지 표현할 수 있는데 〔그림 4〕는 3단계 핸들 테이블의 구조를 나타낸다.

EPROCESS의 ObjectTable 필드는 HANDLE_TABLE 구조체를 가리키고 있다. HANDLE_TABLE 구조체의 TableCode 필드는 하위 3비트를 핸들 테이블 구조의 단계를 표현하는데 사용된다. 만약 2~3 단계를 나타낸다면 하위 3비트를 제외한 나머지 비트가 핸들 테이블 포인터 배열의 시작 주소를 가리키며, 1단계인 경우는 바로 HANDLE_TABLE_ENTRY 배열의 시작 주소를 가리킨다.

HANDLE_TABLE_ENTRY.Object 필드는 각 객체의 헤더인 OBJECT_HEADER의 시작 주소를 가리키는데, 이때 하위 1비트는 잠금 비트로써 사용되기 때문에 AND 연산 마스킹을 통해서 제거해야 한다.

각 객체의 형태는 OBJECT_HEADER.Type을 통해 OBJECT_TYPE 구조체를 참조하여 확인할 수 있다. 예를 들어 객체 형태가 "Process"를 나타낸다면 객체의 본체는 EPROCESS 구조체이며, "File"을 나타낸다면 객체의 본체는 FILE_OBJECT 구조체이다.

3.2.3 물리 메모리 카빙 기법

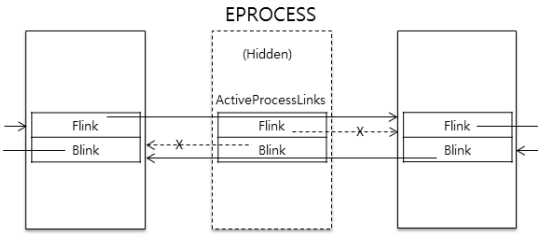
물리 메모리 카빙 기법은 파일 카빙 기법과 유사하게 커널 객체의 시그니처를 이용해 커널 객체를 추출해내는 기법이다. 이 기법을 통해 현재 사용되고 있는 커널 객체 뿐만이 아닌, 사용하고 난 뒤 비할당 페이지에 남겨진 커널 객체까지도 찾아낼 수 있다. 프로세스 객체인 EPROCESS를 예로 들자면, 현재 실행중인 프로세스 이외에도 이미 종료된 프로세스의 객체까지 찾아내어 종료된 시간까지 확인할 수 있다〔22〕.

〔표 3〕 EPROCESS 각 필드의 필수 조건

필드	조건
Pcb.ReadyListHead.Flink	val & 0x80000000 == 0x80000000 && val % 0x8 == 0
Pcb.ThreadListHead.Flink	val & 0x80000000 == 0x80000000 && val % 0x8 == 0
WorkingSetLock.Count	val == 1 && val & 0x1 == 0x1
Vm.VmWorkingSetList	val & 0xC0003000 == 0xC0003000 && val % 0x1000 == 0
VadRoot	val == 0 || (val & 0x80000000 == 0x80000000 && val % 0x8 == 0)
Token.Value	val & 0xE0000000 == 0xE0000000
AddressCreationLock.Count	val == 1 && val & 0x1 == 0x1
VadHint	val == 0 || (val & 0x80000000 == 0x80000000 && val % 0x8 == 0)
Token.Object	val & 0xE0000000 == 0xE0000000
QuotaBlock	val & 0x80000000 == 0x80000000 && val % 0x8 == 0
ObjectTable	val == 0 || (val & 0xE0000000 == 0xE0000000 && val % 0x8 == 0)
GrantedAccess	val & 0x1F07FB == 0x1F07FB
ActibeProcessLinks.Flink	val & 0x80000000 == 0x80000000 && val % 0x8 == 0
Peb	val == 0 || (val & 0x7FFD0000 == 0x7FFD0000 && val % 0x1000 == 0)
Peb.DirectoryTableBase.0	val % 0x20 == 0

커널 객체들이 메모리에 할당되는 경우 3.1.2절에서 언급한 풀 태그가 따라붙게 되는데, 이는 할당된 커널 객체의 형태를 판단하는 요소로써 사용할 수 있다. 그 이외에, 객체의 각 필드에 따른 조건을 통해 올바른 객체의 형태를 갖는지 확인할 수 있는데, [표 3]은 EPROCESS 구조체를 예를 들어 각 필드의 조건을 설명하고 있다[23].

만약 프로세스를 대상으로 물리 메모리 카빙 기법을 적용한다면, 자신 혹은 다른 프로세스를 은닉하기 위한 루트킷을 탐지할 수 있다. 대부분의 루트킷은 SSDT(System Service Descriptor Table)를 후킹하여 시스템 콜를 가로채거나, [그림 5]와 같은 커널 객체 직접 수정(DKOM)을 통해 대상 프로세스의 EPROCESS 구조체의 원형 연결 리스트 구조를 끊어 프로세스 목록에서 대상 프로세스를 은닉시킨다[24]. 하지만 이는 프로세스 목록에서 보이지 않을 뿐 실제로는 이상 없이 실행 중이기 때문에, 실제 해당 프로세스의 객체를 직접 추출한다면 은닉 프로세스를 찾을 수 있다.



[그림 5] 커널 객체 직접 수정

IV. 물리 메모리 분석 도구

4.1 도구 설계

물리 메모리 분석 도구 구현을 위한 설계는 [그림 6]과 같다. 물리 메모리 파일 내의 커널 객체들을 분석하는 모듈들은 사용하는 주소 공간에 따라 커널 레이어와 사용자 레이어로 구분된다. 그 밖에는 루트킷 탐지를 위한 프로세스 정보 비교 모듈, 텍스트 추출 모듈 및 ProcessLibrary.com[25]으로부터 프로세스 정보를 제공받는 모듈이 있다. 다음은 각 모듈에 대한 상세한 설명이다.

▪ 커널 레이어

◦ CMemSpace 클래스

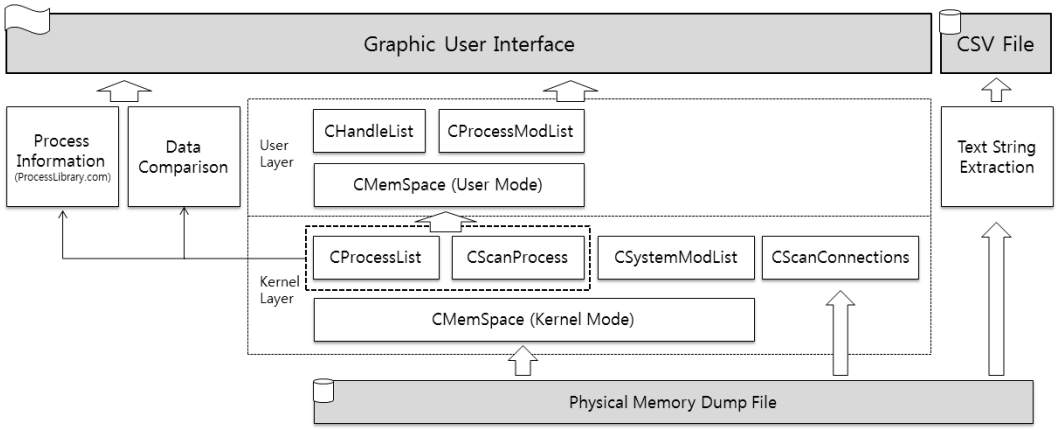
커널 전역 변수 및 디렉토리 테이블 베이스를 탐색한다. 또한 디렉토리 테이블 베이스를 이용해 상위 클래스로부터 전달 받은 가상 주소를 물리 주소로 변환 후 물리 영역을 읽어오는 역할을 수행한다. 이 클래스는 상위 클래스의 주소 공간에 대한 추상화를 제공한다.

◦ CProcessList 클래스

커널 전역 변수인 PsActiveProcessHead로부터 원형 이중 연결 리스트로 구현된 EPROCESS 구조체의 목록을 얻어온다.

◦ CScanProcess 클래스

물리 메모리 전역을 대상으로 카빙 기법을 적용하여 EPROCESS 구조체를 추출한다. 이 클래스는 DKOM을 이용한 은닉 프로세스 및 종료된 프로세스



[그림 6] 물리 메모리 분석 모듈 관계도

를 탐지할 수 있다.

◦ CSystemModList 클래스

커널 전역 변수인 PsLoadedModuleList로부터 로드 된 커널 드라이버의 목록을 얻어온다.

◦ CScanConnections 클래스

물리 메모리 전역을 대상으로 TCP 세션 정보를 담고 있는 TCB 구조체를 추출하여 프로세스의 네트워크 세션 정보를 구성한다.

▪ 사용자 레이어

◦ CMemSpace 클래스

커널 레이어에 존재하는 CMemSpace와 동일한 클래스이며, 상위 클래스인 CHandleList와 CProcessModList에 프로세스 주소 공간을 위한 추상화를 제공한다.

◦ CHandleList 클래스

CProcessList 및 CScanProcess 클래스로부터 전달 받은 프로세스 정보를 통해, 각 프로세스가 가지고 있는 핸들 테이블을 분석하여 데이터를 제공한다.

◦ CProcessModList 클래스

각 프로세스가 로드한 DLL의 목록을 얻어온다.

▪ 보조 모듈

◦ 데이터 비교 모듈

CProcessList와 CScanProcess로부터 얻어온 프로세스 정보를 비교한다. 연결 리스트에 연결되지 않은 프로세스가 종료되지 않은 채 정상적으로 실행되고 있다면 루트킷에 의한 은닉으로 판단할 수 있다.

◦ 프로세스 정보 모듈

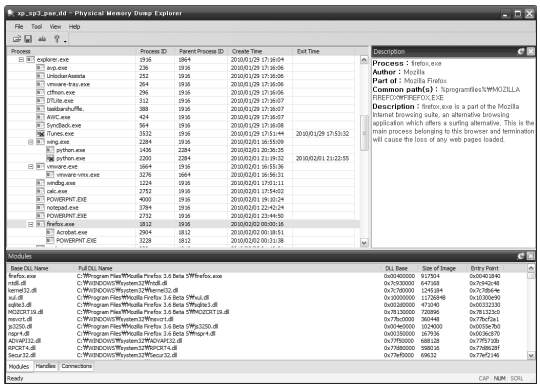
프로세스 정보를 제공하는 ProcessLibrary.com으로부터 얻은 정보를 파싱하여 GUI 레이어에 제공한다. 사용자가 프로세스에 대한 정보를 알지 못하더라도 프로세스의 용도를 파악할 수 있다.

◦ 텍스트 문자열 추출 모듈

물리 메모리 덤프 파일 내의 모든 ASCII, UTF-8, UTF-16 문자열을 추출한다. 옵션으로 문자열의 오프셋 및 16진수 출력이 가능하며, 손쉽게 사전 파일을 생성할 수 있다.

4.2 구현 결과

〔그림 7〕은 구현 완료된 물리 메모리 분석 도구인 "Physical Memory Dump Explorer"이다. 메인 뷰에는 실행, 종료, 은닉 된 프로세스를 아이콘으로 구분하여 부모-자식 관계에 따라 트리 형태로 출력한다. 또한, 프로세스 ID, 부모 ID, 실행 시간, 종료 시간의 기본 정보를 리스트 형태로 출력하여 한눈에 프로세스 정보 및 구조를 파악하기 쉽게 하였다.



〔그림 7〕 구현 완료된 물리 메모리 분석 도구
"Physical Memory Dump Explorer"

하단 뷰에는 시스템 커널 드라이버, 로드된 DLL, 핸들, TCP 세션 정보를 출력하며, 우측 뷰에는 ProcessLibrary.com으로부터 전송 받은 프로세스의 정보(프로세스 이름, 작성자, 프로그램 패키지 이름, 일반적인 설치 경로, 설명)가 출력된다. 이들은 메인 뷰에서 해당 프로세스를 선택하면 자동으로 정보가 출력된다.

본 도구는 ASCII, UTF-8, UTF-16로 인코딩된 문자열 추출을 지원하며, 문자열의 오프셋 및 16진수 형태도 함께 출력 가능하다. 또한 CSV 포맷으로 출력할 때, 문자열을 몇 줄 단위로 잘라 저장할 것인지를 지정할 수 있다.

4.3 기존 도구와의 차이점

2.1절에서 설명한 기존 도구들은 모두 콘솔 명령어 기반의 도구이기 때문에 사용자 편의성이 부족하다. 그리고 일부 도구의 경우 스크립트 언어로 작성되었기 때문에, 동작을 위해서는 반드시 인터프리터를 설치해야한다. 또한 모두 최신 운영체제인 윈도우즈 7을 지

원하지 않고, 하나의 운영체제 버전만을 대상으로 하거나 다수의 버전을 지원하더라도 서비스팩 설치 유무에 따라 제대로 동작하지 않는 경우가 있다.

하지만 Physical Memory Dump Explorer는 그래픽 사용자 인터페이스(GUI)를 제공하기 때문에 도구가 보여줄 수 있는 모든 정보를 한눈에 확인할 수 있어 사용자에게 높은 편의성을 제공한다. 그리고 윈도우즈 XP부터 7까지의 대부분의 윈도우즈 운영체제를 지원하여 기존 도구와 대비하여 높은 호환성을 가진다. 또한 물리 메모리 분석에 필요한 대부분의 기능들을 하나의 도구에서 지원하여 효율성 측면에서도 큰 장점을 보인다.

4.4 물리 메모리 분석 결과

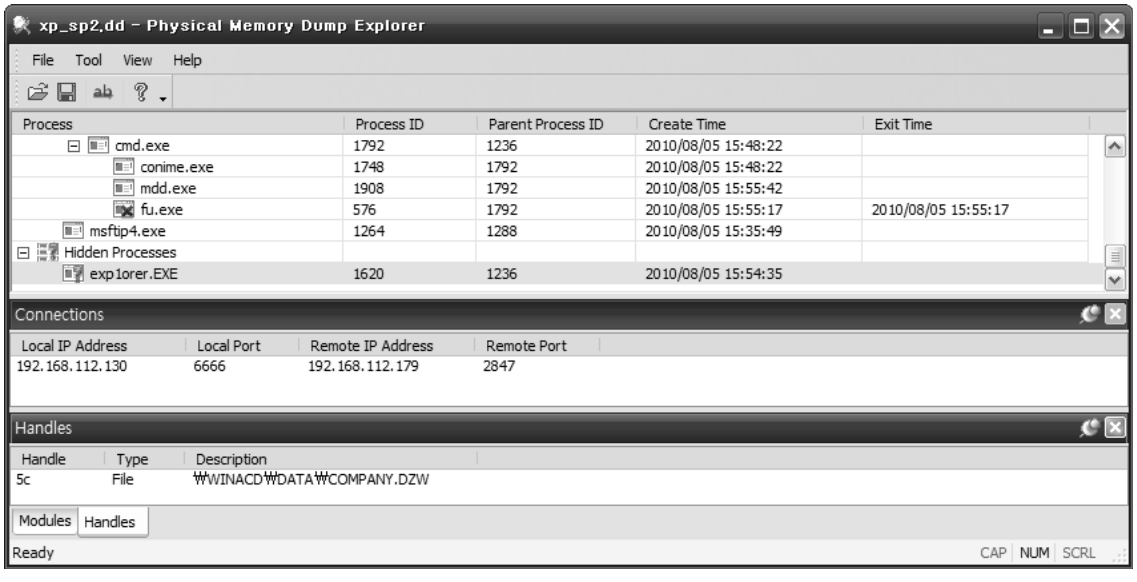
물리 메모리 분석 도구를 이용한 테스트 시스템 분석을 수행하였다. 대상 시스템의 OS는 윈도우즈 XP SP2, 물리 메모리의 용량은 1GB이다. 대상 시스템의 물리 메모리를 수집 후 물리 메모리 분석을 수행한 결과는 〔그림 8〕과 같다.

분석 결과 'exp1orer.exe'라는 은닉 프로세스가 탐지되었으며, 'fu.exe' 프로세스는 실행과 동시에 종료되었다. 'fu.exe'는 DKOM으로 프로세스를 은닉시키는 FU〔26〕라는 루트킷이며, 'exp1orer.exe' 프로세스를 은닉 후 종료된 것으로 판단된다. 'exp1orer.exe'가 열고 있는 파일인 '\WINACD\DATA\COMPANY.DZW'는 회계 소프트웨어의 회계 장부 데이터 파일이며, 현재 TCP/6666 포트를 열고 192.168.112.179 IP주소를 가진 시스템으로 회계 장부 데이터 파일을 전송하고 있음을 예상할 수 있다.

다음, 동일 시스템의 하드디스크로부터 나온 암호화 된 압축파일인 'EncryptedZip.zip'을 문자열 추출 후, 이를 이용해 사전 공격을 통한 복호화를 시도하였다. 아스키 문자열 6,801,869개, UTF-8 문자열 22,442개, UTF-16 문자열 2,768,141개, 총 9,592,452개의 문자열을 압축 파일 패스워드 복구 도구인 Advanced Archive Password Recovery〔27〕를 이용하여 사전 공격한 결과, 2시간 17분 23초 만에 복호화에 성공하여 물리 메모리에서 추출한 문자열이 패스워드 사전 공격에 매우 효과적임을 확인하였다.

V. 결론 및 향후 연구

전통적인 포렌식 방법론인 하드디스크 분석이 효율적인 측면에서 한계를 드러내며, 최근 라이브 포렌식이 관심을 받으면서 급격히 발전하고 있다. 활성 데이터는 전원이 차단되면 소멸되는 프로세스, 네트워크, 열려있는 파일 등의 시스템 동작 상태를 의미하기 때문에 하드디스크 분석만으로는 얻을 수 없는 중요한 정보이다. 최근까지의 라이브 포렌식 방법론 중 가장 널리 쓰이는 방법은, 시스템의 정보를 나열하는 명령



〔그림 8〕 물리 메모리 분석 결과

어 기반 도구의 실행 결과를 취합하는 것이다. 하지만 이는 시스템에 상대적으로 큰 변화를 가져오며, 악성코드에 의한 결과 변조, 그리고 재분석이 용이하지 못하다는 단점을 안고 있어 그 대안이 필요한 시점이다.

본 논문이 소개한 물리 메모리 분석 방법은 현재 널리 쓰이고 있는 도구를 이용한 활성데이터 수집과는 달리, 시스템에 최소한의 영향을 끼치면서 악성코드에 의한 결과 변조를 우회하며, 물리 메모리 덤프 파일을 언제든지 상세히 재분석 할 수 있다는 장점을 가지고 있다. 이 분석 방법을 적용하여 설계 및 구현한 물리 메모리 분석 도구인 'Physical Memory Dump Explorer'는 물리 메모리 분석 방법의 장점을 그대로 따르면서 포렌식 조사관에게 편의성을 제공하여 라이브 포렌식 조사 시 상당한 효율의 상승을 가져올 수 있다. 그리고 자체 실험 결과, 루트킷 탐지 및 악성코드의 행동 분석이 가능하고 텍스트 추출 기능을 이용해 사전 파일 생성 시, 적은 시간 내에 높은 확률로 암호화된 파일의 패스워드를 찾을 수 있음을 입증하였다. 향후 본 도구는 64비트 버전까지 확장할 계획이며, 추가적인 윈도우즈 커널 및 악성코드 분석 기능을 추가해 종합적인 라이브 포렌식 도구로서 발전시킬 계획이다.

참고문헌

〔1〕 R. Jones, "Safer Live Forensic Acquisition," University of Kent at Canterbury, http://www.cs.kent.ac.uk/pubs/ug/2007/co620-projects/forensic/report.pdf

〔2〕 E. Kenneally, and C. Brown, "Risk Sensitive Digital Evidence Collection," Digital Investigation, Vol. 2, Issue 2, pp. 101-119, June 2005.

〔3〕 I. Sutherland, J. Evans, T. Tryfonas, and A. Blyth, "Acquiring Volatile Operating System Data Tools and Techniques," ACM SIGOPS Operating Systems Review, Vol. 42, Issue 3, pp. 65-73, April 2008.

〔4〕 M. Monty, "Live Forensics on a Windows System: Using Windows Forensic Toolchest," Jun 2006. http://www.foolmoon.net/security

〔5〕 B. Carrier, J. Grand, "A Hardware-based Memory Acquisition Procedure for Digital Investigations," Digital Investigation, Vol. 1, Issue 1, pp. 50-60, February 2004.

〔6〕 A. Boileau, "Hit By A Bus: Physical Access Attacks with Firewire.," RUXCON, Sydney, Australia, September 30-October 1, 2006.

〔7〕 Mantech's Memory DD, Mantech, http://www.mantech.com/

〔8〕 MoonSols Windows Memory Toolkit, MoonSols, http://moonsols.com/component/jdownloads/view.download/3/2

〔9〕 H. Carvey, Windows Forensic Analysis DVD Toolkit, Second Edition, Syngress, June 2009.

〔10〕 A. Walters, "FATKit: Detecting Malicious Library Injection and Upping the "Anti"", July 2006.

〔11〕 B. D. Carrier and J. Grand, "A hardwarebased memory acquisition procedure for digital investigations," Digital Investigation, Vol. 1, No. 1, pp. 50 - 60, 2004.

〔12〕 MemParser, C. Betz, http://memparser.sourceforge.net/

〔13〕 PtFinder, A. Schuster, http://computer.forensikblog.de/files/ptfinder/

〔14〕 Volatility, Volatile Systems, https://www.volatilesystems.com/default/volatility

〔15〕 M.E. Russinovich and D.A. Solomon, Windows Internals, 5th Edition, Microsoft Press, June 2009.

〔16〕 S.M. Hejazia, C. Talhia and M. Debbabi, "Extraction of forensically sensitive information from windows physical memory," Digital Investigation, Vol. 6, Supplement 1, pp. S121-S131, September 2009.

〔17〕 M. Burdach. "Finding digital evidence in physical memory," http://forensic.seccure.net/pdf/mburdach physical memory forensics bh06.pdf, 2006.

〔18〕 A. Schuster, "The impact of Microsoft Windows pool allocation strategies on memory forensics," The Proceedings of the Eighth Annual DFRWS Conference, Vol. 5, Supplement 1, pp. S58-S64, September 2008.
〔19〕 S. Lee, A. Savoldi, S. Lee, and J. Lim, "Password Recovery Using an Evidence Collection Tool and Countermeasures," Intelligent Information Hiding and Multimedia Signal Processing, Vol. 2, pp. 97-102, November 2007.
〔20〕 R. Zhang, L. Wang, and S. Zhang, "Windows Memory Analysis Based on KPCR," Proceedings of the 2009 Fifth International Conference on Information Assurance and Security, Vol. 2, pp. 677-680, August 2009.
〔21〕 D. Aumaitre, "A Little Journey inside Windows Memory," Journal in Computer Virology, Vol. 5, No. 2, pp. 105-177, January 2009.
〔22〕 A. Schuster, "Searching for processes and threads in Microsoft Windows memory dumps," Digital Investigation, Vol. 3, Supplement 1, pp. 10-16, September 2006.
〔23〕 B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin, "Robust Signatures for Kernel Data Structures," Proceedings of the 16th ACM conference on Computer and communications security, pp. 566-577. 2009.
〔24〕 B. Blunden, The Rootkit Arsenal : Escape and Evasion in the Dark Corners of the System, Jones & Bartlett Publishers, May 2009.
〔25〕 ProcessLibrary.com, Uniblue, http://www.processlibrary.com/
〔26〕 FU Rootkit, http://rootkit.com/board_project_fused.php?did=proj12
〔27〕 Advanced Archive Password Recovery, Elcomsoft, http://www.elcomsoft.com/archpr.html

〈著者紹介〉

한 지 성 (Jisung Han) 정회원
2009년 2월: 경원대학교 전자거래학 학사
2009년 3월~현재: 고려대학교 정보경영공학전문대학원 석사과정
〈관심분야〉 라이브 포렌식, 운영체제, 역공학

이 상 진 (Sangjin Lee) 종신회원
1987년 2월: 고려대학교 학사 졸업
1989년 2월: 고려대학교 석사 졸업
1994년 2월: 고려대학교 박사 졸업
1989년 10월~1999년 2월: ETRI 연구원 역임
1999년 10월~현재: 고려대학교 정교수
1997년 12월: 국가안전기획부장 표창
〈관심분야〉 디지털 포렌식, 모바일 포렌식, 심층 암호, 해쉬 함수