情報保護學會論文誌
第 21 卷 第 4 號, 2011. 8

Mac OS X 물리 메모리 분석에 관한 연구*

이 경 식,† 이 상 진‡
고려대학교 정보경영공학전문대학원

Research on Mac OS X Physical Memory Analysis*

Kyeongsik Lee,† Sangjin Lee‡
Graduate School of Information Management and Security, Korea University

요 약

그간 디지털 포렌식의 활성 시스템 분석 분야의 한 화두는 물리 메모리 이미지 분석이었다. 물리 메모리 분석은 프로세스를 은닉을 하더라도 그 데이터를 물리 메모리에서 확인할 수 있고 메모리에 존재하는 사용자의 행위를 발견할 수 있어 분석 결과의 신뢰성을 높이는 등 디지털 포렌식 분석에 큰 도움이 되고 있다. 하지만 메모리 분석 기술 대부분이 윈도우 운영체제 환경에 초점이 맞추어져 있다. 이는 분석 대상의 다양성을 고려하였을 때 타 운영체제에 대한 메모리 분석이 필요하게 되었음을 의미한다. Mac OS X는 윈도우에 이어 두 번째로 높은 점유율을 가진 운영체제로 부팅 시 커널 이미지를 물리 메모리에 로딩하면서 운영체제가 구동하고 주요 정보를 커널이 관리한다. 본 논문은 Mac OS X의 커널 이미지가 저장하고 있는 심볼 정보를 이용한 물리 메모리 분석 방법을 제안하고, 제안한 내용을 토대로 물리 메모리 이미지에서 프로세스 정보와 마운트된 장치 정보, 커널 버전 정보, 외부 커널 모듈정보(KEXT)와 시스템 콜 목록 정보의 추출 방법을 보인다. 추가적으로 사례 분석을 통해 물리 메모리 분석의 효용성을 입증한다.

ABSTRACT

Physical memory analysis has been an issue on a field of live forensic analysis in digital forensics until now. It is very useful to make the result of analysis more reliable, because record of user behavior and data can be founded on physical memory although process is hided. But most memory analysis focuses on windows based system. Because the diversity of target system to be analyzed rises up, it is very important to analyze physical memory based on other OS, not Windows. Mac OS X, has second market share in Operating System, is operated by loading kernel image to physical memory area. In this paper, We propose a methodology for physical memory analysis on Mac OS X using symbol information in kernel image, and acquire a process information, mounted device information, kernel information, kernel extensions(eg. KEXT) and system call entry for detecting system call hooking. In additional to the methodology, we prove that physical memory analysis is very useful though experimental study.

Keywords: Mac OS X, Memory Forensics, Rootkit Detection

I. 서 론

컴퓨터의 보조 기억장치인 하드 디스크의 용량이 점점 커짐에 따라 디지털 포렌식 분석의 대상이 되는 시스템의 디스크를 이미징하여 분석하는 것이 점점 시간상의 한계에 부딪치기 시작했다. 이에 최근 디지털 포렌식 분야는 전체 디스크를 이미징하고 디스크 이미

접수일(2010년 9월 29일), 게재확정일(2010년 11월 5일)
* 본 연구는 한국연구재단을 통해 교육과학기술부의 바이오 연구개발사업으로부터 지원받아 수행되었습니다. (2010020634)
† 주저자, rapfer@gmail.com
‡ 교신저자, sangjin@korea.ac.kr

지만을 분석하는 기존 포렌식 방식에서 컴퓨터의 주 기억장치인 물리 메모리와 같은 휘발성 데이터를 수집하고 디스크 이미지와 함께 분석하여 좀 더 디지털 포렌식 분석 결과의 신뢰성과 효율성을 높이는 방향으로 진행되고 있다.

물리 메모리 이미지 분석은 활성 시스템 포렌식 분야 중 하나로 운영체제가 데이터를 사용하기 위해 물리 메모리에 데이터를 로드해야 한다는 점과 프로세스 목록 및 네트워크 정보 같은 운영체제가 지속적으로 관리하는 데이터를 커널 메모리 영역에 보관하여 관리한다는 점을 착안하여 물리 메모리에 존재하는 시스템의 정보를 추출하는 방법을 의미한다. 물리 메모리 이미지 분석은 프로세스를 은닉하더라도 그 데이터를 물리 메모리에서 확인할 수 있고, 메모리에만 존재하는 사용자 행위를 발견할 수 있기 때문에 포렌식 수사에 큰 도움이 되고 있으며 현재에도 많은 연구가 진행되고 있다.

하지만 그간 물리 메모리 분석에 관한 연구는 윈도우 환경에만 초점이 맞추어졌으며, 윈도우 다음으로 개인 사용자가 많이 사용하는 Mac OS X의 물리 메모리 분석은 거의 연구되지 않았다. 이는 윈도우에만 국한된 물리 메모리 분석에서 Mac OS X의 물리 메모리도 디지털 포렌식 관점의 연구가 필요하게 되었음을 의미한다.

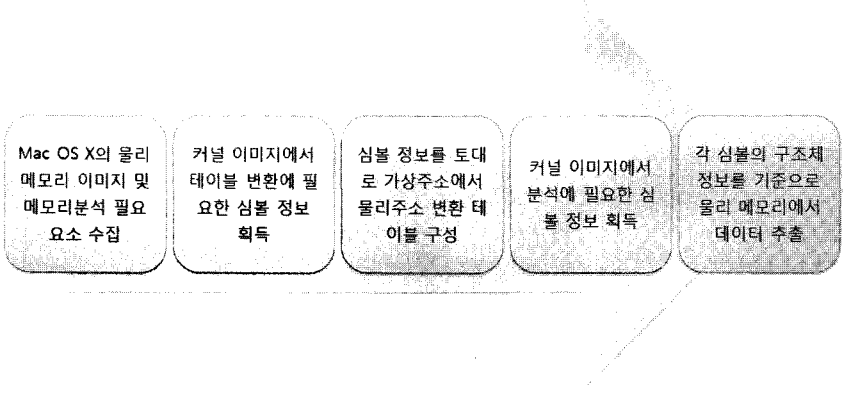
본 논문에서는 윈도우 다음의 점유율을 가지는 Mac OS X의 물리 메모리 이미지 수집부터 분석까지 절차를 기술하고 물리 메모리 이미지에서 주요 정보 추출방안과 현재까지 발표된 Mac OS X의 루트킷을 물리 메모리 분석을 통해 어떻게 찾아낼 수 있는지에 대해 기술한다. 추가적으로 시스템 콜 후킹 탐지 방안을 제안하고, 사례 분석을 통해 그 효용성을 입증한다.

II. Mac OS X의 물리 메모리 수집 및 분석 절차

물리 메모리에 데이터를 저장하고 처리하는 형태는 운영체제뿐만 아니라 CPU 아키텍처에도 의존성을 가지고 있다. 애플 컴퓨터의 Mac OS X는 과거에는 PowerPC 아키텍처를 채용하였으나, 최근에는 PowerPC보다 인텔 아키텍처를 지원하고 있다. 본 논문은 애플 제품군에서 가장 많이 사용되며, PowerPC기반 Mac OS X보다 점유율이 높은 인텔 아키텍처 기반에 애플의 최신 운영체제인 Snow Leopard(10.6) 버전을 기준으로 분석을 수행하였다. Mac OS X의 물리 메모리 이미지 수집부터 분석까지 진행 절차를 [그림 1]과 같이 제시하였다.

[그림 1]의 분석절차는 윈도우의 물리 메모리 이미지 분석 방식과 큰 절차는 동일하지만, 심볼 테이블을 획득하는 방법이나 가상주소에서 물리주소로 변환하는 테이블 변환 방식이 윈도우와 약간의 차이점이 있다.

애플은 마이크로소프트와 같이 심볼 서버를 따로 제공하지 않으며, 커널 디버깅 시 커널 이미지 내부의 심볼 정보를 이용하여 디버깅할 수 있도록 하여 커널 이미지 분석으로 심볼 정보를 획득할 수 있다. 심볼 정보는 커널 디버깅에 사용되므로, 가상 주소 정보가 올바르게 구성하며, 모든 심볼 주소는 커널의 주소변환 테이블을 거쳐서 물리 메모리 주소에 접근할 수 있도록 한다. 윈도우 시스템과 다르게 Mac OS X는 주소 확장 및 64비트 지원을 위해 PAE(Page Address Extensions) 기술과 PML4(Page-Map Level4)



[그림 1] Mac OS X의 물리 메모리 수집 및 분석 절차

〔표 1〕 각 진행과정에 따라 획득할 수 있는 정보

진행	획득 정보
1	분석을 위한 데이터인 커널 이미지, 물리 메모리 이미지, 버전 정보 획득
2	주소변환 테이블의 시작 주소를 가지는 심볼의 가상 주소 획득
3	심볼 데이터의 물리 메모리의 위치 정보를 알 수 있는 테이블 획득
4	커널 이미지 내의 많은 심볼 데이터 중 분석할 심볼 정보만을 선정, 가상 주소 획득
5	분석된 데이터 획득

기술이 사용되어 물리 메모리 분석 시 윈도우의 32비트 환경의 기본 주소 변환 방식을 사용하면 잘못된 결과를 얻을 수 있다.

총 5가지의 분석 절차는 하나의 항목이 제외되면 물리 메모리 분석을 수행할 수 없으며, 각 항목의 진행을 통해 〔표 1〕과 같은 데이터를 획득할 수 있다.

III. 물리 메모리 분석 필요 요소 수집

Mac OS X의 물리 메모리를 분석하기 위해서는 크게 분석할 물리 메모리 이미지와 커널 이미지 파일을 수집하고 커널이 어떠한 CPU 아키텍처로 로드 되었는지를 확인하는 작업이 필요하다.

현재까지 알려진 Mac OS X의 물리 메모리 이미지 수집 방법은 firewire 케이블을 이용하여 이미징하는 방법과 물리 메모리 대신 '/dev/kmem'모듈을 설치하고 'dd'를 이용하여 커널 가상 메모리 영역을 덤프하는 방법, 그리고 하이버네이션(Hibernation) 이미지를 수집하는 방법이 있다.

Firewire케이블을 이용하여 물리 메모리를 수집하는 방법은 2005년 CanSecWest에서 리눅스와 Mac OS X의 물리 메모리 수집 기법을 발표한 Micheal Bencher가 홈페이지를 통해 제공하는 python에서 firewire 케이블을 사용할 수 있는 'pyfw'라이브러리를 이용하여 수집할 수 있다 〔1〕. firewire 케이블을 이용한 물리 메모리 이미징은 속도가 빠르고, 수집할 시스템에 어떠한 작업도 수행하지 않음으로 현재 물리 메모리의 무결성 침해를 최소화할 수 있다. 수집할 시스템에 firewire 단자가 있어야하는 문제점이 있지만, 애플의 데스크탑과 랩탑 계열이 기본으로 단자를 제공하기 때문에 큰 문제가 되지 않는다. firewire 케이블을 이용한 물리 메모리 수집은 무결성 측면에서 디지털 포렌식에 적합한 방법이다.

애플은 Mac OS X에서 유닉스 시스템의 물리 메모리 이미지를 획득하는데 사용되는 장치인 '/dev/mem'과 '/dev/kmem'을 메모리에 악의적인 접근을 방지하기 위해 제거하였다. 대신에 애플은 하드웨어 개발자를 위해 Mac OS X에 KVM(Kernel Virtual Memory) 라이브러리를 이용하여 커널영역의 가상 메모리 공간을 접근할 수 있도록 제공한다. KVM라이브러리를 이용하면 '/dev/kmem'장치를 애플의 커널 확장 모듈인 KEXT로 구현할 수 있다 〔2〕. 하지만 이 방법은 KEXT를 생성하고 로드하는 과정에서 저장장치와 물리 메모리의 무결성이 훼손되고 커널 가상 메모리 영역만을 덤프할 수 있는 문제로 firewire 단자가 없는 시스템에서 차선책으로 사용되어야 할 방법이다.

마지막 방법은 하이버네이션 기능으로 생성되는 하이버네이션 이미지를 수집하여 분석하는 방법이다. 하이버네이션 기능은 노트북과 같은 이동성을 중시하는 제품군에 기본적으로 설정되어 있으며, 기본 경로는 '/private/var/vm' 디렉터리에 'sleepimage'라는 파일로 존재한다. 파일은 가장 최근에 시스템을 하이버네이션 상태로 진입하였을 때의 물리 메모리 이미지를 저장하며 'pmset'명령으로 현재 설정 상태를 확인할 수 있다. 하이버네이션 이미지가 존재하지 않거나 'hibernatemode'가 3으로 설정되어 있다면, '1(suspend to disk only)'로 설정하여 현재 시점의 하이버네이션 이미지를 생성할 수 있다. 하이버네이션 이미지는 암호화하여 저장하며 그 암호화 기법이 애플 기술문서에 명시되어 있지 않기 때문에, 실제 물리 메모리 이미지와 그 구조가 상이하며 복호화 알고리즘의 규명이 필요하다 〔3〕. 하지만, 실제 이미지를 확인했을 때 평문 텍스트가 자주 발견되었으며, 이를 strings와 같은 기본적인 포렌식 분석 도구를 이용하면 일부 분석이 가능하다.

물리 메모리 수집을 완료하더라도, 수사관에게 필요로 하는 요소들이 물리 메모리에 어떠한 위치에 데이터를 저장하는지 확인해야한다. 커널은 운영체제를 관리 목적으로 커널 가상메모리 영역에 디지털 포렌식 분석에 유용한 다양한 정보를 가지고 있다. 윈도우의 경우 커널 심볼(symbol)정보가 독립적인 파일(*.pdb)로 되어 있으며, 마이크로소프트에서 제공하는 심볼 정보를 받아 분석을 수행할 수 있다. Mac OS X는 애플 컴퓨터에서 별도의 파일로 심볼 정보를 제공하고 있지는 않지만 Mac OS X에 존재하는 커널

이미지 파일인 'mach_kernel'에 커널 심볼 정보를 내장하고 있기 때문에 심볼 정보를 획득하여 물리 메모리 분석을 수행할 수 있다. 즉 물리 메모리 분석 시에는 물리 메모리를 수집한 후 커널 이미지인 'mach_kernel'을 가져오거나 심볼 정보를 추출해야 한다.

추가적으로 수집한 시스템의 커널이 어떠한 CPU 아키텍처로 로드 되었는지 확인할 필요가 있다. 커널 이미지 파일은 Universal Binaries 포맷으로 이 포맷의 특징은 하나의 파일 포맷에 다양한 아키텍처를 위한 Mach-O 파일 포맷을 가질 수 있다 [4] . 최근 데스크탑 및 노트북 제품군의 Mac OS X는 일반적으로 Intel 32비트 커널을 로드하고 64비트 어애플리케이션은 64비트용 변환 테이블을 따로두어 운영하지만, 이는 커널 부팅 옵션을 변경하여 바꿀 수 있기 때문에 'uname -a'명령어를 통해 로드된 커널 버전을 확인해야 한다. 커널 버전 확인은 분석 시 시작 포인트를 잡는데 중요한 역할을 하며, 분석 시간을 줄이기 위해 꼭 필요한 부분이다.

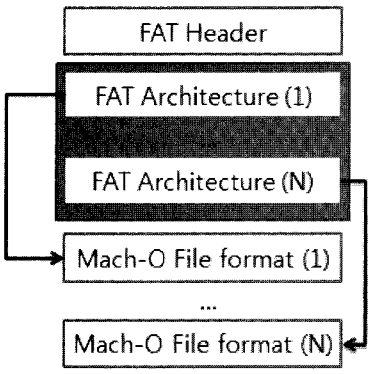
IV. 커널 이미지에서 심볼 정보 추출

커널 이미지(mach_kernel)는 Universal Binaries라 불리는 파일 포맷을 가지고 있으며 내부에 심볼 정보를 가지고 있다. 심볼 정보는 물리 메모리 이미지 분석에 필수요소로 커널 이미지가 사용하는 자료형의 이름과 가상 메모리의 주소를 표현하여 하드웨어 개발자들에게 커널 디버깅을 지원하는 역할을 한다. 심볼 정보는 가상 주소를 제공하기 때문에 커널 주소와 물리 주소 변환이 가능하다면, 물리 메모리에서 해당 자료형의 위치를 파악할 수 있다.

4.1 Universal Binaries

Universal Binaries는 내부에 여러 Mach-O 파일 포맷 이미지를 가지고 있어 하나의 파일을 통해 다양한 아키텍처의 Mac OS X에서 동작하도록 설계하였다. 이에 사전에 수집한 시스템에서 얻은 로드된 커널 아키텍처 정보를 이용하여 Universal Binaries중 해당 아키텍처로 구현된 Mach-O 포맷 이미지의 심볼만을 추려 획득해야 한다. Universal Binaries를 간단하게 표현하면 [그림 2]와 같다.

Universal Binaries의 헤더 영역은 CPU 아키텍처에 상관없이 Big Endian형태로 구성되어 있으



[그림 2] Universal Binaries

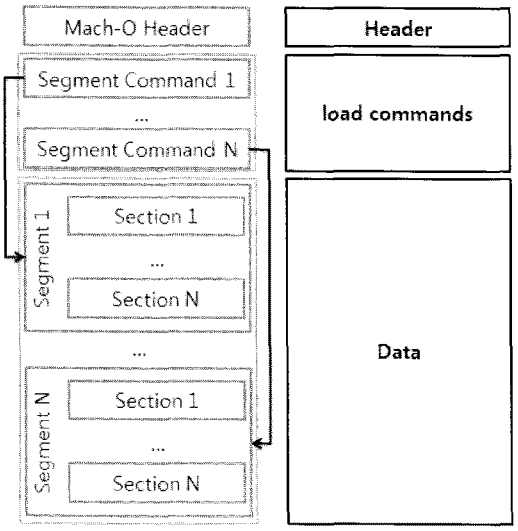
며, 총 8바이트 구조체의 'FAT Header'는 처음 4바이트의 '0xCAFEBABE' 시그너처로 파일 포맷의 일치성을 확인한다. 다음에 나오는 4바이트는 바이너리 파일 내에 존재하는 CPU 아키텍처의 수를 의미하며 'FAT Architecture'구조체는 CPU 아키텍처 수만큼 'FAT Header' 하부에 위치한다.

'FAT Architecture'는 각각이 단일 아키텍처에 대해 분류되는 Mach-O 파일의 정보와 영역 정보를 가진다. 이 구조체는 CPU 타입이 표시되어 있으며, 해당 타입으로 구현된 Mach-O 포맷의 데이터 위치와 크기 그리고 정렬(Alignment) 정보를 가지고 있다. 물리 메모리 분석 시에 수집한 로드된 커널 정보와 이 구조체의 CPU정보를 비교하여 해당하는 위치와 크기 정보만 추출하면, 실제 사용하는 심볼들에 대해서만 추려서 분석할 수 있다.

4.2 Mach-O 파일 포맷에서 심볼 정보 추출

Universal Binaries의 FAT Architecture를 통해 분석할 아키텍처의 영역을 덤프하면, 해당 아키텍처 기반의 Mach-O 파일 포맷형태의 이미지를 획득할 수 있다. 파일 포맷의 기본적인 구조는 [그림 3]과 같다.

Mach-O는 프로세스 생성 시 실제 물리 메모리에 로드되는 부분으로 각각의 세그먼트를 관리하는 'load commands' 구조체를 가지고 세그먼트에 따라 섹션을 가지거나 별도의 구조체를 가진다. 다양한 세그먼트 중 'LC_SEGMENT'와 'LC_SEGMENT_64' 세그먼트는 파일이 프로세스로 메모리 주소 공간에 매핑될 때 주소 정보를 가지고 있으며, 구동에 필요한 실행 코드와 데이터 영역을 섹션으로 분류하여 데이터를



〔그림 3〕 Mach-O 파일 포맷

관리한다. '__LINKEDIT' 세그먼트는 동적 링커에 의해 사용되는 바이너리 데이터와 재할당 테이블 엔트리 정보와 문자열에 대한 심볼 정보를 가진다. 메모리 분석에 필요한 정보인 'LC_SYMTAB' 커맨드는 파일이 정적 및 동적 링커에 의해 링크될 때 사용할 심볼 정보와 커널 디버깅을 위한 심볼 정보를 가지고 있다. 'LC_DYSYMTAB' 커맨드는 앞의 'LC_SYMTAB' 커맨드에서 동적 링커에 의해 사용될 심볼 테이블 정보를 추가적으로 가지고 있다.

심볼 정보는 'LC_SYMTAB'의 세그먼트 정보를 기준으로 구조체를 파싱하여 각각의 심볼 명에 대한 가상 주소를 획득할 수 있다. 애플의 'XCode'설치 시 기본으로 설치되는 'gobjdump'명령어나 Mac OS X에서 기본적으로 제공하는 'otool', 'nm'을 이용하여도 심볼 정보를 획득할 수 있다. 〔그림 4〕는 본 과정을 수행하여 획득할 수 있는 커널 이미지 내의 심볼 정보 일부이다.

V. 가상주소에서 물리주소 변환 방식

커널 이미지에서 심볼을 획득 하더라도, 주소 정보가 가상 주소(Virtual Address)로 표현되기 때문에, 이를 물리 주소로 변환해야 한다. Mac OS X 중 본 논문에서 분석하는 인텔 아키텍처의 Snow Leopard는 64비트 프로그램과 32비트 프로그램이 동시에 동작하기 위해 32비트 주소의 한계를 탈피할 수 있는 인텔의 주소 확장 기술(PAE)과 PML4(64비트 주소 지정 기술)을 함께 사용한다 [5] .

```
00101000 g       0F SECT   0d 0000 __INITPT.__data _IdlePDPT
00100000 g       0F SECT   0d 0000 __INITPT.__data _IdlePML4
```

〔그림 5〕 커널 이미지의 변환 시작점의 심볼 정보

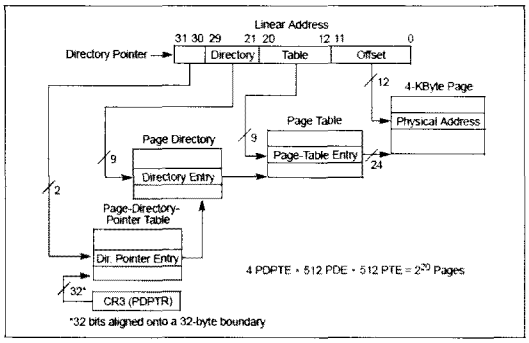
앞서 분석한 커널 이미지 파일의 심볼 정보를 확인하면 〔그림 5〕와 같이 'IdlePDPT', 'IdlePML4'를 확인할 수 있으며, 두 심볼은 각 아키텍처에 따른 가상 주소에서 물리 주소로 변환하는 테이블의 시작점을 가리킨다. 두 심볼의 가상 주소는 물리 메모리 상의 주소와 일치하게 되는데, 이는 커널 이미지가 부팅되면서 운영체제가 구동되고, 이 정보를 이용하여 주소 변환 테이블을 구성하여 물리 메모리의 주소를 그대로 선점하면서 나타나는 특징이다.

'IdlePDPT' 심볼의 가상 주소는 Mac OS X가 인텔 32비트 커널로 구동되었을 경우 사용되는 정보로 물리 메모리 변환 과정에서 'Page Directory Pointer Table'정보의 물리 메모리에서 시작 위치를 가리킨다. 이 정보는 운영체제가 가동되면서 초기화하는 과정에서 변수를 선언하고 초기화하기 때문에, 심볼 정보에서 설정된 가상주소를 그대로 물리 메모리 주소로 로드한다. PAE가 활성화된 상태에서 32비트 CPU의 가상주소에서 물리주소 변환과정은 〔그림 6〕와 같다.

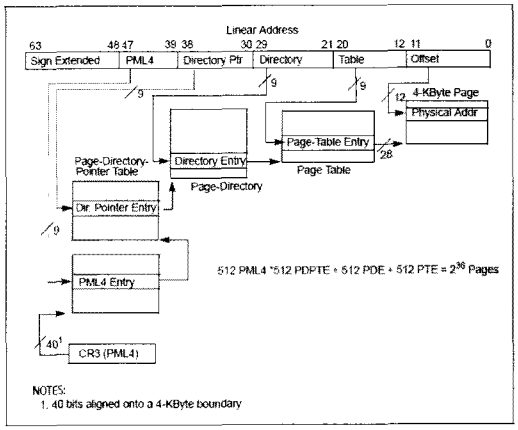
Mac OS X는 커널이 64비트 환경으로 로딩되거

```
0000000005614542       d  3c OPT     00 0000 radr://5614542
ffffff800066c50c g        0F SECT    08 0000 .data .constructors_used
ffffff800066c514 g        0F SECT    08 0000 .data .destructors_used
ffffff8000448f8e g        0F SECT    01 0000 .text _AllocateNode
ffffff8000204c3e g        0F SECT    01 0000 .text _Assert
ffffff80003ab93e g        0F SECT    01 0000 .text _BF_decrypt
ffffff80003ab594 g        0F SECT    01 0000 .text _BF_encrypt
```

〔그림 4〕 커널 이미지 내의 심볼 정보 추출



〔그림 6〕 Intel 32비트의 페이징 구조
(PAE Enable, Page Size: 4KB)



〔그림 7〕 IA-32e 모드 페이징 구조
(PAE Enable, Page Size: 4KB)

나 32비트 커널에서 64비트 애플리케이션 구동을 위해, 'IdlePML4'의 정보를 이용한다. 이 주소 변환은 인텔 공식 문서에 'PAE-Enabled Paging in IA-32E Mode'라고 명시되어 있으며, PAE활성 주소 방식에서 새로운 페이지 매핑 테이블인 PML4테이블을 이용하는 방식이다.

〔그림 7〕의 주소 표현방식은 64비트 시장에서 먼저 사용된 AMD64 아키텍처가 48비트 크기의 주소영역을 표현한다는 점 때문에 호환성을 위해 최대 48비트의 가상 주소 크기를 가리킬 수 있도록 구성되어 있다. 이 방식을 사용할 경우 'Page Directory Entry'와 'Page Table Entry'는 4바이트에서 8바이트 크기로 확장되어 다음 엔트리 주소를 가리킨다.

VI. 메모리 포렌식 분석에 필요한 심볼 추출

커널 이미지에서 포렌식 분석에 필요한 심볼 정보는 〔표 2〕와 같다.

주의할 점은 시스템 콜 목록은 커널 이미지 파일에서 심볼로 명시되어 있지 않다는 점이다. 하지만 시스템 콜의 개수를 나타내는 '_nsysent'의 정보를 이용하면 간접적으로 그 위치를 유추할 수 있다. 이는 분석 대상인 Snow Leopard 운영체제가 시스템 콜 정보를 '_nsysent'구조체 위에 위치한다는 점을 이용하는 것으로 C언어로 나타내면 〔그림 8〕과 같다.

각 심볼들은 변수 형태를 가지거나 구조체의 형태를 가지게 된다. 윈도우 운영체제의 경우 이러한 구조체 정보를 커널 디버깅을 하거나 심볼 파일을 통해 획득할 수 있지만, Mac OS X의 커널 이미지 상의 심볼 정보는 구조체 정보를 따로 담고 있지 않다. 애플은 하드웨어 개발자 및 시스템 프로그래머를 위해 Mac OS X의 일부 소스를 공개하였으며 오픈된 소스 중에는 애플의 커널인 XNU의 일부 소스도 존재하여 이를 이용해 유닉스 시스템과 같이 커널 구조체 정보를 획득할 수 있다 [6] . 여러 커널 버전이 존재하기 때문에 자신이 수집한 커널 버전에 맞는 공개된 XNU 커널의 일부 소스를 획득하여 분석에 사용할 수 있다.

〔표 2〕 획득할 구조체 정보

커널 심볼	얻을 수 있는 정보
__DATA.__common _osversion	Darwin커널의 버전정보
__DATA.__common _machine_info	운영체제 버전, 물리 메모리 정보, CPU정보
.data _g_kernel_kmod_info .data _kmod	Kernel Extensions(KEXT) 정보, KEXT의 시작 주소와 크기 정보
__DATA.__common _mountlist	시스템에 Mount된 장치 정보
__DATA.__common _kernproc	프로세스 목록 정보, 프로세스 시작 시간, 실행한 사용자, 연결된 파일 정보 등
.data __nsysent	System Call목록을 덤프하기 위해 사용되며, 시스템 콜의 개수를 표현

```
system_call_table = &nsysent - ( (int32_t)nsysent * sizeof(sys_call_entry) )
```

〔그림 8〕 시스템 콜 테이블의 물리 메모리 상의 위치 정보

VII. 물리 메모리 이미지에서 주요 정보 추출

앞 장의 정보를 기준으로 실제 데이터 분석을 수행하였다. 분석 프로그램은 python기반으로 구현하였고, mach-o파일 포맷 분석은 python에서 제공하는 라이브러리인 'macholib-1.2.1'버전을 이용하였다〔7〕. 가상 주소에서 물리 주소 변환은 Volatility Framework의 'x86.py'와 'addrspace.py'을 이용하였다〔8〕.

7.1 운영체제 기본 정보 추출

운영체제 정보인 'machine_info' 심볼의 구조체는 총 40바이트의 크기를 가지고 있으며, 구조체 정보는 'bsd/sys/machine.h'에 명시되어 있다. 구조체에는 XNU 커널 버전 정보와 CPU정보, 물리 메모리 정보를 가지고 있다. 〔그림 9〕는 물리 메모리 이미지에서 수집한 운영체제 정보이다.

'machine_info' 정보에서 출력되는 XNU 커널 버전 정보는 세부 버전 정보를 표현하지 않아 마이너 버전의 보안 패치 현황은 파악할 수 없다. Mac OS X는 추가적으로 Darwin 커널 버전 정보를 함께 사용하며, 이 정보를 이용하면 커널의 세부 버전 정보를 획득할 수 있다. Darwin 커널의 세부 버전 정보는 '_osversion'에 명시되어 있다.

7.2 커널 모듈(KEXT) 정보 추출

I/O Kit은 XNU 커널 내부의 객체 기반의 디바이스 드라이버 프레임워크로 동적으로 로드된 디바이스 드라이버를 추가 및 관리할 수 있다. I/O Kit을 통해

```
-= machine info structure =-
Major Version: 10
Minor Version: 4
Number of Physical CPUs: 2
Size of memory in bytes: 2147483648 bytes
Size of physical memory: 4026531840 bytes
Number of physical CPUs now available: 2
Max number of physical CPUs now possible: 2
Number of logical CPUs now available: 2
Max number of logical CPUs now possible: 2
```

〔그림 9〕 운영체제 기본 정보 추출

추가할 수 있는 디바이스 드라이버 모듈 정보를 KEXT라고 한다. KEXT 정보는 커널 이미지의 '.g_kernel_kmod_info' 심볼과 '_kmod' 심볼에 위치하며 구조체 정보는 XNU 커널 소스의 헤더 정보인 'bsd/sys/kmod.h'에 명시되어 있다. '.g_kernel_kmod_info' 심볼은 커널 자신을 가리키는 정보이며, '_kmod'는 커널 기능 확장을 목적으로 애플 또는 타 어플리케이션의 모듈의 정보를 가지고 있다. Mac OS X의 악성코드는 자기 자신을 커널 레벨에서 동작 목적과 자신이 숨기고자 하는 정보의 은폐 목적으로 KEXT을 추가하고, 시스템 콜을 후킹하여 루트킷의 KEXT를 Mac OS X의 'kextstat'과 같은 명령어로부터 숨기는 행위를 한다. 물리 메모리에서 KEXT 정보를 링크드 리스트 체인을 따라 수집하면, 비정상적인 외부 모듈이 있는지 확인할 수 있다. 〔그림 10〕은 KEXT 정보를 물리 메모리 이미지에서 추출한 화면이다.

KEXT 구조체 정보에는 각 KEXT의 커널 가상 주소로 표현된 시작 주소와 크기 정보를 가지고 있어서 악성 KEXT라 판단되는 모듈을 메모리 이미지에서 추출하여 분석할 수 있음으로, 악성코드를 탐지하고 분석하는 행위를 물리 메모리 이미지만으로 수행할 수 있는 이점을 가진다.

단 물리 메모리 이미지에서 KEXT의 링크를 통해 수집하는 방법은 원하는 명령을 수행 후 링크드 리스트를 제거하여 자기 자신을 숨기는 경우에는 해당 KEXT를 찾을 수 없는 한계점이 존재한다〔9〕. 이 경우엔 뒷 절에 설명될 시스템 콜 후킹 탐지를 통해 일부 루트킷의 존재여부를 파악할 수 있다.

7.3 시스템에 마운트된 장치 정보

리눅스 및 유닉스 시스템은 저장장치를 시스템에 마운트시켜 동작한다. 마운트 정보는 커널 이미지의 '__common _mountlist' 심볼에 존재하며 구조체의 첫 번째 'LIST_ENTRY' 구조체를 통해 다음 마운트된 장치 정보를 획득할 수 있다. 〔그림 11〕은 물리 메모리에서 마운트된 장치 정보를 추출한 화면이다.

mount 구조체에는 이 외에도 저장 장치의 총 크기와 사용량 정보가 존재한다.

```
Memory Image: 34159efffe1a741e-00000000-100000000.memdump
Kernel Image: mach_kernel
Information: kext_info

-= Kernel Extentions(Kext) =-
kmod_info_ptr   info_version    id      name    version reference_count reference_list  a
57bb0d00        1       130     com.apple.iokit.IOFireWireIPPrivate     2.0.3   0       6
6cf6a680        1       127     com.apple.driver.AppleBluetoothHIDKeyboard      1.2.0a3 6
57b76b40        1       126     com.apple.driver.IOBluetoothHIDDriver   2.3.3f8 1       6
5832f960        1       125     com.apple.driver.AppleHIDKeyboard       1.2.0a3 1       6
583397c0        1       124     com.apple.driver.AppleHWSensor  1.9.3d0 0       08fb95e0
57c9f6c0        1       123     com.vmware.kext.vmnet   3.0.0   0       09001ec0        5
57c7ff80        1       122     com.vmware.kext.vmioplug        3.0.0   0       07afe440
6c482d20        1       121     com.vmware.kext.vmci    3.0.0   0       08fbea40        5
57c8c580        1       120     com.vmware.kext.vmx86   3.0.0   0       07af3900        6
57903620        1       119     com.apple.filesystems.autofs    2.1.0 1 0       07afa8c0
```

〔그림 10〕 KEXT 정보 추출

```
-= Mount List =-
list entry      fstypename      mount on name   mount from name
0x071ad290      hfs             /               /dev/disk0s2
0x071ac948      devfs           /dev            devfs
0x071ac000      autofs          /net            map -hosts
0x00000000      autofs          /home           map auto_home
```

〔그림 11〕 마운트 된 장치 목록 추출

7.4 프로세스 정보

커널의 기능 중 하나인 프로세스 관리를 위해서 커널은 프로세스 목록을 체인형태로 구성하여 관리하고 있다. Mac OS X의 프로세스 목록 정보는 XNU 소스의 'bsd/sys/proc_internal.h'에 명시된 proc 구조체 형태를 가지게 되며, 더블 링크드 리스트(Double Linked List)형태를 가지고 있다. 〔그림 12〕는 애플에서 공개한 XNU 커널 소스 중 'bsd/kern/bsd_init.c'의 내용 중 일부로, Mac OS X는 부팅 시 BSD의 프로세스 구조체 형태와 유사한 proc 구조체의 포인터 변수로 kernproc를 설정하고, 커널 초기화 과정에서 proc0을 proc구조체로 선언한다. kernproc 포인터 변수는 proc0의 주소 값을 저장하여 시스템 부팅 시 유동적으로 변할 수 있는 proc0의 주소를 참조한다. Mac OS X는 kernproc 변수를 이용하여 'kernel_task'라 명명된 운영체제의 첫 번째 프로세스를 생성하고 'kernel_task'정보는 proc0에 저장된다.

```
/* Initialize ... [illegible] ... before kmeminit()/bsd_autoconf()!)*/
tty_init();

kernproc = &proc0;  /* ... implicitly bzero'ed */

/* ... */
set_bsdtask_info(kernel_task,(void *)kernproc);

/* give ... name */
bsd_init_kprintf("calling process_name\n");
process_name("kernel_task", kernproc);
```

〔그림 12〕 첫 번째 프로세스 생성과정

커널 이미지에는 kernproc와 proc0의 심볼이 둘 다 존재하지만, proc0은 커널 이미지에 명시된 주소에 올라가지 않을 수 있기 때문에 Mac OS X의 접근 방식과 동일한 접근법을 사용해야 한다. kernproc를 통해 첫 번째 프로세스에 접근한 후 proc 구조체 내부의 더블 링크드 리스트 정보를 이용하여 프로세스 체인을 탐색하며 프로세스 목록을 추출할 수 있다. 〔그림 13〕은 메모리 이미지에서 프로세스 목록을 추출한 화면이다.

Mac OS X는 사용자 정보를 세션 단위로 관리하기 위해 session 구조체에 저장하고 있다. 그리고 프로세스들을 그룹별로 묶어 session 정보의 위치를 저장하게 된다. proc구조체는 프로세스 그룹 정보를 가진 pgrp 구조체의 포인터 정보를 가지고 있기 때문에

```
-= process list =-
list_entry      pid     ppid    process name            username
0x073fbd20      0       0       kernel_task
0x073fb2a0      1       0       launchdask              _spotlight
0x073fb7e0      10      1       kextddask               root
0x073fb540      11      1       DirectoryService                root
0x073fb000      12      1       notifydask              root
0x07777d20      13      1       diskarbitrationd                root
0x07777a80      14      1       configdask              root
0x077777e0      15      1       syslogdask              root
0x07777540      16      1       distnotedk              root
0x077772a0      17      1       blueddask               root
0x079a5d20      18      1       mDNSResponder           _mdnsresponder
0x079a5a80      20      1       coreservicesd           forensic
0x079a57e0      24      1       securitydk              _mdnsresponder
0x079a52a0      28      1       ntpdhdask               _mdnsresponder
0x079a5000      30      1       krb5kdcask              _mdnsresponder
0x08051a80      31      1       xgridagentd             _mdnsresponder
```

〔그림 13〕 프로세스 목록 추출

```
-= syscall list =-
number  sy_narg sy_resv sy_flags        sy_call_ptr     sy_arg_munge32_ptr
0       0       0       0               491c8f          0
1       1       0       0               4773de          4ef470
2       0       0       0               4790e0          0
3       3       0       0               495785          4ef490
4       3       0       0               494eac          4ef490
5       3       0       0               2ea3e6          4ef490
6       1       0       0               46b595          4ef470
7       4       0       0               476835          4ef4e0
8       0       0       0               491c8f          0
9       2       0       0               2e981e          4ef480
10      1       0       0               2e9884          4ef470
11      0       0       0               491c8f          0
12      1       0       0               2ea92f          4ef470
13      1       0       0               2ea9fe          4ef470
14      3       0       0               2e7766          4ef490
15      2       0       0               2e5ee2          4ef480
16      3       0       0               2e5e9a          4ef490
```

〔그림 14〕 시스템 콜 목록 추출

이 구조체를 이용해 session구조체의 포인터를 획득하여 프로세스를 실행한 사용자의 정보를 획득할 수 있다. 추가적으로 proc 구조체에는 프로세스 내부의 스레드 구조체 정보와 프로세스 구동시각, 오픈한 파일 포인터 정보를 담고 있기 때문에, 프로세스에 대한 세부적인 정보를 추출할 수 있다. 물리 메모리에서 프로세스 체인을 통한 프로세스 목록 수집은 루트킷의 프로세스를 숨기기 위해 유닉스 명령어인 ps, top에서 사용하는 'sysctl' 시스템 콜 함수를 후킹하여 프로세스를 숨기는 기법을 우회하여 수집할 수 있다 〔10〕.

7.5 시스템 콜 정보

Mac OS X는 유저 영역에서 커널 영역으로 진입하는 함수의 목록인 시스템 콜 테이블을 가지고 있다. 물리 메모리 이미지 상의 시스템 콜 테이블 정보는 〔그림 8〕의 정보를 이용해 찾을 수 있으며, 구조체 정보는 'bsd/sys/sysent.h'정보에 나타나 있다. 헤더 파일의 구조체는 22바이트로 되어 있지만 물리 메모리에서 추출 시에는 24바이트 단위로 추출해야 한다. 물리 메모리 이미지의 구조체가 헤더의 구조체와 다른 이유는 인텔 32비트 아키텍처에서 구조체를 성능의 목적으로 4바이트로 정렬하는 특징으로 인해 마지막에 2바이트의 패딩(padding)을 추가하기 때문이다. 〔그림 14〕는 물리 메모리 이미지에서 추출한 시스템 콜 정보이다.

7.6 시스템 콜 후킹 탐지 기법 소개

루트킷은 시스템 콜 정보를 후킹 해 실제 커널 이미지 상의 시스템 콜 정보와 물리 메모리에 존재하는 시스템 콜 테이블의 위치를 변경하여 주소가 변경된 함수 호출 시 자신이 원하는 일을 수행할 수 있도록 하는 시스템 콜 후킹 기법을 수행한다 〔11〕. 이러한 시스템 콜 후킹 기법은 운영체제가 실행 중인 환경에서 물리 메모리의 시스템 콜 테이블을 수정하기 때문에 추출한 시스템 콜 테이블의 주소 정보와 커널 이미지에 심볼로 존재하는 시스템 콜 목록을 비교하여, 심볼 정보에 존재하지 않는 시스템 콜 엔트리의 가상 주소가 존재하는 지 확인하는 방법으로 탐지할 수 있다. 〔그림 15〕는 〔그림 14〕의 시스템 콜 정보와 커널 이미지의 심볼 정보를 매칭시킨 화면이다.

VIII. 사례 분석

시스템 콜 후킹에 대한 올바른 탐지 여부를 판단하기 위해 'Snow Leopard Server'가 설치된 시스템에 자기 자신을 은닉하기 위한 기술로 사용하는 'File Hiding' 기법을 이용하는 KEXT 기반 루트킷을 로드하여 후킹 탐지 여부를 판단하는지 확인하였다.

커널 이미지에는 시스템 콜 함수에 대한 심볼 정보를 가지고 있으며, 심볼 정보 내에 존재하는 주소를 물리 메모리 주소로 변환하여, 시스템 콜 함수를 맵핑

```
-= syscall list =-
number  sy_narg sy_resv sy_flags        sy_call_ptr     sy_arg_munge32_ptr      sy_arg_munge64_ptr
0       0       0       0       _nosys  0       0       1       0       valid function
1       1       0       0       _exit   4ef470  0       0       4       valid function
2       0       0       0       _fork   0       0       1       0       valid function
3       3       0       0       _read   4ef490  0       6       12      valid function
4       3       0       0       _write  4ef490  0       6       12      valid function
5       3       0       0       _open   4ef490  0       1       12      valid function
6       1       0       0       _close  4ef470  0       1       4       valid function
```

〔그림 15〕 추출한 시스템 콜 목록과 커널 이미지의 심볼 정보 매칭

```
forensic-iMac:case n0fate$ python volafox.py -i Mac\ OS\ X\ Server\ 10.6.vmem -s mach_kernel -o syscall_info
 | grep hooking
222     8       0       0       1b1b7168        4ee5a0  0       1       32      syscall hooking possible
344     4       0       0       1b1b7309        4ee560  0       6       16      syscall hooking possible
```

〔그림 16〕 후킹된 시스템 콜 목록

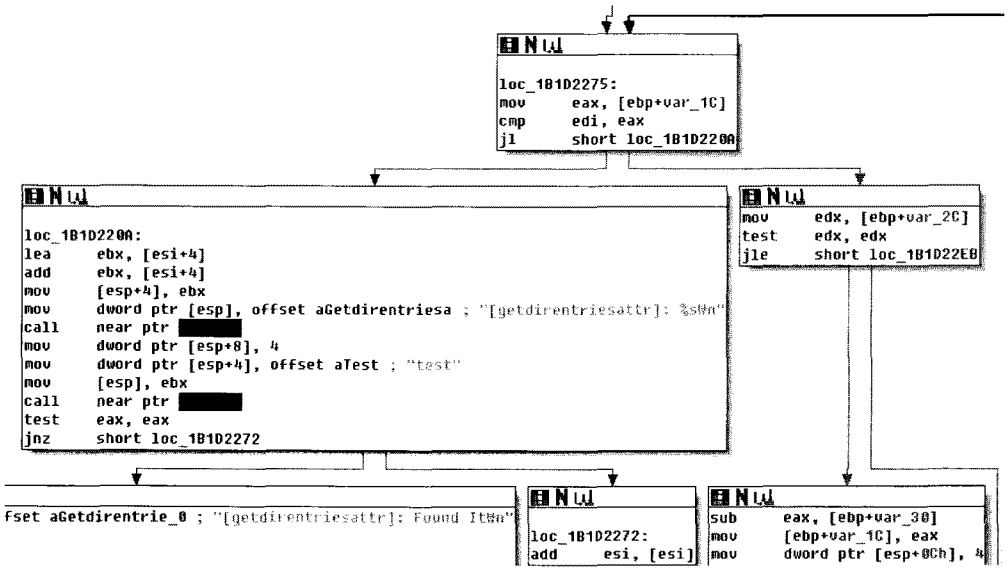
한다. 악의적인 사용자는 시스템 콜 후킹을 시도할 때 물리 메모리의 시스템 콜 엔트리 중 후킹을 원하는 시스템 콜을 찾는다. 시스템 콜을 찾으면, 구조체 내의 'sy_call_t'를 찾아 그 함수의 주소를 공격자가 정의한 함수로 변경하여 자신이 원하는 일을 수행할 수 있게 한다.

파일 은닉을 위해 루트킷은 'getdirentriesattr' 시스템 콜과 'getdirentries64' 시스템 콜을 자신이 정의한 함수를 가리키도록 수정하고, 내부적으로 원래 시스템 콜 함수를 호출하여, 리턴된 결과를 조작하는 행위를 한다[9].

해당 루트킷을 로드한 후 물리 메모리 이미지를 수집하여, 앞서 제시한 탐지 방안을 적용하였고 〔그림 16〕과 같은 결과를 확인하였다. 두 시스템 콜 함수의 주소가 커널 이미지 심볼 정보의 가상 주소와 일치하지 않음으로 인해 시스템 콜 함수 명이 보이는 것이 아닌 변형된 함수의 주소가 보이는 것을 확인할 수 있다. 루트킷이 KEXT로 로드되었기 때문에, KEXT 구조체 내에 존재하는 KEXT의 가상 시작 주소와 크기 정보를 이용하여 로드된 KEXT 정보를 추출하여, 〔그림 17〕과 같이 IDA를 통해 KEXT를 분석할 수 있다. 물리 메모리에서 덤프한 KEXT는 호출 함수의 주소가 물리 메모리에서의 주소 값을 가지기 때문에 커널 이미지의 심볼에 존재하는 시스템 콜 함수 명과 시스템 콜 함수 주소를 비교하여 어떠한 함수를 호출하는지 확인할 수 있어서 디스크에 있는 KEXT 파일과 동일한 분석을 수행할 수 있다. 만약 악성 루트킷이 KEXT를 숨기는 기법을 이용하여 자기 자신을 숨기더라도, 변경된 시스템 콜 주소를 기준으로 물리 메모리 페이지 프레임(Page Frame) 단위로 덤프하거나, 변경된 시스템 콜 이전 확인하여, KEXT의 시그너처가 나올 때 까지 덤프하는 방법으로 바이너리 데이터에 대한 분석을 수행할 수 있다.

IX. 결 론

Mac OS X의 사용률이 점점 높아지는 상황에서 윈도우 환경에서 사용되던 포렌식 기술을 Mac OS X에 적용하고 Mac OS X에만 존재하는 새로운 항목을 연구할 필요가 있다. 물리 메모리 분석은 자기 자신을 은닉하는 기법을 많이 사용하는 악성코드와 루트킷을 탐지할 수 있는 가장 효율적인 방법이며, 커널의 구조



〔그림 17〕 디스어셈블러를 이용하여 덤프한 KEXT 분석

체 정보를 수정하는 행위를 하는 루트킷과 같은 프로세스 또한 물리 메모리에 위치해야하기 때문에 디지털 포렌식 관점에서 꼭 분석해야하는 정보이다. 본 논문에서는 물리 메모리이미지에서 수집한 Mac OS X의 기본적인 정보 분석 및 프로세스를 숨기는 악성코드를 찾아내거나 악의적인 커널 모듈을 판단하는 방법을 제공하며, 물리 메모리 분석을 통해 시스템 콜 후킹을 탐지하는 방법을 제안하고 물리 메모리 분석 도구를 구현하였다. 도구의 루트킷 탐지 능력을 판단하기 위해 Mac OS X의 루트킷이 자주 사용하는 KEXT를 이용하여 시스템 콜을 후킹하는 악성코드의 후킹 여부를 판단하고 KEXT 덤프를 통해 물리 메모리 분석이 가능함을 보였다. 아직 도구의 기능이 물리 메모리에서 추출할 수 있는 모든 정보를 추출한 것이 아니기 때문에 네트워크 정보 분석과 같은 내용을 추가하여 도구를 재구성 할 것이며, KEXT 및 프로세스 체인을 제거하여 자기 자신을 은닉하는 루트킷의 탐지 방안을 추가적으로 연구할 것이다.

참고문헌

[1] Becher, M, Dornsief, M., and Klein, C.N., "Firewire: all your memory are belong to us," CansetWest 2005, 2005, (cansecwest.com/core05/2005-firewire-cansecwest.pdf).

[2] Amit Singh, "Accessing Kernel Memory on the x86 Version of Mac OS X, 2006", Mac OS X Internals, 2006, (bonus chapter: http://www.osxbook.com/book/bonus/chapter8/kma/).

[3] Apple, Apple Computer, "Mac OS X Security," pp.13, 2009.

[4] Apple, Apple Computer, "Mac OS X ABI Mach-O File Format Reference," Mac OS X Reference Library, pp.7-10, Feb. 2009.

[5] Intel, Intel, "System Programming Guide, Part1," Intel 64 and IA-32 Architectures Software Developer Manual Volume 3A, pp116-131, Aug. 2007.

[6] Apple Computer, Apple Open Source, http://www.opensource.apple.com/

[7] Python, Python Package: macholib, http://pypi.python.org/pypi/macholib/

[8] Walters, A, Petroni Jr and N. L, "Volatools: Integrating Volatile Memory Forensics into the Digital Inverstigation Process" Black Hat DC, 2007, (www.blackhat.com/presentations/bh-dc-07/Walters/Paper/bh-dc-07-Walters-WP.pdf)

[9] Miller, C. and Dino Dai Zovi, D.A. Mac hacker's handbook, Wiley, pp.342-345, 2009.

[10] Erikson, B., "Runtime Kernel Patching on Mac OS X," Defcon 17, 2009, (http://defcon.org/images/defcon-17/dc-17-presentations/defcon-17-bosse_eriksson-kernel_patching_on_osx.pdf).

[11] wowie, "Developing MacOSX Kernel Rootkits," Phrack 66, 2009, (www.phrack.org/issues.html?issue=66&id=16).

〈著者紹介〉

이 경 식 (Kyeongsik Lee) 학생회원
2009년 2월: 세종대학교 컴퓨터공학과 졸업
2009년 3월~현재: 고려대학교 정보경영공학전문대학원 석사과정
〈관심분야〉 디지털 포렌식, 리버스 엔지니어링, 퍼징

이 상 진 (Sangjin Lee) 정회원
1987년 2월: 고려대학교 수학과 졸업
1989년 2월: 고려대학교 수학과 이학석사
1994년 2월: 고려대학교 수학과 이학박사
1989년 10월~1999년 8월: ETRI 연구원
2001년 9월~현재: 고려대학교 정보경영공학전문대학원 교수
〈관심분야〉 디지털 포렌식, 모바일 포렌식, 심층 암호, 해쉬 함수