Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.4, Aug. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.4.775

클라우드 보안성 강화를 위한 연산 효율적인 인스턴스 메모리 모니터링 기술*

최 상 훈,[1†] 박 기 웅[2‡]
[1]세종대학교 정보보호학과 시스템보안연구실, [2]세종대학교 정보보호학과

Computationally Efficient Instance Memory Monitoring Scheme for a Security-Enhanced Cloud Platform*

Sang-Hoon Choi,[1†] Ki-Woong Park[2‡]
[1]Sejong Univ. SysCore Lab.,
[2]Sejong Univ. Dept. of Computer and Information Security

요 약

클라우드 컴퓨팅 기반의 인프라 구축이 활성화됨에 따라, 안전성과 보안성이 강화된 클라우드 구축을 위한 기술이 큰 화두로 인식되고 있다. 이에 대한 방안으로써, 클라우드 사용자 인스턴스의 시스템 보안 강화를 위한 다양한 보안 솔루션이 등장하고 있다. 특히 인스턴스(가상머신)의 메모리 분석을 통한 악성코드 분석 및 탐지에 관한 연구가 활발히 진행되고 있다. 하지만 메모리 분석을 통한 보안 모니터링 기술은 메모리 덤프 시 수반되는 연산 오버헤드로 인해 다수의 인스턴스가 하나의 물리적 서버 노드에서 구동되는 클라우드 플랫폼과 같은 환경으로의 적용에 어려움이 있어왔다. 본 논문에서는 메모리 덤프 시 발생하는 오버헤드를 최소화하기 위해 악성코드 분석 및 탐지에 필요한 인스턴스 메모리의 특정 부분을 모니터링 하는 기술을 제안하고, 부분 메모리 모니터링 기반 악성코드 탐지 시스템을 통해 제안 기술의 실효성을 검증한다.

ABSTRACT

As interest in cloud computing grows, the number of users using cloud computing services is increasing. However, cloud computing technology has been steadily challenged by security concerns. Therefore, various security breaches are springing up to enhance the system security for cloud services users. In particular, research on detection of malicious VM (Virtual Machine) is actively underway through the introspecting virtual machines on the cloud platform. However, memory analysis technology is not used as a monitoring tool in the environments where multiple virtual machines are run on a single server platform due to obstructive monitoring overhead. As a remedy to the challenging issue, we proposes a computationally efficient instance memory introspection scheme to minimize the overhead that occurs in memory dump and monitor it through a partial memory monitoring based on the well-defined kernel memory map library.

Keywords: Cloud Computing, VM Introspection, Computational Efficiency

Received(03. 15. 2017), Modified(06. 19. 2017), Accepted(06.19.2017)

* 본 연구는 한국연구재단 연구과제(NRF-2017R1C1B2003957) 지원으로 수행하였습니다.

* 본 논문은 2016년도 동계 학술대회에 발표한 우수논문을 개선 및 확장한 것임

† 주저자, csh0052@gmail.com

‡ 교신저자, woongbak@sejong.ac.kr(Corresponding author)

I. 서 론

최근 공공기관 및 기업체를 타겟으로 하는 정보유출 보안사고[1]를 방지하기 위한 방법으로 물리적인 컴퓨터 자원을 안전한 사내 클라우드에 구축하여 컴퓨터 자원을 안전하게 운용하기 위한 프라이빗 클라우드 컴퓨팅[2] 기술이 관심을 받고 있다.

정부 또한 클라우드 컴퓨팅의 활성화 및 발전을 위해 많은 정책을 수립하고 있다[3]. 클라우드 관련 법률의 시행을 대표적인 예로 들 수 있다. 2015년 3월 3일 '클라우드 컴퓨팅 발전법'[4]이 국회에 통과되어 2015년 9월 28일부터 시행되고 있다. 공공기관이나 기업에서 클라우드 기술을 도입하고 이를 업무에 활용하도록 함으로써 클라우드 서비스를 이용하는 사용자가 증대되고 있다. 이러한 클라우드 서비스는 가상화 기술[5]을 이용하여 고성능의 물리적 서버에서 다수의 가상머신을 구동시켜 사용자에게 컴퓨터 자원을 제공한다.

클라우드 서비스를 이용하는 사용자가 증대됨에 따라, 사용자의 인스턴스에 대한 시스템 보안성 강화를 위해 클라우드 보안 및 관제 기술에 관한 다양한 연구가 진행되고 있다[6, 7, 8]. 특히 클라우드 플랫폼 내 구동중인 가상머신의 메모리 분석을 통해 악성행위 여부를 판단하거나 이에 대한 증거 수집을 위한 클라우드 디지털 포렌식 기술에 관한 연구가 그것이다.[9, 10]. 클라우드 가상머신 메모리 덤프를 통한 시스템 모니터링 기술은 가상머신 내 운영체제와 독립적으로 가상머신 구동을 관리하는 하이퍼바이저 계층에서 메모리 덤프를 수행하고 분석 할 수 있도록 한다. 게스트 운영체제에 별도의 에이전트 설치가 불필요하고, 메모리 분석을 통해 운영체제의 다양한 정보를 추출할 수 있다는 큰 이점이 있다. 이러한 이점에도 불구하고 메모리 분석기술이 클라우드 플랫폼과 같은 대규모 환경에서 모니터링 도구로써 활용될 수 없었던 이유는 가상머신의 메모리를 분석하기 위해 수행되어야 하는 메모리 덤프 연산이 과도한 연산 및 I/O 오버헤드를 발생하기 때문이다. 이러한 연산 및 I/O 성능적인 한계점으로 인해 다수의 가상머신이 하나의 서버 플랫폼에서 구동되는 클라우드 플랫폼과 같은 환경에서는 메모리 덤프 및 분석을 통한 심층 모니터링 기술이 활용되지 못하고 있다.

본 논문에서는 클라우드 보안성 강화를 위한 연산 효율적인 인스턴스 메모리 모니터링을 위한 기법을 제안한다. 본 논문에서 제안하는 기법은 게스트 운영체제의 커널 메모리 구조 분석을 통해 가상머신의 전체적인 메모리 덤프 없이, 시스템 모니터링에 필요한 커널 메모리 중 일부분만을 덤프 하여 분석을 하도록 하여, 메모리 덤프 시 발생되는 연산 및 I/O 오버헤드를 최소화 하였으며, 이를 통해 다수의 가상머신에 대한 효율적인 시스템 모니터링을 가능하게 한다.

본 논문의 구성은 다음과 같다. 2장에서는 메모리 분석 연구가 클라우드 모니터링에 활용하는데 따른 한계점이 대해 설명한다. 3장에서는 메모리 분석기술의 한계점을 해결한 메모리 부분 모니터링 덤프 기술과 이를 검증하기 위한 부분 메모리 모니터링 기반 인스턴스 악성행위 탐지 시스템에 대해 설명한다. 4장에서는 메모리 부분 모니터링 기술에 대한 연산 성능 비교 실험에 대한 결과를 설명하고, 5장에서는 결론 및 추후연구를 기술한다.

II. 관련연구

2.1 가상머신 메모리 덤프 기술

클라우드 컴퓨팅 환경에서 구동중인 인스턴스에 대한 메모리를 획득하기 위한 대표적인 도구로는 LibVMI[11], Libvirt[12]가 있다. LibVMI를 활용하여 구동중인 클라우드 인스턴스가 사용하고 있는 메모리 분석을 통해 하드웨어 이벤트나 vCPU 레지스터 접근 등에 관한 상세한 정보를 모니터 할 수 있다. 하지만 LibVMI와 같은 하이퍼바이저 계층에서 구동하는 메모리 모니터링 기법은 호스트 운영체제의 메모리로부터 가상머신이 할당받아 사용하고 있는 메모리로 영역까지의 메모리 주소 추적을 통해 분석이 이뤄지기 때문에 성능적/시간적 오버헤드가 발생한다[13].

Libvirt는 하이퍼바이저 계층에서 구동중인 가상머신을 관리하기 위한 연산을 구현한 대표적인 라이브러리이다. Libvirt에서 제공되는 API 중 인스턴스(가상머신)에 대한 메모리 덤프 연산을 위한 API, `virsh_dump()` 호출을 통해 인스턴스에 대한 메모리 덤프 파일을 얻을 수 있다. Libvirt에서 구현된 메모리 덤프 수행을 위한 API, `virsh_dump()`는 구동중인 가상머신을 일시정지 시킨 후, 호스트 운영체제의 페이지테이블 분석을 통해 가상머신이 사용 중인 메모리영역 정보를 얻은 후, 데이터를 추출 한다. 추출 된 데이터를 호스트 디스크에 출력을 요청하고 데이터 출력(쓰기연산)이 완료되면 가상머신을 재 구동

하는 절차를 갖는다.

2.2 가상머신 메모리 분석기술

메모리 분석기술은 메모리에 대한 라이브 분석을 통해 악성코드를 분석하거나, 사용자의 행위정보를 파악하기 위해 쓰이는 악성코드 분석 및 디지털 포렌식 기법 중 하나이다. 운영체제 메모리 공간에는 프로세스 정보, 네트워크 연결 정보, 레지스트리 정보, 파일 정보, 드라이버 정보 등 운영체제의 대부분의 정보가 포함되어 있다. 이와 같은 메모리 분석의 장점 때문에 난독화 된 악성코드를 분석하는데 Volatility[14], Rekall[15]과 같은 도구를 활용하기도 한다.

2.3 메모리 덤프/분석 기술의 한계점

메모리 분석의 다양한 장점에도 불구하고, 클라우드와 같은 환경에서 모니터링 도구로써 활용될 수 없었던 이유는 메모리 덤프/분석 시 발생하는 성능적인 오버헤드 때문이다.

2.3.1 인스턴스 메모리 덤프 덤프 연산 오버헤드

클라우드 컴퓨팅 환경에서 구동중인 인스턴스에 대한 메모리 덤프 정보를 추출하기 위해서는 인스턴스가 호스트 운영체제로부터 할당받아 사용하고 있는 메모리 영역을 읽어 이를 파일의 형태로 저장하는 연산이 필요하다. 이를 수행하기 위해서는 파일 연산 오버헤드가 발생한다. 본 논문에서는 클라우드 인스턴스 메모리 덤프연산에 따른 오버헤드를 측정하기 위한 실험을 수행하였다. 실험 결과에 따르면 Libvirt를 이용하여 1GB의 메모리를 갖는 인스턴스의 메모리 덤프를 수행하는데 걸리는 소요시간은 4,945ms로 측정되었다. 이는 클라우드 인스턴스에 모니터링 주기는 최소 4,945ms라는 것을 의미한다. 시스템 모니터링은 실시간성이 중요하기 때문에, 클라우드 인스턴스에 대한 메모리 덤프에 수반되는 시간적 오버헤드는 메모리 덤프를 활용한 모니터링 기술의 실현에 있어 한계점으로 작용된다.

2.3.2 클라우드 인스턴스 일시 멈춤 현상에 따른 한계점

클라우드 인스턴스에 대한 메모리 모니터링을 위해 가상머신에 대한 메모리 덤프 수행이 진행되는 동안 해당 인스턴스에 대한 일시 멈춤 현상이 발생하게 되어 사용성을 저해하는 문제가 발생한다. 메모리 덤프에 따른 가상머신 구동성능 실험 결과에 따르면 Libvirt를 이용하여 1GB의 메모리를 가진 가상머신에 대한 메모리 덤프를 수행하는 동안 가상머신 내 운영체제를 포함한 모든 SW의 동작이 4,945ms 동안 일시정지 되는 현상이 발생했다. 이는 특정 인스턴스에 대한 메모리 덤프연산을 수행하는 동안, 인스턴스 내 구동중인 운영체제의 구동이 멈추는 데에 기인한 것이다.

2.3.3 복수 인스턴스에 대한 인스턴스 메모리 모니터링 수행 한계점

2.3.1장에서 수행한 클라우드 인스턴스 메모리 덤프 연산 오버헤드를 측정한 실험결과에 따르면, 클라우드 플랫폼에서 1GB의 메모리를 갖는 가상머신의 메모리를 분석하기 위해서, 1GB의 메모리 정보를 파일로 추출하는데 약 4,945ms가 소요된다. 인스턴스의 메모리 덤프를 수행하는 동안에는 호스트 운영체제로부터 할당된 메모리 영역을 파일로 쓰는 연산이 수행되어, 해당 인스턴스가 일시적으로 멈추는 현상이 발생하게 된다. 이로 인해 최소 4,945ms 이상 동안 가상머신의 동작이 일시적으로 멈춰지게 된다. 그렇기 때문에 다수의 가상머신이 구동되는 클라우드 플랫폼에서는 메모리 덤프를 통한 인스턴스 모니터링 연산을 수행하게 될 경우, 사용자가 운용 중인 인스턴스에 주기적인 일시 멈춤 현상이 발생하게 되며, 동시에 복수개의 가상머신에 대한 인스턴스 모니터링 연산을 수행하게 될 경우 (예를 들어, 10개의 인스턴스가 하나의 물리적 서버 플랫폼에서 구동되는 상황의 경우) 각 인스턴스의 일시 멈춤 현상은 최소 49,450ms 이상이 발생하게 된다. 이처럼 기존 Libvirt에서 제공하는 인스턴스에 대한 메모리 덤프 연산은 확장성(Scalability) 및 병렬성(Parallelism)을 제공하지 않아, 복수개의 인스턴스에 대한 메모리 덤프 연산 시 발생하는 오버헤드는 하나의 서버 플랫폼에 구동중인 인스턴스의 개수에 비례하여 증가하게 된다.

III. 부분 메모리 덤프를 통한 연산효율적인 인스턴스 모니터링 기술

본 장에서는 기존 메모리 분석을 통한 클라우드 시스템 모니터링 연산에서 발생하는 오버헤드를 획기적으로 줄일 수 있는 부분 메모리 덤프를 통한 연산 효율적인 인스턴스 모니터링 기술을 제안한다. 메모리 덤프 시 발생하는 오버헤드를 최소화하기 위해 악성코드 분석 및 탐지에 필요한 인스턴스 메모리의 특정 부분을 모니터링 하는 기술을 제안하고, 부분 메모리 모니터링 기반 악성코드 탐지 시스템을 통해 제안 기술의 실효성을 검증한다. 구현된 부분 메모리 덤프 기술의 활용성을 검증하기 위해, 메모리 분석을 통해 악성코드를 탐지 할 수 있는 시스템을 설계하였다.

3.1 메모리 덤프 연산 오버헤드 발생 요인 분석

본 논문에서는 가상머신의 메모리 정보를 추출하기 위해 가상머신 관리도구의 대표적인 Libvirt를 사용하였다. 메모리 덤프 연산 시 발생하는 오버헤드의 원인을 파악하기 위해 내부 연산과정을 분석하였다. Perf[16]의 report기능을 이용하여 Libvirt의 내부 메모리를 덤프하기 위한 과정과 QEMU에서 메모리를 덤프하기위해 수행되는 연산 과정을 분석해본결과 Table 1.과 같다. 메모리 덤프 요청 시 가상화 지원을 위한 *qemu-system-x86_64*를 제외하고 *copy_page_rep()* 함수가 11.15%로 덤프 연산 시 가장 많은 성능 지연을 유발했다. *copy_page_rep()* 함수는 가상머신의 메모리 정보를 디스크로 출력하는데 호출되는 함수이다. 따라서 디스크로 출력되는 메모리의 사이즈를 최소화 할 수 있다면, 메모리 덤프 연산 시 발생하는 오버헤드를 최소화 할 수 있다.

Table 1. Operation profile for cloud instance memory dump

Symbol (Object)	Overhead
[q] qemu-system-x86_64	34.97 %
[k] copy_page_rep	11.15 %
[k] intel_idle	4.21 %
[q] 0x2e87c	2.77 %
[q] 0x1332de	2.25 %
[q] 0x58bf	1.37 %
[q] vfprintf	1.20 %

k: kernel function call
q : qemu and I/O function call

3.2 가상머신 부분 메모리 덤프 기술

메모리 덤프/분석 시 발생하는 연산의 오버헤드를 최소화시키기 위해서는 메모리 덤프 영역을 최소화시켜야 한다. 메모리 덤프 영역을 최소화시키기 위해 악성코드 분석 및 탐지에 필요한 인스턴스 메모리의 특정 부분에 대한 모니터링 덤프를 수행하고, 분석하는 방법을 고안하였다.

Fig. 1.과 같이 호스트 머신에서 가상머신이 사용하는 메모리 영역 중 커널 메모리 영역에 대한 부분 메모리 덤프를 수행하기 위해, 가상머신의 페이지 테이블 분석을 수행하였다. 추출된 커널 메모리 영역의 주소를 기반으로 QEMU[17]의 pmemsave() 함수를 이용하여 각 가상머신의 커널 메모리 영역을 추출할 수 있도록 하였다.

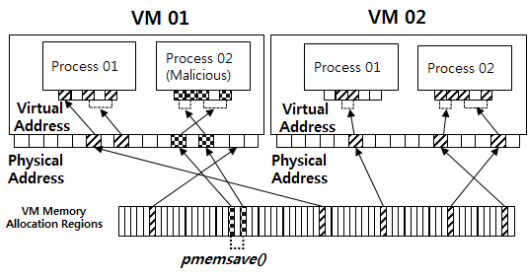


Fig. 1. Partial memory dump for cloud instance using 'pmemsave'

3.3 커널 메모리 맵 라이브러리 설계 및 구현

모니터링 대상 가상머신에 대한 커널 메모리정보를 취득하기 위해서는, 각 가상머신의 운영체제 버전별로 커널 메모리로 활용되는 메모리 영역의 시작 주소 및 오프셋 정보가 필요하다. 이와 같은 메모리 영역의 시작 주소는 가상머신에 할당된 메모리 용량 및 설치된 운영체제에 따라 커널 메모리 영역의 시작주소가 가변적이다. 이와 같은 문제를 해결하기 위해 가상머신의 메모리 할당량 및 설치된 운영체제의 버전에 따른 커널 메모리 영역에 대한 시작주소 정보를 효율적으로 관리하기 위해 운영체제 별, 메모리 사이즈 별 커널 메모리 맵 라이브러리를 구축하였다.

커널 메모리 맵 라이브러리는 Fig. 2.와 같이 가상머신 별 커널 메모리 영역에서 추출 가능한 시스템

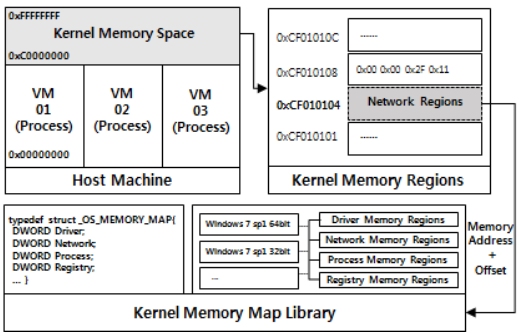


Fig. 2. Design of library for kernel memory map

정보에 대한 메모리 위치를 반환해주는 메모리 맵 테이블이다. 커널 메모리 맵 라이브러리를 통해 각 가상머신의 네트워크, 프로세스, 레지스트리, 드라이버 등 각 영역의 시스템 상태를 확인할 수 있는 커널 메모리 영역의 시작 주소와 오프셋 정보를 획득 할 수 있다.

커널메모리 맵 라이브러리를 통해 모니터링을 수행할 가상머신을 지정하게 되면, 커널 메모리 맵 테이블을 통해 커널 가상머신에 탑재된 운영체제의 저장된 커널 메모리 영역의 범위를 추출하고, 추출 된 메모리 영역에 대한 부분 메모리 덤프 및 분석을 수행할 수 있게 된다. 커널 메모리 라이브러리 맵 테이블의 경우 운영체제 별 커널 영역 메모리에 대한 레이아웃 분석을 통해 운영체제 별 메모리 할당정보를 데이터베이스화 하였다. 이를 통해 사전에 파악된 운영체제의 시스템 정보가 저장되는 특정 커널 메모리 영역을 파악하고 해당 영역에 대한 부분 메모리 덤프 연산을 통해 시스템 행위정보를 분석하게 된다. 그러므로 시스템 구동 중에는 커널 메모리 라이브러리 맵 생성에 따른 오버헤드는 발생하지 않는다.

3.4 메모리 분석을 통한 인스턴스 악성행위 탐지 기술

본 논문에서 제안한 부분 메모리 모니터링 기술이 클라우드 플랫폼에서 모니터링 도구로 활용이 가능함을 검증하기 위해 악성코드가 수행하는 행위를 탐지할 수 있는 시스템을 설계하였다.

부분 메모리 모니터링 기반의 악성코드 탐지 시스템은 악성코드가 동작 시 수행되는 연산정보 및 사용되는 드라이버, 네트워크, 레지스트리 등의 시그니쳐 정보 맵핑을 통해 악성코드 감염 여부를 판단한다.

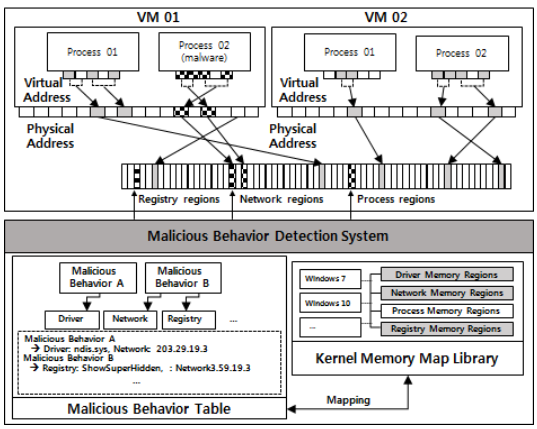


Fig. 3. Overall architecture for malicious behavior detection in cloud platform

악성코드 탐지 실험을 위해 악성코드를 분석하여 악성코드 동작 시 사용되는 드라이버, 네트워크, 레지스트리 등의 시그니쳐 정보를 수집하였다. Fig. 3.과 같이 수집된 악성코드 시그니쳐 정보와 커널 메모리 맵 라이브러리를 통해 모니터링 되어야 하는 최소한의 커널 메모리 영역을 추려낸다. 악성코드 탐지를 위해 지정된 최소의 커널 메모리 영역에 대해서 메모리 덤프와 분석을 반복적으로 수행하고, 커널 메모리 영역에서 악성코드의 시그니쳐 정보가 탐지되면 악성코드에 감염되었다고 판단하고 악성코드 탐지 정보를 알리도록 구현하였다.

3.5 메모리 모니터링 기술의 활용 (오픈스택)

본 논문에서 제안한 부분 메모리 모니터링 기술을 클라우드 플랫폼에 적용하기 위해 설계 및 구현한 메모리 모니터링 시스템의 구조는 Fig. 4.와 같다. 구현 된 메모리 모니터링 시스템은 클라우드 오픈소스 플랫폼인 오픈스택에서 보안 도구로써 활용 할 수 있도록 설계되었다.

구현 된 메모리 부분 모니터링 시스템의 내부 구조는 크게 2가지(Memory Monitor, Openstack Manager) 기능으로 구성되어 있다. 각 기능에 대한 설명은 다음과 같다.

Memory Monitor: Memory Monitor는 클라우드 플랫폼 내 구동중인 가상머신의 커널 메모리 영역을 모니터링 하여 악성코드 감염에 여부에 대해 판단한다. Memory Monitor의 내부구조는 6개의 모듈로 이루어져있다. 모듈에 대한 설명은 다음과 같

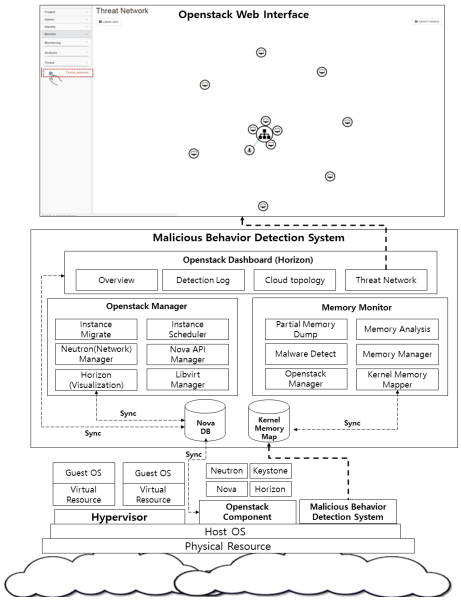


Fig. 4. Partial memory monitor system in Openstack platform

다. Partial Memory Dump 모듈은 메모리의 특정 영역을 파일로 추출한다. Memory Analysis 모듈은 덤프 된 메모리 정보를 분석하고, Memory Manager 모듈은 덤프 된 메모리 파일과 분석된 정보를 인스턴스 별, 시간 별 관리한다. 추가적으로 Malware Detection 모듈은 Kernel Memory Mapper 모듈에서 관리하는 운영체제 커널 메모리 맵 라이브러리 정보를 활용하여 커널 메모리 영역에 악성코드가 시그니쳐 정보가 로드되는지 탐지한다. 마지막으로 Openstack Manager 모듈은 메모리 모니터링 모듈과 Openstack 컴포넌트가 효율적으로 연동 될 수 있도록 인터페이스 역할을 수행한다.

Openstack Manager: Openstack Manager는 Memory Monitor의 기능을 오픈스택 내부 컴포넌트와 연동시킬 수 있도록 인스턴스에 제어에 관한 다양한 인터페이스를 제공한다. Instance Scheduler 모듈은 현재 오픈스택 내 구동중인 인스턴스의 상세정보(인스턴스 id, 네트워크 id, 메모리 사이즈 등)를 제공해준다. Instance Migrate 모듈은 구동중인 인스턴스가 악성코드에 감염 되었을 때 다른 컴퓨트 노드로 라이브 마이그레이션을 수행할 수 있도록 하는 기능을 제공한다. Neutron Manager 모듈은 현재 구동중인 인스턴스가 사용하는 가상 네트워크 상세정보를 제공한다. Nova API Manager 모듈과 Libvirt Manager 모듈은 인스턴스를 웹 대시보드에서 관리 할 수 있도록 인스턴스에 대한 다양한 관리 기능 (소프트 재부팅, 하드 재부팅, 일시정지 등)을 제공한다. 마지막으로 Horizon 모듈은 메모리 모니터링 수행 내역에 대한 로그 정보, 인스턴스 악성코드 감염 정보, 구동중인 인스턴스 토폴리지 등을 시각화하여 사용자에게 제공 해준다.

IV. 성능평가

본 장에서는 구현된 메모리 모니터링 시스템에 대한 활용성을 검증하기 위해 모니터링 모듈이 적재된 클라우드 플랫폼에 악성코드 탐지 성능을 평가한다. 클라우드 플랫폼에 대한 활용성을 검증하기 위해 IaaS 오픈 클라우드 서비스인 오픈스택(Ver. of Liberty)[18]을 구축하였다.

4.1 실험 환경

본 연구의 실험 환경을 구축하기 위하여 5대의 서버 노드로 구성된 클라우드 테스트베드를 구성하였으며, 각 서버 노드에는 Xeon E5-2609(2.5GHz), 32GB의 메모리, SSD 128GB를 장착하였다. 가상머신이 구동되는 컴퓨트 노드의 호스트 운영체제는 Ubuntu 16.04-64bit를 설치하였으며, 하이퍼바이저는 KVM(QEMU)을 사용하였다. 추가적으로 메모리 모니터링 대상이 되는 가상머신은 메모리를 1GB를 갖는 Windows 7 SP 1-64bit를 설치하였다.

4.2 부분 메모리 덤프를 통한 악성코트 탐지 성능

부분 메모리 모니터링 기술과 전체 메모리 모니터링 기술의 성능을 비교하기 위해 클라우드 플랫폼에 다수의 가상머신을 구동 시키고 악성코드에 감염시켰을 때의 탐지까지의 소요시간을 측정하였다.

본 실험에서는 커널 메모리영역 중 네트워크 커널 메모리 영역에 대한 부분 메모리 덤프를 추출하고 이를 통해 악성행위를 탐지한다. 측정 대상의 가상머신에는 Windows 7 SP1-64bit가 탑재되었으며, 1GB의 메모리를 할당하였다. 1GB의 메모리가 할당된 가상머신의 경우 네트워크 커널 메모리 영역에 해당되는 부분 메모리 덤프 용량은 약 10MB이었으며, 가상머신의 전체 메모리를 모니터링 방법과 가상

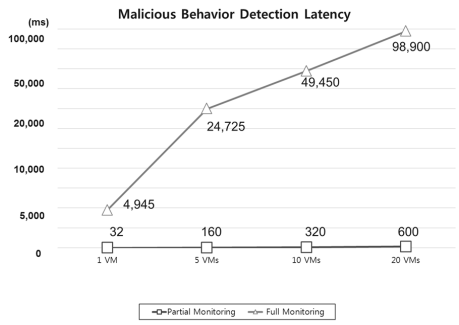


Fig. 5. Performance comparison of malicious behavior detection latency

머신의 부분 메모리(네트워크 커널 메모리 영역) 모니터링을 수행한 성능 비교 결과는 Fig. 5.와 같다.

1대의 가상머신에 대해 메모리를 모니터링 수행한 결과 전체 메모리 모니터링의 경우 악성코드를 탐지하는데 약 4,945ms의 시간이 소요되었고, 부분 메모리 모니터링의 경우 약 32ms의 시간이 소요되었다. 이와 같은 결과는 부분 메모리 모니터링 기술은 전체 메모리 덤프 기술과 비교하여 오버헤드가 1/154로 감소되었음을 보여준다. 추가적으로 10대의 가상머신에 대한 메모리를 모니터링을 할 경우 전체 메모리를 모니터링하게 되면 약 24,725ms 시간 동안 구동 중인 모든 가상머신들이 일시정지 되었지만, 부분 메모리의 경우 160ms의 시간 동안 가상머신 일시정지 현상이 발생되었다. 즉 부분 메모리 덤프 기반의 모니터링 기술은 다수의 가상머신이 구동되는 환경에서도 다른 가상머신에게 끼치는 영향이 기존 전체 메모리 덤프 기술과 비교하여 1/154로 줄어들었다.

실험 결과와 같이 부분 메모리 모니터링 기술을 통해 전체 메모리 모니터링 시 발생하는 덤프 성능 오버헤드와 일시정지현상 문제를 해결하였다. 또한 짧은 시간 내에 수많은 가상머신에 대해서 메모리 추출이 가능하졌기 때문에, 병렬 메모리 모니터링의 한계점도 해결되었다. 식별이 완료된 악성코드에 대한 탐지율 비교를 전제하였을 경우, 식별에 필요한 커널 메모리 영역을 사전에 인지할 수 있기 때문에, 전체 메모리 덤프 방법과 부분 메모리 덤프의 악성코드 탐지 능력 성능은 동일하게 된다. 이는 전체 메모리 덤프의 경우 커널 메모리 전체 영역에 대한 악성코드 시그니쳐 정보를 탐색하는 것이고, 부분 메모리 덤프의 경우 악성코드 식별에 필요한 커널 영역의 메모리 정보가 저장되는 최소의 영역에 대한 메모리 덤프 연산을 수행하기 때문이다.

V. 결 론

본 논문에서는 메모리 덤프 기술이 모니터링 기술로써 활용될 수 없었던 연산 오버헤드 문제를 해결하기 위해 부분 메모리 덤프 기반의 모니터링 기술을 제안하였다. 또한 이를 검증하기 위해 악성코드 탐지 시스템을 구현하였다. 구현한 악성코드 탐지 시스템을 통해 기존 메모리 덤프 기술과 성능을 비교해 본 결과 연산 오버헤드가 1/154로 줄어들었다. 이와 같은 결과를 통해 본 논문에서 제안한 부분 메모리 덤프 기반 모니터링 기술이 다수의 가상머신이 구동되는 클라우드 플랫폼 같은 대규모 환경에서 가상머신의 보안성 강화를 위한 모니터링 도구로써 활용될 수 있음을 입증하였다. 본 연구에서는 제한된 운영체제 집합에 대한 커널 메모리 라이브러리 맵 생성을 통한 부분 메모리 덤프 연산에 대한 연산효율화 방안을 연구를 수행하였지만, 이를 확장하여 동적 커널 메모리 라이브러리 삽입 및 모듈화를 통한 부분 메모리 덤프 기술에 대한 확장연구를 수행할 계획이다.

References

[1] Jae-yoon Sim and Kyung-ho Lee, "A Study on Information Access Control Policy Based on Risk Level of Security Incidents about IT Human Resources in Financial Institutions," Journal of The Korea Institute of Information Security & Cryptology, 25(2), pp. 343-361, Apr, 2015

[2] Armbrust, Michael, et al. "A view of cloud computing," Communications of the ACM vol. 53, no. 4, pp. 50-58, Apr, 2010

[3] Sang-Baek Chris Kang, "Cloud Computing Strategy Recommendations for Korean Public Organizations - Based on U.S. Federal Institutions' Cloud Computing Adoption Status and SDLC Initiative," The Jounal of Society for e-Business Studies, vol. 20, no. 4, pp. 103-236, Nov, 2015

[4] Cloud Computing Development Act, http://www.law.go.kr/lsInfoP.do?lsiSeq=169562&chrClsCd=010204#0000

[5] Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," Computer vol. 38, no. 5, pp. 48-56, Feb, 2005

[6] Taehyoung Kim, Inhyuk Kim, Junghan Kim, Changwoo Min, Jee-hong Kim and Young Ik Eom, "Security-Enhanced Local Process Execution Scheme in Cloud Computing Environments," Journal of The Korea Institute of Information Security & Cryptology, 20(5), pp. 69-79, Oct, 2010

[7] Safaa Salam Hatem, Dr. Maged H. wafy and Dr. Mahmoud M. El-Khouly, "Malware detection in Cloud computing," International Journal of Advanced Computer Science and Applications, vol. 5, no.4, pp. 187-192, Apr, 2014

[8] Tamas K. Lengyel, Steve Maresca, Bryan D. Payne, George D. Webster, Sebastian Vogl and Aggelos Kiayias, "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system," Proceedings of the 30th Annual Computer Security Applications Conference, pp. 386-395, Dec, 2014

[9] Artem Dinaburg, Paul Royal, Monirul Sharif and Wenke Lee, "Ether: malware analysis via hardware virtualization extensions," Proceedings of the 15th ACM conference on Computer and communications security, pp. 51-62, Oct , 2008

[10] Rayan Mosli, Rui Li, Bo Yuan and Yin Pan, "Automated malware detection using artifacts in forensic memory images," Technologies for Homeland Security (HST), 2016 IEEE Symposium on, pp. 1-6, May, 2016

[11] Haiquan Xiong, Zhiyong Liu, Weizhi Xu and Shuai Jiao, "Libvmi: a library for bridging the semantic gap between guest OS and VMM, " : Proceedings of the 12th International Conference on Computer and Information Technology, pp.549-556, Oct, 2012

[12] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehörster and André Brinkmann, "Non-intrusive virtualization management using libvirt," Proceedings of the Conference on Design, Automation and Test in Europe, pp. 574-579, Mar, 2010

[13] Sang-hoon Choi", Accelerated memory dump and memory recording schemes for analyzing cloud computing platform in a microscopic level," Daejeon university of graduation thesis, pp. 1-26, Aug, 2016

[14] Volatility, http://www.volatilityfoundation.org/about

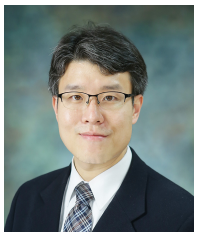
[15] Rekall, http://www.rekall-forensic.com/about.html

[16] De Melo, Arnaldo Carvalho. "The new linux perf' tools," Slides from Linux Kongress. Vol. 18, Sep, 2010

[17] Bellard, Fabrice. "QEMU, a fast and portable dynamic translator," USENIX Annual Technical Conference, pp. 41-46, Apr, 2005

[18] Sefraoui, Omar, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, vol. 55, no. 3, pp. 38-42, Oct, 2012

〈저자소개〉



박 기 웅 (Ki-Woong Park) 종신회원
연세대학교 Computer Science 학사
KAIST Electrical Engineering 석사 (시스템보안 전공)
KAIST Electrical Engineering 박사 (시스템보안 전공)
~2009년 10월: Microsoft Research, Graduate Research Fellow
~2012년 8월: 국가보안기술연구소
~2016년 8월: 대전대학교 정보보안학과 조교수
2016년 9월~현재: 세종대학교 정보보안학과 조교수
〈관심분야〉 클라우드 시스템 보안, 초고속 보안 시스템, 시스템 심층관제

최 상 훈 (Sang-Hoon Choi) 학생회원
대전대학교 Computer and Information Security 학사
대전대학교 Computer and Information Security 석사 (시스템보안 전공)
2017년 3월~현재: 세종대학교 Computer and Information Security 박사과정
〈관심분야〉 클라우드 시스템 보안, 메모리 포렌식, 시스템 모니터링