리눅스 시스템의 메모리 포렌식 : 난제와 발전방향

박재유*, 조슈아 제임스**

*한국과학기술원 소프트웨어대학원, **한림대학교 국제학부

Linux Memory Forensic Analysis : Challenges and Future Directions

Jae-You PARK*, Joshua I. James**

*KAIST Software Graduate Program, **Hallym University

요 약

활성 시스템의 침해사고 대응 상황에서 물리 메모리를 덤프하여 분석하는 '메모리 포렌식' 기법에 대한 연구가 널리 진행되고 있으며, Volatility[6]와 Rekall[8] 등의 오픈소스 분석 도구들이 디지털 포렌식 실무에서 사용되고 있다. 한편, 리눅스 시스템은 매우 다양한 종류의 Distro가 존재하고 개인이 커스터마이징한 커널 또한 상당히 많으므로 분석을 위한 프로파일 추출이 윈도우 등의 다른 운영체제에 비해 상당히 곤란하다. 뿐만 아니라 메모리 조작을 방어하기 위한 수단으로 도입한 다양한 보안 기능들이 오히려 포렌식 분석을 어렵게 만드는 부작용을 초래하고 있다. 이에 본 논문에서는 활성 시스템의 침해사고 대응 관점에서 리눅스의 메모리를 추출하고 분석할 때에 직면하게 되는 몇 가지 난제를 언급하고, 최근 연구동향 분석을 통해 이를 해결할 수 있는 방안을 모색한 뒤 실험한 결과를 제시한다. 또한, 리눅스 시스템에서의 메모리 포렌식의 향후 발전 방향을 제안하고자 한다.

I. 서론

기존의 디지털 포렌식은 대부분 컴퓨터 시스템의 전원을 차단하고 HDD 등 전자적 저장매체를 분리하여 분석하는 방식으로 이루어지고 있었다. 그러나 이러한 오프라인 대응방식은 최근 유행하는 Fileless Malware등에 의한 침해사고에서 무용지물이 된다는 문제점이 있다. 이러한 종류의 악성코드는 대상 시스템의 하드 드라이브에 어떠한 디렉터리나 파일도 남기지 않은 채, 실행 중인 프로세스의 메모리에 삽입되어 RAM에 상주하도록 설계되어 있다. 따라서 이러한 종류의 사이버범죄에 대응하기 위해서는 필수적으로 활상 상태에서 메모리를 추출한 후 분석해야만 한다. 현재는 통상적으로 이러한 기법을 일명 '메모리 포렌식(Memory Forensic)'으로 지칭하며, 2004년경 최초의 기반기술 개발을 기점으로, 이후 많은 연구가 진행되고 있다.

일반적으로 메모리 내에 데이터가 저장되는 구조는 해당 시스템의 아키텍처 및 운영체제에 따라 상이한 방식으로 배치된다. 기본적으로 Windows, Linux, macOS 등으로 크게 구분되고, 같은 계열 안에서도 배포판 종류 및 Kernel 버전에 따라 달라진다. 이처럼 다양한 종류의 시스템은 저마다 고유한 메모리 레이아웃을 가지고 있으므로, 분석하려는 시스템의 구체적인 Specification 정보를 파악하는 것이 가장 먼저 선행되어야 하며, 이를 통해 OS에 대한 프로파일링 작업을 수행함으로써 메모리로부터 유의미한 정보를 식별할 시금석을 마련할 수 있다.

현재까지의 관련 연구를 살펴보면, Windows 시스템에 대한 프로파일 분석 작업은 상당히 높은 완성도를 갖추고 있다. 그러나 Linux 및 macOS(Unix) 등의 진영은 비교적 그 진척의 속도가 늦고, 공개되어 있는 프로파일도 제한적이어서 분석 작업에 어려움이 많다. 본 논문에서는 특히 Linux에 초점을 맞추어, 해당 환경에서의 메모리 포렌식을 어렵게 하는 난제를 논의하고, 이를 개선할 방향을 모색하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 메모리 포렌식에 대한 배경지식을 살펴보고, 관련 도구 동향을 파악한 후, 작업수행에 필요한 절차와 각 운영체제별 특이점을 다룬다. 3장에서는 특히 리눅스 시스템에 해당되는 난제들을 살펴보고, 4장에서는 이를 해결하기 위한 발전 방향을 제시한다. 5장에서는 결론을 맺는다.

II. 배경지식

2.1 기존 연구

Mariusz Burdach는 Black Hat 2006 컨퍼런스에서 Physical Memory를 분석함으로써 안티 포렌식에 대응할 수 있음을 주장하였고[1], 이후 Kristine Amari 등은 RAM 내부의 휘발성 메모리를 추출하고, 그 덤프를 분석함으로써 각종 데이터 복원, 악성코드의 행위 및 시스템의 변경 내역을 추적할 수 있음을 발표하였다.[2]

지금까지의 연구는 대부분 Windows 계열 운영체제에 초점이 맞추어져 있었고, 상대적으로 Linux 관련 연구는 진척이 더딘 상황이다. 하지만 일반 사용자의 PC환경을 제외하면 기업의 서버 시스템의 경우 리눅스 활용 빈도가 월등히 높으며, 최근 IoT 기반 저전력 임베디드 시스템 구동을 위한 리눅스 디바이스의 사용량이 급증하고 있기에 이를 타깃으로 한 다양한 악성코드 역시 다수 출현하고 있다. 그러므로 리눅스 기반 환경에서의 악성코드 침해사고에 대한 포렌식 분석 연구 필요성을 결코 무시할 수 없는 상황이다.[3]

2.2 휘발성 메모리 추출

메모리 포렌식을 성공적으로 수행하기 위해서는 라이브 시스템에서 휘발성 데이터를 안정적인 방법으로 추출(Acquisition)하는 작업이 가장 핵심적인 관건이다. 과거 고전적인 방법으로 Windows에서는 \Device\PhysicalMemory을 주로 사용하고, Linux 및 Unix에서는 /dev/mem을 통해 OS 자체적으로 지원하는 메모리 덤프 기능을 사용할 수 있었다. 그러나 멀웨어 역시 이러한 기능을 악용할 수 있다는 보안상의 이유로, 일부 기능을 제한하거나 반드시 최고관리자(root)로의 권한을 요구하는 방식으로 점차 변경되고 있다. 이에 따라 포렌식 수사를 위한 전용 메모리 덤핑 도구들이 필요하게 되었다.

이후 /proc/kcore, FMEM 등의 도구들이 출시되었지만, 편리성과 유용성 측면에서 현재 가장 각광받고 있는 도구는 LiME으로 알려져 있다.[4] LiME은 데이터 처리 시 사용자 영역과 커널 부분을 독립적으로 전달하는 방식을 사용하여 안정성을 크게 향상시켰고, 안드로이드 등의 리눅스 기반 모바일 운영체제에서도 상당히 높은 정확도로 작동하기 때문에 리눅스 시스템의 메모리 포렌식 분석에서 가장 권장된다.[5]

2.3 메모리 덤프의 분석

덤프 된 메모리 분석(Analysis)을 통해 유의미한 정보를 추출하고, 대상 시스템에서 발생한 침해사고 상황을 역추적할 수 있게 된다. 리눅스 환경의 메모리 포렌식을 위한 도구로는 Volatility[6][7], Rekall[8] 등의 오픈소스가 널리 활용되고 있고, 기업을 위한 상용 라이센스 제품인 Forcepoint Threat Protection for Linux도 있다.

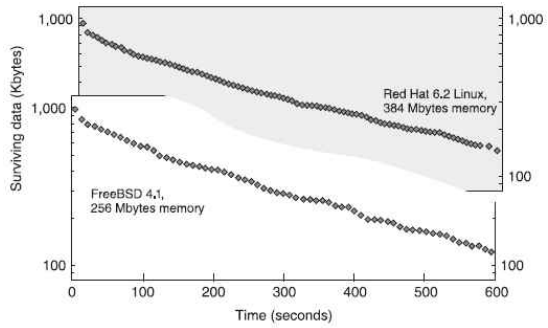
III. 난제

본 단원에서는 특히 리눅스 시스템에서 포렌식을 수행할 때 발생하는 난제들을 논한다.

3.1 신속성 확보 및 원본 동일·무결성 입증

RFC 3227 - 증거 수집과 보관에 관한 지침[9]이 제시하고 있는 휘발성 순서(Order of Volatility)에 따르면, 레지스터와 캐시가 1순위, 네트워크 연결 정보와 프로세스 목록 등의 정보가 2순위 등으로 가장 높은 순위를 차지하고 있으며, 이러한 휘발성이 높은 데이터들은 대부분 메모리에 적재되어 있다. 기타 파일 시스템이나 디스크 관련 정보는 3순위 이하에 불과하다. 때문에 기존의 디스크 포렌식에 비해 메모리 포렌식 대응에서는 높은 신속성이 요구된다. 이와 관련하여 Dan Farmer의 실험 결과[10]를

보면, [그림 1]과 같이 시간이 흐를수록 휘발성 메모리의 내용이 급격히 손실된다. 즉, 침해사고 발생으로부터 시간이 많이 흘렀다면 이미 다른 데이터가 덮어써서 침해 상황 당시의 내용을 정확히 복원하지 못할 가능성이 크다.



[그림 1]종료된 프로그램의 메모리 소실[10]

또한, 디지털 포렌식과 관련한 국내 대법원의 판례를 살펴보면, 형사소송법상의 증거 규정에 따라 '동일성', '무결성' 등을 입증할 수 있어야 한다고 해석하고 있는데[11], 이는 종전의 디스크 포렌식 관점에서는 적용이 되더라도, 휘발성 메모리의 내용은 계속해서 변경되는 속성 때문에 재현이 불가능한 문제점이 있다. 동일한 시스템에서 메모리를 두 번 추출하더라도 그 내용은 필연적으로 판이하게 달라질 수밖에 없기 때문이다. 따라서 메모리 분석 결과물의 법의학적 신뢰성 입증에는 추가적인 법제 및 절차적 검토가 보완되어야 할 것으로 보인다.

3.2 커널 별 상이한 프로파일

Volatility 또는 Rekall Framework를 사용하여 메모리 덤프를 분석할 때에는 각 커널에 적합한 프로파일이 있어야만 원활한 메모리 분석이 가능하며, 따라서 해당 운영체제의 커널 프로파일 생성 작업이 반드시 선행되어야 한다. 이러한 프로파일을 구성하는 요소는 해당 커널의 데이터 구조를 정의한 vtypes 파일과, Symbol 정보를 내포하고 있는 System.map 파일이다. 이를 zip으로 압축하여 볼라틸리티의 플러그인 디렉터리에 저장해두면 분석 시 대상 시스템을 인식할 수 있게 된다.[7]

윈도우 운영체제의 경우 이미 많은 연구가 진행되었으며 프로파일들이 상당히 정형화되어 있기 때문에 사용자가 별다른 작업을 하지 않아도 기본 설정을 통해 편리하게 작업할 수 있다. 그러나 리눅스 환경의 경우 오픈소스의 특징상 매우 다양한 종류의 Distro가 존재하며, 그 하위에서도 서브 커널 버전들과 사용자가 직접 튜닝한 커스텀 버전들이 즐비하다. 그나마 대중적으로 가장 널리 사용되고 있는 Ubuntu, Debian, RedHat, CentOS 및 OpenSuSe 등의 경우 볼라틸리티 제조사에서 공식적으로 프로파일을 제공하고 있으나[12], 대부분의 경우에는 분석가가 해당 시스템과 정확히 일치하는 프로파일을 직접 생성해야만 한다. 만약 vtypes와 System.map 파일이 존재하지 않는 시스템일 경우 직접 커널의 모듈을 빌드해야 하므로 그 과정이 상당히 어렵고, 무엇보다 분석가가 해당 시스템을 직접적으로 임의 조작할 경우 무결성 측면에서 손상을 입힐 위험이 크다.

3.3 커널 메모리 보호 기법

운영체제 시스템에서 커널 내부의 주소 위치를 악용하는 취약점이 대두되면서, 관련된 보안 패치가 출시되기 시작하였다. 특히 메모리 변조 공격에 대응하기 위한 ASLR(Address Space Layout Randomization)을 단순히 User영역이 아닌 Kernel차원에서 적용한 KASLR이 2013년 리눅스 보안 컨퍼런스에서 제안되었다.[13] 이후 실제로 구현되어 2014년 발표된 Kernel 3.14 버전부터 포함되었으며, Distribution별로 선택적으로 적용해왔다. 최근 출시되는 시스템에서는 대부분 Default로 채택되어 설치 즉시 작동되고 있다. [표 1]은 대표적인 Ubuntu Linux의 Release 버전에 따른 KASLR 사용 현황이다.

[표 1] Ubuntu Kernel에 따른 KASLR 적용여부

Version	Kernel	KASLR
12.04 LTS	3.11	Unsupported
14.04 LTS	3.13	Unsupported
15.04	3.19	Optional
16.04 LTS	4.4	Optional
16.10	4.8	Default
17.04	4.10	Default

해킹에 의한 커널 메모리 부정 조작을 방지하기 위한 방어기법으로써 도입된 KASLR은 한편으론 메모리 포렌식 분석을 난해하게 하는 부작용을 초래하고 있다. 실제로 Volatility를 활용하여 Kernel 4.8 이상의 리눅스 배포판(Ex: Fedora 25, Ubuntu 16.10 등)에서 추출된 메모리 덤프 파일을 분석하려고 하면, 프로파일을 올바르게 생성하였음에도 불구하고 오류가 발생하는 현상을 겪게 된다. 이는 KASLR에 의해 메모리 공간의 주소가 난독화되었기 때문에 정확한 메모리 시작 위치를 찾을 수 없어서 발생하는 문제이다. 실제로 각각의 프로세스가 같은 Driver를 호출하더라도, Kernel space에 로드되는 위치가 모두 달라지기 때문에 시작 주소와 끝 주소를 예측하여 분석하기가 어렵다.

IV. 발전방향 제시

본 단원에서는 3장에서 제기하였던 상황들 중 법제·정책적인 이슈는 일부 논외로 하고, 기술적인 관점에서의 난제 해결방안을 제시한다.

4.1 리눅스 프로파일 오픈 데이터베이스 구축

리눅스 시스템 환경의 특성상, Distro 종류가 상당히 많고 각각이 채택한 Kernel과 모듈 버전에 따라 프로파일이 달라지므로 이러한 다양한 경우의 수를 충족할 수 있는 방대한 양의 데이터베이스 구축이 시급하다. 그러나 모든 종류의 시스템을 직접 구축하여서 각각의 프로파일 정보를 추출하는 작업은 비용과 시간 측면에서 소모가 클 것이다. 이러한 과정을 간소화하기 위해서는 클라우드 컴퓨팅 환경의 IaaS를 적극 활용하여 인프라 구축 비용을 절감하고 가상머신 환경에서 다수의 운영체제 및 커널을 신속하게 설치함으로써 효율을 극대화할 수 있다. 또한 Osboxes.org 프로젝트에서 제공하는 가상머신 이미지를 사용하면 설치과정 역시 상당 부분 생략할 수 있어서 시간을 굉장히 절약할 수 있다. 단, 실험 결과에 따르면 KVM, VirtualBox 등의 일부 가상머신 소프트웨어의 경우 호스트 컴퓨터와 VM 인스턴스 사이에 통신 지연이 발생하여 메모리 페이징 정보가 손상되는 사례가 다수 발생하였다. 이러한 오류는 timeout을 제한하는 방식으로 갈음할 수는 있지만 메모리의 무결성을 훼손할 우려가 있으므로 되도록 클라우드 환경이 제공하는 가상머신 소프트웨어의 오작동이 없는지 충분히 점검하여야 한다. 본 논문의 실험에서는 Amazon EC2 Instance를 활용하였으며 해당 시스템의 스냅샷을 Export 한 후 로컬 시스템에서 VMware Workstation으로 로드한 후 재검증하였을 때 프로파일의 해시값이 일치하는 것을 확인함으로써 무결성을 보장할 수 있음을 입증하였다.

다음으로 VM환경에 리눅스 운영체제를 설치하고, 해당 커널의 모듈과 버전에 따른 헤더를 추출해야 하는데, 현재로써는 lmg(Linux Memory Grabber) 오픈소스 프로젝트[14]를 활용하는 방안이 가장 적절한 방안으로 보인다. lmg는 본래 USB 등의 이동형 저장 디스크를 활용하여 다수의 시스템에 마운트하려는 목적으로 개발되었으나, 가상머신 인스턴스에 직접 설치하는 방식으로 테스트한 결과 정상 작동하는 것을 확인하였다. 특히 dwarfdump[15]를 정적으로 링크한 프리컴파일된 32bit 버전과 64bit 버전 모두를 탑재하였기 때문에 시스템에서 별도로 빌드 작업을 수동으로 실시할 필요가 없으며, 대부분의 작업을 bash 쉘 스크립트로 완전히 자동화하여 처리할 수 있다는 것이 큰 장점이다. 작업이 완료되면 module.dwarf 파일과 System.map을 압축한 zip 파일이 생성되며, 이것을 해당 시스템의 프로파일로써 고유하게 식별할 수 있도록 Hash 값을 계산한 후 파일과 함께 데이터베이스 서버로 전송하여 저장한다.

[표 2]는 프로파일 데이터베이스에 저장한 항목들을 예시로 보여주고 있다. 각각의 Distro 버전과 Kernel 버전을 명시하고, 해당 시스템에서 추출된 프로파일 해시값을 저장한다. 프로파일 압축파일은 별도의 파일 업로드 기능을 통해 저장되며, 별도의 웹 페이지에서 DB의 정보와 매칭하여 원하는 프로파일을 조회할 수 있도록 하고 필요시 다운로드하도록 한다. 해당 데이터베이스의 관리를 오픈소스화하여 포렌식 커뮤니티에서 자유롭게 기여할 수 있도록 공개하면 아직 알려지지 않은 시스템에 대한 프로파일 수집이나 빠른 업데이트가 가능할 것이다.

[표 2] 리눅스 프로파일 데이터베이스 구조설계 및 데이터생성 예시

Creator	Date	Bit	Distro	Kernel	Profile Hash (SHA-1)
iMHLv2	20140731	32	Fedora 20	3.11	756eb692454efc1881ea39cbbe7044e08f60f6f7
adamziaja	20160801	64	Debian 8.6	3.16	e0432faf9147d3b5904f6638c1566d7ccd370e54
Lukach	20160102	64	Ubuntu 15.10	4.2	37aa0ca82d849f210dd515010feb4c67bd1c965d
cpuu	20170510	64	Ubuntu 14.04	4.4	d8426f917e9007f35646167a0e52cc18f7ba4db1
cpuu	20170511	64	Ubuntu 16.04	4.8	7a678e1d180604bcdb63ad8659e738c098664a89
cpuu	20170515	64	Kali 2017.01	4.9	095718fdc826e5edfe963b041bf41021da39dd97

4.2 커널 베이스 주소 산출 플러그인 개발

KASLR 적용으로 인한 난제는 다른 운영체제에서도 동일하게 발생한다. 이미 Windows 운영체제의 경우 2007년 출시된 Windows Vista에서부터 KASLR이 적용되었음에도 불구하고 Volatility에서 해석이 가능한 이유는 KDBG(KdDebuggerDataBlock)을 통해 데이터 구조를 분석하면 실행 중인 프로세스 목록이나 커널 모듈들의 정보를 쉽게 추출할 수 있기 때문이다. 이러한 기능을 활용하여 구현된 플러그인이 kdbgscan인데, 미리 수집된 약 10~14 Bytes 길이의 Signature와 대조하여 메모리 덤프를 조사하면 이에 상응하는 해당 OS Kernel 버전을 판별할 수 있고, 그에 적합한 De-randomize기법을 적용할 수 있도록 돕는다. 그러나 kdbgscan 플러그인은 아직까지 리눅스 환경은 지원하지 않고 있다. 그러므로 리눅스 환경에서 이와 유사한 접근법을 수행하는 플러그인을 구현하는 것이 시급하다.

Gu Yufei은 가상머신 환경에서의 32bit의 리눅스 3.14 ~ 4.0 사이의 버전 20개를 기준으로 커널의 코드와 데이터를 분석하여 각각을 구분할 수 있는 고유한 Signature를 추출하고 탐지하는 연구를 최초로 수행한 바 있다.[16] 해당 실험에서 우수한 확률로 각각의 Kernel 버전을 식별할 수 있음을 보였다. 그러나 생성된 Signature를 이용하여 실제 메모리 포렌식 분석을 수행하는 구체적인 방안은 제시하고 있지 않기에, 해당 연구내용을 바탕으로 하여 베이스 주소를 계산하는 실제 구현을 수행하였다.

한편, macOS는 2012년 10.8(Mountain Lion) 버전부터 KASLR이 적용되었으며 전용 메모리 포렌식 도구인 volafox[17]에서 커널의 베이스 주소를 계산하는 기능을 이미 적용한 바 있다.[18] volafox의 방법은 페이지의 테이블 베이스 주소를 기준으로 설정하고 메모리에서 추정한 페이지 테이블 주소과 비교하여 가감하는 방식으로 Randomize된 Offset을 배제하고 원래의 커널 베이스 주소를 산출하는 것이다. 이 점에 착안하여 이와 유사한 해법을 Linux 시스템에서 적용할 수 있는지 실험하였다. [표 3]은 각각 Ubuntu 14와 16, Kali 2017 버전에서 테스트한 데이터셋이며, KASLR이 적용되지 않은 Kernel 4.4의 경우에는 물리 메모리와 가상메모리에 Shift가 발생하지 않았으나, KASLR이 작동된 4.8 이상의 커널에서는 Shift가 산발적으로 발생하였다. 해당 Offset 값을 저장했다가, 이후 다른 플러그인을 사용할 때에 Address를 얼마만큼 이동시켜야 원하는 데이터 값이 위치한 곳으로 접근할 수 있는지와, DTB(Directory Table Base)를 함께 명시적으로 지정하면 플러그인을 정상 작동시킬 수 있음을 입증하였다. 이처럼 Physical주소와 Virtual주소 사이의 Shift 연산을 자동으로 수행해주는 리눅스 전용 KASLR SHIFT 플러그인을 개발할 수 있었다.

[표 3] Kernel별 KASLR Shift 계산 수치비교

	Physical	Virtual	DTB
4.4	0x0	0x0	0x1c0c000
4.8	-0x29c00000	0x39200000	0x11406000
4.9	-0x2a400000	0x37000000	0xe607000

하지만, 이러한 기능은 다른 플러그인까지 전역으로 적용되어야 작동하므로 Volatility 차원에서도 이를 보완할 통합 업데이트가 요구되는 상황이다.

V. 결론

본 논문에서는 활성 시스템의 침해사고 대응 관점에서 리눅스의 메모리를 추출하고 분석할 때에 직면하게 되는 두 가지 난제를 언급하고, 최근 연구동향 분석을 통해 이를 해결할 수 있는 방안을 제시한 후 실험하였다.

다른 운영체제에 비해 리눅스 특유의 개방성과 확장성 때문에 발생하는 프로파일 제작의 어려움을 해결하기 위해서는 오픈소스 커뮤니티 방식으로 커널 프로파일에 대한 정보를 공유할 수 있는 체계가 구축되어야 할 것이며, 이를 위한 Mock-up을 제시하였다.

또한, 커널 메모리 보호 기법을 인지하고 Address를 Shift하여 Base 주소를 찾는 플러그인을 개발하였으며, 이를 적용할 분석도구의 업데이트가 요구된다.

그러나 이와 같은 방법만으로는 디지털 포렌식 수사 실무에서 폭넓게 받아들여지기에는 아직 남겨진 과제가 많다. 향후 대검찰청에서 “디지털수사통합업무관리시스템” 등과 연동할 수 있으려면 프로파일 검증 메커니즘 등이 추가적으로 필요할 것이다.

[참고문헌]

[1] Burdach, Mariusz. "Physical memory forensics." Black Hat USA (2006).

[2] Amari, Kristine. "Techniques and tools for recovering and analyzing data from volatile memory." SANS Institute (2009).

[3] 주한익. “리눅스 물리메모리 덤프파일을 이용한 심볼정보 재구성 포렌식 도구 구현.” 서강대학교 정보통신대학원 학위논문(석사), 2016.

[4] LiME : Linux Memory Extractor, 2012. https://github.com/504ensicsLabs/LiME

[5] Sylve, Joe, et al. "Acquisition and analysis of volatile memory from android devices." Digital Investigation 8.3 (2012): 175-184.

[6] The Volatility Framework: Volatile Memory Artifact extraction Utility Framework, 2016. https://github.com/volatilityfoundation/volatility

[7] Ligh, Michael Hale, et al. “The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory.” John Wiley & Sons, 2014.

[8] Google, Rekall Memory Forensic Framework, 2016. https://github.com/google/rekall

[9] Brezinski, D., and Tom Killalea. “Guidelines for evidence collection and archiving. No. RFC 3227.”, 2002.

[10] Farmer, Dan, and Wietse Venema. Forensic discovery. Vol. 6. Upper Saddle River: Addison-Wesley, 2005.

[11] 대법원 2007. 12. 13. 선고 2007도7257

[12] Volatility profiles for Linux and Mac OS X, 2014, https://github.com/volatilityfoundation/profiles

[13] Cook, K. "Kernel address space layout randomization, 2013." Linux Security Summit.

[14] Hal Pomeranz. "Automating Linux Memory Capture." SANS DFIR Summit, 2014.

[15] Anderson, D. "Libdwarf and dwarfdump." (2011).

[16] Gu, Yufei, and Zhiqiang Lin. "Derandomizing kernel address space layout for memory introspection and forensics." Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. ACM, 2016.

[17] Volafox memory analysis framework, 2015, https://github.com/n0fate/volafox.

[18] 이경식. "Mac OS X의 물리 메모리 분석 연구." 고려대학교 정보경영공학전문대학원 학위논문(석사), 2010.

[19] Andrew Case, et al. "Memory forensics: The path forward," Digital Investigation, Volume 20, 2017.