https://doi.org/10.33778/kcsa.2023.23.4.023

컨테이너 기반 가상화에서 격리성 보장을 위한 취약성 고찰★

염 다 연*, 신 동 천**

요 약

클라우드 컴퓨팅 환경에서 컨테이너 기반 가상화는 게스트 운영체제 대신에 호스트 운영체제를 공유함으로써 가벼운 사용감으로 가상머신 기반 가상화 기술의 대안으로 많은 관심을 받고 있다. 그러나 호스트 운영체제를 공유함으로써 발생하는 문제점이 컨테이너 기반 가상화의 취약성을 높일 수 있다. 특히 컨테이너들이 자원들을 과도하게 사용함으로 인해 컨테이너들의 격리성을 침해할 수 있는 noisy neighbor problem은 사용자들의 가용성을 위협하게 되므로 보안 문제로 인식할 필요가 있다. 본 논문에서는 컨테이너 기반 가상화 환경에서 noisy neighbor problem이 격리성 보장을 위협할 수 있는 취약성을 고찰한다. 이를 위해 컨테이너 기반의 가상화 구조를 분석하여 기능별 계층에 대한 격리성 보장에 위협이 될 수 있는 취약점을 도출하고 해결 방향과 한계점을 제시한다.

A Study on Vulnerability for Isolation Guarantee in Container-based Virtualization

Dayun Yum*, Dongcheon Shin**

ABSTRACT

Container-based virtualization has attracted many attentions as an alternative to virtual machine technology because it can be used more lightly by sharing the host operating system instead of individual guest operating systems. However, this advantage may owe some vulnerabilities. In particular, excessive resource use of some containers can affect other containers, which is known as the noisy neighbor problem, so that the important property of isolation may not be guaranteed. The noisy neighbor problem can threat the availability of containers, so we need to consider the noisy neighbor problem as a security problem. In this paper, we investigate vulnerabilities on guarantee of isolation incurred by the noisy neighbor problem in container-based virtualization. For this we first analyze the structure of container-based virtualization environments. Then we present vulnerabilities in 3 functional layers and general directions for solutions with limitations.

Key words : Cloud computing, Virtualization, Container, Noisy neighbor problem, Isolation property

접수일(2023년 08월 20일), 수정일(2023년 09월 11일), 게재확정일(2023년 10월 19일)

★ 이 논문은 2021년도 중앙대학교 연구장학기금 지원에 의한 것임.

* 중앙대학교 융합보안학과

** 중앙대학교 산업보안학과(교신저자)

1. 서 론

클라우드 컴퓨팅은 유휴 리소스를 공유함으로써 효율성을 높이고 비용을 절감하는 데 목적을 두었기 때문에 많은 사람이 인터넷 자원을 서비스로 사용할 수 있는 대규모 저장소가 될 수 있다. 클라우드 컴퓨팅은 기업과 개인 모두에게 기존 IT 환경을 변화시켜 다양한 이점을 제공하고 있으며, 이러한 이유로 많은 기업이 클라우드 기반의 인프라 및 서비스를 사용하여 비즈니스 운영을 발전시키고 있다.

가상화는 클라우드 컴퓨팅의 주요한 특징이다. 클라우드 컴퓨팅에서 가상화 기술을 활용하여 리소스를 효율적으로 활용할 수 있으며, 가상화를 통해 하나의 물리 서버나 리소스를 여러 개의 가상 환경으로 분할하고 동시에 다양한 어플리케이션을 실행할 수 있다[1]. 이를 통해 하드웨어의 사용률을 최적화하고, 서버의 가용성과 탄력성을 향상할 수 있다.

컨테이너 기반의 가상화는 최근 클라우드 컴퓨팅에서 가장 많이 사용되는 가상화 방식 중 하나로, 컨테이너는 어플리케이션과 그에 필요한 종속성을 패키징하여 실행 환경을 격리된 상태로 구성하는 것을 가능하게 한다[2]. 가상머신에 비해 컨테이너는 가볍고 빠르게 배포되는 이식성을 제공하기 때문에 개발과 운영의 효율성을 크게 향상 시킨다는 장점이 있어 많이 사용되고 있다[3].

가상머신에서 동작하는 게스트 운영체제(Guest OS)는 호스트 운영체제(Host OS)와 같을 필요가 없으므로, 하나의 가상머신을 기준으로 제공할 수 있는 서비스의 다양성 측면에서는 기존 가상머신이 더 유리하다고 볼 수 있다. 또한, 가상머신 사이에 격리성(isolation)이 보장되어 하나의 가상머신 장애가 다른 가상머신이나 시스템 전체에 영향을 미치지 않는다[4]. 그러나 컨테이너는 호스트 운영체제가 통제하는 영역을 사용하지만, 많은 컨테이너가 동일한 커널을 공유한다. 따라서 컨테이너 기반 가상화는 가상머신처럼 철저하게 분리되어 있지 않기 때문에 보안이나 안정성 측면에서 문제가 발생할 수 있다[5].

컨테이너 기반의 가상화 환경에서 발생하는 noisy neighbor problem은 다수의 컨테이너가 하나의 호스트에서 동작할 때 발생하는 현상을 말한다. 이 문제는 컨테이너 간의 리소스 공유로 인해 발생하며, 한 컨테이너가 과도한 리소스를 사용하거나 예기치 않은 동작을 수행할 때 다른 컨테이너의 성능에 영향을 줄 수 있다. 따라서 noisy neighbor problem은 컨테이너 기반 가상화 기술을 사용하면서 성능이 저하되는 문제로 판단한다[6], [7]. 리소스 사용의 문제로 인해 클라우드 성능에 문제가 발생하는 것은 IT 보안을 평가하기 위해 사용하고 있는 보안의 3요소 중 가용성을 저해한다고 할 수 있다. 따라서 컨테이너의 noisy neighbor problem은 컨테이너의 성능을 저하시키는데 그치지 않고 보안 문제로 이어지는 중요한 문제라고 할 수 있기 때문에 컨테이너 기반 가상화의 취약성에 대한 분석은 중요하다.

본 논문에서는 컨테이너 기반 가상화 기능들로 인하여 발생할 수 있는 noisy neighbor problem이 격리성 보장에 미칠 수 있는 취약성을 고찰하고 일반적인 해결 방향 및 한계점을 제시한다. 이를 위해 많은 연구에서 사용하고 있는 LXC(Linux Container)를 기반으로 컨테이너 기반 가상화 구조를 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 컨테이너 기반 가상화에서 발생할 수 있는 문제점인 noisy neighbor problem에 관해 기술한다. 3장에서는 컨테이너 기반의 가상화 구조에 대해 분석하며 4장에서는 noisy neighbor problem 관점에서 격리성을 위한 컨테이너 기반 가상화의 계층별 취약점에 관해 기술한다. 5장에서는 결론을 맺는다.

2. Noisy neighbor problem

2.1 가상화

가상화는 컴퓨팅 리소스를 추상화하고, 하드웨어 계층에서 소프트웨어 계층을 분리하며, 사용 중인 하드웨어에서 실행 중인 응용 프로그램을 격리한다[8]. 이러한 가상화는 클라우드 컴퓨팅 서비스를 효율적으로 지원하고 관리할 방안이자 클라우드 컴퓨팅 서비스의 기반이 되는 기술이다. 가상화는 사용자와 물리적 리소스 간의 유연성을 높이기 때문에 클라우드 환경에서 중요하다고 할 수 있다[9]. 가상화는 자동화 된 IT 관리와 문제가 생겼을 때 빠른 시간 내에 복구가 가능하다는 장점이 있고, 리소스를 공유할 수 있다는 장

점이 있다. 모든 리소스가 실행 프로그램 전용으로, 가상화 되지 않은 환경과 달리 가상화된 환경에서는 VM, 즉 가상머신이 기본 호스트의 메모리, 디스크 및 네트워크 장치와 같은 물리적 리소스를 공유한다. 가상화는 하이퍼바이저와 컨테이너 기반 가상화로 구분할 수 있다[10].

하이퍼바이저는 하드웨어 가상화를 제공하는 소프트웨어로 모든 하드웨어 리소스의 가상화를 지원하기 때문에 단일의 물리적 호스트에서 여러 컴퓨팅을 실행할 수 있다. 하이퍼바이저의 기본 기능은 게스트 운영체제 분리를 적용하고 가상머신 간의 통제된 리소스 공유를 수행하는 것으로 하드웨어 수준에서 작동하기 때문에 호스트 시스템과 독립적이고 격리된 독립 실행형 가상 머신을 지원하며 하이퍼바이저가 기본 호스트 시스템에서 가상머신을 격리하는 역할을 한다. 그러나 하이퍼바이저 때문에 게스트 운영체제가 가상머신에 설치될 수 있고, 게스트 운영체제가 존재함으로써 가상머신의 독립 실행이 가능해지지만, 이미지가 더 커진다는 단점이 존재한다. 또한 하이퍼바이저는 하드웨어를 추상화하기 때문에 하드웨어 및 가상 장치 드라이버 가상화 측면에서 오버헤드가 발생할 수 있다. 이러한 하이퍼바이저의 단점은 컨테이너 기반 가상화를 발전시켰다[11].

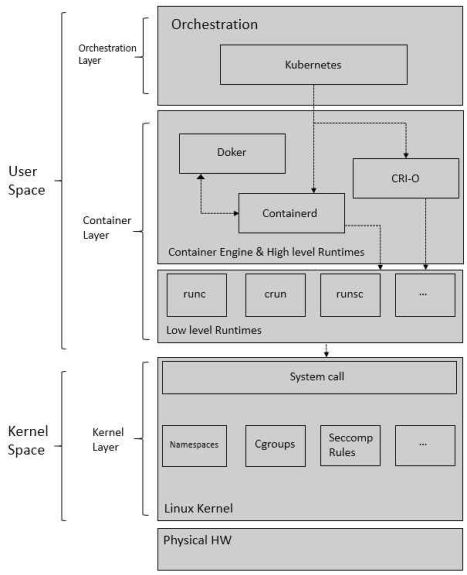
컨테이너는 소프트웨어를 실행하는 데 사용되는 가상화 기술이다[12]. 어플리케이션과 그 어플리케이션을 실행하는 데 필요한 모든 라이브러리, 실행 파일, 설정을 포함하고 있는 독립적인 실행 환경을 제공한다. 이러한 컨테이너는 다양한 환경에서 일관된 방식으로 실행될 수 있어 개발, 배포 및 관리 과정을 단순화하고 효율성을 높여준다. 또한 컨테이너는 어플리케이션과 그 종속성을 모두 포함하고 있어 다른 환경에서도 일관된 방식으로 실행될 수 있다. 개발 환경과 운영 환경 사이의 일관성을 유지하고, 어플리케이션을 쉽게 배포하고 이식할 수 있다. 컨테이너는 호스트 시스템으로부터 격리되어 독립적으로 실행된다. 각 컨테이너는 자체 파일 시스템, 프로세스 공간, 네트워크 인터페이스를 가지며, 다른 컨테이너나 호스트 시스템과 분리하는 역할을 한다. 이를 통해 어플리케이션 간의 충돌을 방지하고 보안을 강화할 수 있다. 컨테이너는 쉽게 복제하고 확장할 수 있다. 어플리케이션의 부하에 따라 필요한 만큼의 컨테이너를 생성하여 수평적으로 확장할 수 있으며, 로드 밸런싱과 같은 기술을 활용하여 자동으로 분산 처리할 수 있다. 이를 통해 어플리케이션의 의존성 관리와 버전 관리가 쉬워지며, 인프라 구성을 코드로 관리하는 것이 가능하다. 이러한 컨테이너는 현대의 소프트웨어 개발과 운영에 많은 장점을 제공한다.

2.2 Noisy neighbor problem

동일한 물리적 서버를 기반으로 하는 컨테이너는 CPU, 메모리 또는 캐시 같은 리소스를 공유한다. 리소스 공유로 인해 컨테이너는 함께 배치된 다른 컨테이너에 영향을 미치거나 영향을 받을 수 있다. 이를 noisy neighbor problem이라고 한다[6]. Noisy neighbor problem 현상은 특정 컨테이너가 과도한 리소스를 사용하거나 예기치 않은 동작을 수행할 때 다른 컨테이너의 성능에 영향을 미칠 수 있다[13]. 컨테이너 기반의 가상화 환경에서 발생하는 noisy neighbor problem은 다수의 컨테이너가 하나의 호스트에서 동작할 때 발생하는 현상에 기인한다. 하이퍼바이저 기반 가상화 환경에서는 각 가상머신마다 게스트 운영체제가 있기 때문에 하나의 가상머신이 컴퓨팅 환경을 이루고 강력한 격리를 이룬다. 그러나 컨테이너 환경에서는 공유된 운영체제로 수평적 확장된 컨테이너를 관리하기 때문에 가상머신을 활용하는 하이퍼바이저보다 격리가 약하다고 할 수 있다.

컨테이너 기반 가상화의 보안 문제점은 커널을 공유함으로써 발생하는 현상에서 시작한다. 대부분의 연구는 배포된 이미지를 컨테이너에 이식하며 이로부터 들어오는 공격 및 컨테이너 사용자로부터 발생하는 직간접적인 공격에 초점이 맞추어져 있다[14], [15]. 반면 noisy neighbor problem은 컨테이너 기반 가상화의 커널 공유에 초점을 맞추어 리소스 공유에서 비롯한 다른 컨테이너에 대한 영향을 미치는 문제다. 리소스를 공유하면서 특정 컨테이너가 리소스를 독점하거나 예기치 않은 동작을 수행할 때 다른 컨테이너의 성능에 영향을 미칠 수 있는 문제이다. 따라서 기존에 제기된 컨테이너 기반 가상화의 보안 문제에 새롭게 제기할 수 있는 문제라고 할 수 있다[16].

3. 컨테이너 기반의 가상화 구조 분석



(그림 1) 컨테이너 기반 가상화 구조[17]

대부분의 컨테이너 기반 가상화 연구는 리눅스 커널을 사용하는 LXC(Linux Container)를 기반으로 한다. LXC는 단일 리눅스 시스템에 동작하고 있는 프로세스를 격리시켜 프로세스마다 독자적인 리눅스 시스템 환경을 구축할 수 있는 환경을 말한다[16], [17]. LXC는 리눅스 커널 기능만을 이용해서 컨테이너 역할을 할 수 있는 소프트웨어이기 때문에 번거로운 작업 없이 커널을 이용하여 컨테이너 기능을 활용할 수 있다는 장점이 있어 많이 활용되고 있으며 컨테이너의 시작점이자 현재 쓰이고 있는 다양한 컨테이너 관련 응용 도구들의 시작점이다. 따라서 본 논문에서는 LXC 기반의 컨테이너 가상화 구조를 기반으로 분석한다.

3.1 커널 계층

컨테이너 기반 가상화에서 커널 계층은 기본적인 인프라를 제공하고, 개별의 컨테이너를 실행하고 관리하는 계층을 말하며 리소스 할당, 컨테이너 격리 그리고 시스템호출(Systemcall) 인터페이스를 통해 컨테이너를 제어하는 역할을 한다[17].

리소스 할당이란 컨테이너가 실행됨에 있어 필요한 리소스들을 각각의 컨테이너에 할당해주는 것을 말한다. CPU, 메모리, 네트워크, 디스크 공간과 같은 리소스를 컨테이너에 할당하며 모든 리소스는 커널의 기능에 따라 컨테이너 계층에서 수행된다.

컨테이너 격리란 각 컨테이너에 독립적인 실행 환경을 보장하기 위해, 커널 계층이 각 컨테이너를 격리하는 것을 말한다. 컨테이너가 독립적인 환경을 구성할 수 있도록 하고 독립적인 환경 구성은 컨테이너 간의 접근을 제어하기 때문에 보안성과 안정성을 향상시킨다. 이에 따라 전체적인 구조로 보았을 때 여러 컨테이너가 동시에 사용되고 있지만 각 컨테이너는 다른 컨테이너의 존재를 인식하지 못하게 된다.

(그림 1)의 시스템호출 인터페이스란 컨테이너가 하드웨어를 직접 관리하지 않고 커널을 통해 리소스를 받을 수 있도록 하는 역할이다. 컨테이너가 직접 하드웨어와 상호 작용할 경우, 다수의 컨테이너가 존재하기 때문에 잘못 사용하는 경우를 방지하기 위해 커널 계층을 통함으로써 오용을 방지하는 기능이라고 할 수 있다. 커널 계층을 통해 각 컨테이너는 여러 컨테이너가 공존하는 환경에서도 독립적인 실행 환경을 제공받을 수 있으며, 시스템 자원을 오남용하는 것을 방지할 수 있다.

위와 같은 격리와 제어는 하이퍼바이저 기반 가상화에서 각 가상머신별로 독립적인 공간을 만들듯이 컨테이너 기반 가상화 환경의 커널 계층에서는 'namespace'와 'cgroup'이라는 기능을 통해 서로가 충돌하지 않도록 한다.

3.1.1 Namespace

Namespace는 프로세스를 실행할 때 컴퓨팅 리소스를 프로세스 별로 분리해서 실행할 수 있도록 도와주는 기능으로, namespace의 종류는 <표 1>과 같다. PID namespace는 각 컨테이너에 대해 독립적인 프로세스 ID 공간을 제공한다. PID namespace를 사용할 때 컨테이너 내부에서는 해당 프로세스가 시스템에서 유일한 프로세스처럼 보일 수 있다. NET namespace는 각 컨테이너에 대해 독립적인 네트워크 스택을 제공한다. 네트워크 스택의 종류에는 IP주소나 라우팅

테이블 등이 있으며 이를 통해 컨테이너는 자신만의 네트워크 인터페이스와 IP 주소, 포트 범위를 가질 수 있다. IPC namespace는 각 컨테이너에 대해 독립적인 시스템V IPC 공간을 제공한다. 이를 통해 컨테이너는 다른 컨테이너와 격리된 형태로 프로세스 간 통신(IPC)을 수행할 수 있다. MNT namespace는 각 컨테이너에 대해 독립적인 파일 시스템 마운트 포인트를 제공하며, 컨테이너는 다른 컨테이너와 격리된 형태로 파일 시스템을 마운트하고 해제할 수 있다. UTS namespace는 각 컨테이너에 대해 독립적인 호스트 이름과 도메인 이름을 제공하며 USER namespace는 각 컨테이너에 대해 독립적인 사용자와 그룹 ID 공간을 제공한다. 이를 통해 컨테이너 내부에서는 루트 사용자로 동작할 수 있지만, 컨테이너 외부에서는 제한된 권한을 가진 사용자로 동작한다. 컨테이너를 사용하는 사용자들이 많은 컨테이너 속에서 자신만의 컨테이너 외에 다른 컨테이너의 존재를 인식하지 못하고 상호작용을 할 수 없는 것은 namespace가 상호작용을 제한하기 때문이라고 할 수 있다.

3.1.2 Cgroup(Control groups)

Cgroup(Control groups)은 Linux 커널의 기능 중 하나로, 프로세스 그룹의 리소스 사용을 제한하고, 격리하고, 측정하고, 제어하는 기능을 제공한다[14]. Cgroup은 CPU, 메모리, 네트워크 대역폭, 디스크 I/O와 같은 컴퓨팅 리소스에 대해 제한을 설정할 수 있게 해준다. Cgroup을 사용해서 컨테이너가 사용할 수 있는 최대 CPU 사용률, 메모리 사용량을 제한 할 수 있다. 제한함으로써 어느 특정 컨테이너가 시스템 리소스를 독점 혹은 과도하게 사용하여 다른 컨테이너

<표 1> namespace 종류

namespace	isolate
PID namespace	프로세스 ID
NET namespace	IP address, ports, network devices
IPC namespace	시스템V IPC
MNT namespace	마운트 포인트
UTS namespace	호스트 이름, 도메인 이름
USER namespace	사용자 및 그룹 ID

나 커널에 영향을 미치는 것을 방지한다. 또한 cgroup은 컨테이너 간의 컴퓨팅 리소스를 격리할 수 있다. 각 컨테이너는 독립적인 리소스 풀을 가질 수 있고, 다른 컨테이너가 공유된 커널 안에서 리소스를 사용할지라도 영향을 받지 않을 수 있다. Cgroup은 컨테이너의 리소스 사용을 측정할 수 있다. 리소스 사용률을 모니터링하며, 특정 컨테이너가 컴퓨팅 리소스를 과도하게 사용하는 것을 감지할 수 있고, cgroup의 컴퓨팅 리소스를 제어하는 기능을 통해 더 많은 메모리 사용량을 할당하거나 줄이는 등 리소스 할당을 동적으로 조정하여 조처를 할 수 있다.

3.1.3 Seccomp(Secure Computing Mode)

Seccomp(Secure Computing Mode)는 커널 계층에서 활용하는 기능으로 프로세스가 사용할 수 있는 시스템 호출을 제한해서 보안을 강화하는 기술이다. Seccomp는 컨테이너 내부의 프로세스가 사용할 수 있는 시스템 호출을 제한하거나 필터링하는 역할을 한다. 이는 컨테이너가 커널에 대한 불필요한 시스템 호출을 수행하는 것을 방지하고, 컨테이너 내부의 프로세스가 커널의 리소스를 과도하게 사용하거나 시스템의 안전을 위협하는 것을 막는 데 도움이 된다.

Seccomp는 두 가지 모드로 작동이 된다. Strict 모드와 filter 모드를 제공하고 있으며, Strict 모드는 제한적인 상태로 프로세스가 사용할 수 있는 시스템 호출을 read(), write(), _exit() 및 sigreturn() 네 가지로 제한한다. 하지만 filter 모드는 사용자가 정의한 필터를 사용하면서 프로세스에 허용할 시스템 호출을 세밀하게 제어할 수 있다. Docker와 같은 컨테이너 엔진에서는 seccomp를 사용해서 컨테이너의 보안을 강화할 수 있고, Docker는 기본적으로 seccomp를 사용해서 시스템 호출 중 일부를 제한할 수 있다. 컨테이너 사용자는 seccomp를 커스터마이징 해서 사용자의 필요에 따라 제한을 추가하거나 제거할 수 있다. Seccomp를 사용하면 컨테이너의 작동 범위를 제한할 수 있고, 컨테이너화된 어플리케이션의 안전성을 향상시킬 수 있다.

3.2 컨테이너 계층

컨테이너 계층은 (그림 1)과 같이 컨테이너 런타임

과 컨테이너 엔진으로 구성된다. 컨테이너 런타임이란 커널 계층과의 상호작용을 통해 컨테이너의 생성, 시작, 실행, 정지, 삭제와 같은 컨테이너 전반의 생명 주기를 관리하는 역할을 한다. 컨테이너 런타임은 실행하기 위해 이미지를 다운로드 하고, 받은 이미지를 번들에 압축 해제한 후 번들에서 컨테이너를 실행하는 단계로 나눌 수 있다. 하지만 가장 널리 쓰이는 컨테이너 런타임 중의 하나인 도커는 컨테이너화된 어플리케이션의 배포와 실행만을 표준화했고, 이에 따라 실제로 컨테이너를 실행하는 런타임인 Low-Level Container Runtimes와 이미지 전송 및 관리와 압축을 해제하는 High-Level Container Runtimes로 나뉘게 되었다. 컨테이너 런타임은 위와 같은 작업을 실행하며 커널 계층과는 상호작용을 하며 컨테이너에 필요한 리소스를 할당하고, 컨테이너의 내부 프로세스를 관리하며, 네트워크를 액세스하는 기능을 한다.

컨테이너 엔진은 컨테이너 런타임을 포함하는 개념으로 사용자 인터페이스와 이미지 관리를 제공하고, 컨테이너의 수명 주기를 관리하는 것뿐만 아니라, 네트워크의 구성, 이미지 저장소 관리의 추가적인 기능을 하며 컨테이너 런타임이 실행, 취소 등의 작업에 초점을 맞추었다면 컨테이너 엔진은 런타임을 포함하는 큰 개념이라고 볼 수 있다. (그림 1)에서 볼 수 있는 컨테이너 런타임의 컨테이너드는 컨테이너 런타임 중 하나로 docker를 중심으로 구글 등 컨테이너 기술에 관심 있는 여러 집단이 한데 모여 만든 OCI(Open Container Initiative) 프로젝트를 준수하는 컨테이너의 표준이라고 할 수 있다. 도커와 컨테이너드의 발전에 따라 컨테이너드 중심으로 바뀌었으며 (그림 1)에서 또한 오케스트레이션 계층에서 컨테이너드로 이어지는 모습을 확인할 수 있다.

3.3 오케스트레이션 계층

컨테이너 오케스트레이션 계층은 여러 컨테이너 간의 작업을 자동화하고 조율하는 계층이다. 오케스트레이션 계층에서 활용하는 오케스트레이션 도구는 클러스터에서 컨테이너 관리를 담당하며 전체 컨테이너 수를 동적으로 늘리거나 줄이는 스케일링 관리를 할 수 있다. 또한 오케스트레이션 도구는 어플리케이션의 네트워크 트래픽을 여러 컨테이너 인스턴스 간에 분산하는 기능을 제공하고, 시스템 오류가 발생했을 때 자동으로 복구할 수 있으며 컨테이너의 리소스 사용량을 모니터링하고 로그를 수집한 후 저장하고 분석하는 기능을 제공하며 커널 계층으로 직접적인 호출을 하지 않는다.

4. 컨테이너 기반 가상화 취약점

4.1 커널 계층

Cgroup과 namespace는 리눅스 커널을 구성하는 기술로 단일 프로세스가 사용할 수 있는 컴퓨팅 리소스를 제한하거나 각 프로세스 사이에 볼 수 있는 리소스의 범위를 제한하여 여러 프로세스를 격리한다. 프로세스는 단독으로 구성할 수 있으며 하위에 여러 프로세스가 만들어질 수 있는 하위 프로세스는 상위 프로세스와 격리되어 단독 프로세스처럼 사용할 수 있다. Cgroup은 root cgroup계층 구조로 구성될 수 있으며 각 컴퓨팅 리소스 별 cgroup에 리소스를 할당한다. 예를 들어 root cgroup 하위에는 CPU를 담당하는 cgroup이 여러 개가 될 수 없지만, 하나 이상의 프로세스 혹은 task는 연결될 수 있다. cgroup에선 각 리소스가 리소스 컨트롤러에 의해 컴퓨팅 리소스에 대해 프로세스별 총 리소스의 양을 제한 할 수 있다. 따라서 리눅스를 기반으로 하는 LXC에서 발생하는 noisy neighbor problem 방지를 위해 cgroup과 namespace를 사용하는 것에 대해 기대할 수 있는 효과는 리눅스에서 cgroup과 namespace를 사용했을 때 얻을 수 있는 효과와 같다. 그러나 리눅스 커널을 단독으로 사용할 때 나타나지 않았던 noisy neighbor problem이 컨테이너 기반 가상화 환경에서 발생하는 이유는 커널을 사용하는 특정 사용자의 리소스 사용량이 cgroup과 namespace에서 할당한 자원을 넘어서는 것에 있음을 알 수 있다.

Noisy neighbor problem으로 인해 한 컨테이너의 리소스 사용이 과도해짐에 따라 다른 컨테이너의 리소스가 부족해질 수 있고, 리소스가 부족해짐에 따라 발생하는 추가적인 결과와 전체 컨테이너 시스템 성능 예측에 대한 사용자의 가용성을 보장하기 어렵다. 따라서 컨테이너 관련 cgroup의 컴퓨팅 리소스 제한에 대한 커널 메커니즘은 cgroup으로 제한된 리소스

를 사용하는 범위까지 신뢰할 수 있다고 할 수 있으며, 커널에서 컴퓨팅 리소스를 제한하고 격리하기 위해서는 추가의 매커니즘 수정이 필요하다. 이를 위해 커널 리소스에 대한 정량적인 할당 혹은 재할당이 필요하지만 이를 컨테이너 사용자에게 맡기거나 커널 관리자가 직접 할당하는 것만으로는 부족하다. 또한 현재 이에 대한 수치적인 분석이 부족하고 커널 리소스를 제한 없이 늘어날 수 있는 컨테이너 사용자마다 정량적으로 할당하는 데 있어 어려움이 존재한다. 공통적으로 컨테이너를 사용하는 사용자에게 시스템 호출 권한을 부여하는 것에 대한 위험성을 강조할 수 있다. 예를 들어 여러 컨테이너 중 하나의 컨테이너 사용자가 시스템호출을 통해 커널에 대한 불필요한 직접 접근으로 컴퓨팅 리소스를 악용할 소지가 있다. 또한 리소스를 분배하면서 정량적인 배분이 불가능하다는 점을 알 수 있다.

4.2 컨테이너 계층

컨테이너 엔진을 실행할 때 호스트 운영체제에서는 기본적으로 시스템호출이 제한되고 필요한 경우에만 특정 호출이 허용된다. 이를 privileged mode라고 한다. 리눅스 커널을 사용하는 컨테이너에서 privileged mode를 사용하게 되면 소프트웨어는 제한이 없는 권한으로 실행된다. 해당 모드가 실행이 되면 소프트웨어가 모든 하드웨어 리소스에 액세스할 수 있도록 하고 이후 리눅스 커널은 해당 모드에서 실행된다. 컨테이너에 모든 기능을 제공하고 cgroup 컨트롤러에 의해 적용되는 모든 제한을 해제할 수 있으며, 컨테이너에서 호스트 운영체제가 할 수 있는 모든 것을 제공할 수 있는 기능이다.

따라서 privileged mode로 실행된 컨테이너는 호스트 운영체제의 모든 디바이스에 접근할 수 있게 되고, 커널 기능에 대해 제한이 없어지기 때문에 컨테이너 내에서 실행되는 프로세스가 호스트 운영체제에서 root 권한으로 작동하는 것처럼 만들 수 있다. 컨테이너가 사용자의 컨트롤에 따라 권한을 부여받는 행위는 컨테이너 사용에 있어 자율성을 높여줄 수 있다. 그러나 모든 권한을 특정 컨테이너가 사용하는 것에 대해 자율성이 높아진 만큼 보안은 낮아지기 때문에 privileged mode는 지양되고 있는 기술 중 하나다[18].

각 컨테이너는 자신만의 실행 환경을 가지고 있고 다른 컨테이너로부터 격리되어 있지만 호스트 운영체제를 공유하고 있기 때문에 privileged mode를 사용하여 특정 컨테이너가 호스트 운영체제의 자원을 남용하는 경우 전체 컨테이너 기반 가상화 환경에 있어 noisy neighbor problem을 일으킬 수 있다. 이에 대해 보안이 강화된 컨테이너 런타임을 설계하는 노력을 하고 있으나, 일례로 Kata 컨테이너의 경우 컨테이너 환경 안에 가상머신을 만드는 것이기 때문에 성능 저하가 불가피하다는 한계가 존재한다[19].

4.3 오케스트레이션 계층

컨테이너 기반 가상화는 가상머신에 비해 가볍고 배포가 빠르다는 특징으로 인해 컨테이너 기반 가상화의 사용이 빠르게 늘었다. 또한 개발자의 관점에서 컨테이너 기반 가상화는 어플리케이션 개발 환경을 빠르게 배포할 수 있다는 장점이 있다. 사람마다 개발 환경이 다르기 때문에 매번 환경을 맞춰야 했으나 컨테이너 이미지에 어플리케이션을 실행하는 데 필요한 라이브러리 등을 사전에 설치해서 배포하면 같은 개발 환경을 얼마든지 제공할 수 있기 때문에 개발 속도를 빠르게 하는 것이 가능하다는 장점이 있다. 특정 어플리케이션에서 실행된 악성파일이 커널 계층으로 영향을 미치는 경우가 존재한다. 컨테이너 이미지에 백도어, 웹쉘, 암호화폐 채굴 프로그램과 같은 악성 소프트웨어가 도커 이미지 개발자에 의해 의도적으로 첨부되는 경우가 존재하기 때문이다. 해당 악성 소프트웨어가 포함된 이미지 파일을 받게 되는 경우 이미지를 기반으로 하는 컨테이너가 호스트 컴퓨팅 시스템에서 실행되면 제공된 가용 리소스를 넘어 사용할 수 있기 때문에 noisy neighbor problem을 일으킬 수 있다.

<표 2>는 컨테이너 기반 가상화를 구성하는 커널, 컨테이너, 오케스트레이션에 대한 계층별 주요 기능과 주요 기능에서의 취약성, 해결 방향 및 한계점을 보여주고 있다. 커널 계층은 cgroup에 대한 취약점이 존재하며 리소스를 정량적으로 할당 및 재할당 되어야 하지만 실제로 정략적인 배분이 불가능하다는 한계가 존재한다. 컨테이너 계층에서 컨테이너 사용자에게 부여되는 privileged mode는 호스트 운영체제를 공유하

게 되므로 위험성을 내포하고 있다. 따라서 privileged mode 사용을 지양하거나 보안이 강화된 컨테이너 런타임 도구를 사용할 필요가 있으나 컨테이너의 가볍고 빠른 배포라는 특성을 대가로 하는 한계가 존재한다. 오케스트레이션 계층에는 이미지를 통한 악성 소프트웨어와 암호화폐 채굴 프로그램 때문에 문제가 발생하고 있으나 검증된 이미지나 이미지의 취약점을 검사하는 방안을 통해 해결되고 있다.

연구로 컨테이너 기반 가상화 구조의 계층별 취약성을 해결방향을 제시하고 있다. 아울러 해결 방향에 대한 한계점도 제시하고 있지만 컨테이너 기반 가상화의 성능을 유지하는 해결 방향을 제시하지 않고 있다. 따라서 컨테이너 기반 가상화에서 noisy neighbor problem 관점에서 격리성 보장과 함께 성능을 유지할 수 있는 해결 방안에 대한 연구가 기본적으로 이어져야 한다.

<표 2> 컨테이너 기반 가상화의 계층별 주요 기능, 취약성, 해결 방향 및 한계

계층	주요 기능	취약성	해결 방향	한계
커널	• 인프라 제공 • 컨테이너 관리 • cgroup • namespace • seccomp • 시스템호출	• cgroup • 시스템호출 권한	• 리소스 정량적 할당 • 리소스 재할당 • 시스템호출 권한 제한	• 컨테이너별 리소스의 정량 분배 불가능
컨테이너	• 컨테이너 생명 주기 관리 • privileged mode	• privileged mode	• privileged mode 사용 지양 • 보안이 강화된 컨테이너 런타임 도구 사용(gVisor, Kata container 등)	• 사용자 편의성, 시간 및 리소스 절약 포기
오케스트레이션	• 컨테이너 작업 자동화 • 컨테이너 조율	• 악성 소프트웨어 • 암호화폐 채굴 프로그램	• 검증된 이미지 사용 • 취약점 검사 제공	

5. 결 론

Noisy neighbor problem은 컨테이너의 격리성 보장과 성능에 영향을 미치는 문제인 동시에 다른 컨테이너 사용자의 가용성을 저해할 수 있는 보안 관점의 문제이다. 본 연구에서는 컨테이너 기반 가상화에서 발생할 수 있는 noisy neighbor problem 관점에서 격리성 보장을 위협할 수 있는 취약성을 고찰하고 해결 방안 및 한계점을 제시하고자 하였다.

본 연구는 보안 관점에서 취약성 해결을 위한 선행

참고문헌

[1] F. Sabahi, "Virtualization-level security in cloud computing," 2011 IEEE 3rd International Conference on Communication Software and Networks, Xi'an, China, pp. 250-254, 2011.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and

Technology," NIST Special Publication, 2011.

[3] K. Brady, S. Moon, T. Nguyen and J. Coffman, "Docker Container Security in Cloud Computing," 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, pp. 0975-0980, 2020.

[4] C. Pahl, "Containerization and the PaaS Cloud," in IEEE Cloud Computing, vol. 2, no. 3, pp. 24-31, May-June. 2015.

[5] D. Williams, R. Koller, and B. Lum.. "Say goodbye to virtualization for a safer cloud". In Proceedings of the 10th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'18). USENIX Association, USA, 2018.

[6] T. Lorido-Botran, S. Huerta, L. Tomás, J. Tordsson, B. Sanz, "An unsupervised approach to online noisy-neighbor detection in cloud data centers," Expert Systems with Applications, Volume 89, Pages 188-204, 2017.

[7] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, USA, 2010, pp. 51-58, 2010.

[8] O. AbdElRahem, A. M. Bahaa-Eldin and A. Taha, "Virtualization security: A survey," 2016 11th International Conference on Computer Engineering & Systems (ICCES), Cairo, Egypt, pp. 32-40, 2016.

[9] J. Sahoo, S. Mohapatra and R. Lath, "Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues," 2010 Second International Conference on Computer and Network Technology, Bangkok, Thailand, pp. 222-226, 2010.

[10] M. Souppaya, J. Morello, K. Scarfon, "Application Container Security Guide", National Institute of Standards and Technology(NIST), 2017.

[11] N. G. Bachiega, P. S. L. Souza, S. M. Bruschi and S. d. R. S. de Souza, "Container-Based Performance Evaluation: A Survey and Challenges," 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, USA, pp. 398-403, 2018.

[12] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," in IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, Sept. 2014.

[13] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis and H. Wang, "ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds," 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, CO, USA, pp. 237-248, 2017.

[14] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou. "A Measurement Study on Linux Container Security: Attacks and Countermeasures" In Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18). Association for Computing Machinery, New York, NY, USA, 418 - 429, 2018.

[15] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang.. "Houdini's Escape: Breaking the Resource Rein of Linux Control Groups". In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), NY, USA, 1073 - 1086, 2019.

[16] Ruan, B., Huang, H., Wu, S., Jin, H. "A Performance Study of Containers in Cloud Environment." Advances in Services Computing. APSCC 2016. Lecture Notes in Computer Science, vol 10065. Springer, 2016.

[17] Y. Yang, W. Shen, B. Ruan, W. Liu and K. Ren, "Security Challenges in the Container Cloud," 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), Atlanta,

GA, USA, pp. 137-145, 2021.
[18] V. V. Sarkale, P. Rad and W. Lee, "Secure Cloud Container: Runtime Behavior Monitoring Using Most Privileged Container (MPC)," 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, USA, pp. 351-356, 2017.
[19] Wang, X., Du, J. & Liu, H. "Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes". Cluster Comput 25, pp. 1497 - 1513, 2022.

〔저자소개〕

염 다 연 (Da-yun Yum)
2019년 2월 학사
2021년 2월 석사
2021년 9월 박사 재학
email : dyum@cau.ac.kr

신 동 천 (Dong-Cheon Shin)
1985년 2월 학사
1987년 2월 석사
1991년 2월 박사
email : dcshin@cau.ac.kr