The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.9, pp.404~412

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.9.404

eBPF-based Container Activity Analysis System

Jisu Kim† · Jaehyun Nam††

ABSTRACT

The adoption of cloud environments has revolutionized application deployment and management, with microservices architecture and container technology serving as key enablers of this transformation. However, these advancements have introduced new challenges, particularly the necessity to precisely understand service interactions and conduct detailed analyses of internal processes within complex service environments such as microservices. Traditional monitoring techniques have proven inadequate in effectively analyzing these complex environments, leading to increased interest in eBPF (extended Berkeley Packet Filter) technology as a solution. eBPF is a powerful tool capable of real-time event collection and analysis within the Linux kernel, enabling the monitoring of various events, including file system activities within the kernel space. This paper proposes a container activity analysis system based on eBPF, which monitors events occurring in the kernel space of both containers and host systems in real-time and analyzes the collected data. Furthermore, this paper conducts a comparative analysis of prominent eBPF-based container monitoring systems (Tetragon, Falco, and Tracee), focusing on aspects such as event detection methods, default policy application, event type identification, and system call blocking and alert generation. Through this evaluation, the paper identifies the strengths and weaknesses of each system and determines the necessary features for effective container process monitoring and restriction. In addition, the proposed system is evaluated in terms of container metadata collection, internal activity monitoring, and system metadata integration, and the effectiveness and future potential of eBPF-based monitoring systems.

Keywords : Container, eBPF, Monitoring, Observability

eBPF를 활용한 컨테이너 활동 분석 시스템

김 지 수† · 남 재 현††

요 약

클라우드 환경의 도입은 애플리케이션 배포와 관리 방식을 혁신적으로 변화시켰으며, 마이크로서비스 아키텍처와 컨테이너 기술이 그 중심에 자리 잡고 있다. 그러나 이러한 변화는 마이크로서비스와 같은 복잡한 서비스 환경에서 서비스 간 상호작용을 정확히 파악하고 내부 프로세스를 세밀하게 분석해야 하는 새로운 과제를 제기했다. 기존의 모니터링 기법은 이와 같은 복잡한 환경을 충분히 분석하는 데 한계를 보였으며, 이를 해결하기 위해 최근 eBPF(extended Berkeley Packet Filter) 기술이 주목받고 있다. eBPF는 리눅스 커널에서 발생하는 이벤트를 실시간으로 수집하고 분석할 수 있는 강력한 기술로, 커널 영역에서 발생하는 파일 시스템 활동을 비롯한 다양한 이벤트를 모니터링할 수 있다. 본 논문에서는 이러한 eBPF를 기반으로 한 컨테이너 활동 분석 시스템을 제안하며, 이 시스템이 컨테이너와 호스트 시스템의 커널 영역에서 발생하는 이벤트를 실시간으로 모니터링하고, 수집된 데이터를 분석하는 방법을 제시한다. 또한, eBPF를 활용한 대표적인 컨테이너 모니터링 시스템인 Tetragon, Falco, Tracee를 이벤트 탐지 방식, 기본 정책 적용 여부, 이벤트 유형 식별, 시스템 콜 차단 및 경고 생성 기능 측면에서 비교 분석하여 각 시스템의 장단점을 평가하고, 컨테이너 프로세스 감시 및 제한에 필요한 기능을 파악하였다. 나아가, 본 논문에서는 제안된 시스템을 컨테이너 메타데이터 수집, 내부 활동 모니터링, 시스템 메타데이터 연계의 세 가지 측면에서 설명하고 평가하며, 관련 연구와의 비교를 통해 eBPF 기반 모니터링 시스템의 효용성과 향후 발전 가능성에 대해 논의한다.

키워드 : 컨테이너, eBPF, 모니터링, 관측 가능성

1. 서 론

현대 사회에서 많은 기업이 접근성 및 확장성을 이유로 클라우드 환경을 구축하고 있다. 클라우드 환경은 자원의 효율적인 사용을 가능하게 하며, 필요에 따라 자원을 유동적으로 관리할 수 있다는 장점이 있다. 이러한 클라우드 환경의 도입

※ 이 성과는 2024년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2023-00212738).
※ 이 논문은 2024년 ACK 2024의 일반논문으로 “eBPF를 활용한 실시간 컨테이너 모니터링 시스템”의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 단국대학교 인공지능융합학과 석사과정
†† 종신회원 : 단국대학교 컴퓨터공학과 조교수
Manuscript Received : July 9, 2024
Accepted : July 31, 2024
* Corresponding Author : Jaehyun Nam(namjh@dankook.ac.kr)

으로 인해 애플리케이션 배포 및 관리 방식 또한 변화하였다. 해당 변화의 대표적인 예로 마이크로서비스 아키텍처와 컨테이너 기술이 있다[1].

마이크로서비스 아키텍처는 애플리케이션을 작고 독립적인 서비스 단위로 분리하여 개발 및 배포하는 방식을 의미한다. 각 서비스는 독립적으로 개발, 배포, 확장이 가능하므로 이를 통해 애플리케이션 자체의 유연성을 높이고, 개발 주기를 단축할 수 있다. 그러나 마이크로서비스 아키텍처의 도입은 각 서비스 간의 상호작용을 관리하고 모니터링하는 새로운 도전 과제를 동반한다.

컨테이너 기술은 마이크로서비스 아키텍처를 지원하는 핵심 기술로, 애플리케이션과 그 종속성을 함께 패키징하여 일관된 실행 환경을 제공한다. 또한, 경량화된 특성으로 인해 짧은 수명 주기를 가지며, 빠르게 생성되고 삭제될 수 있다. 이러한 특성을 사용하여 컨테이너는 마이크로서비스 아키텍처의 서비스를 구성할 수 있지만, 컨테이너 내부에서 발생한 활동을 분석하는 작업의 중요성이 커지게 된다. 따라서, 컨테이너의 효율적인 관리와 보안을 위해 정교한 모니터링 도구의 필요성이 강조되고 있다[2].

컨테이너를 통해 마이크로서비스를 구축할 경우 전통적인 서버에서 이뤄졌던 서버 모니터링과는 다른 접근이 필요하다. 컨테이너는 기존의 가상 머신과 달리 호스트 시스템의 커널 자원을 공유하기 때문에, 커널 레벨에서의 모니터링이 중요한 요소로 작용한다. 이러한 배경에서 eBPF(extended Berkeley Packet Filter)[3]는 효율적이고 확장 가능한 모니터링 기법으로 주목받고 있다.

eBPF는 리눅스 커널 내에서 실행되는 경량 프로그램을 작성하고 배포할 수 있게 해주는 기술이다. eBPF 프로그램은 커널의 특정 이벤트가 발생했을 때 실행되며, 실시간으로 데이터를 수집 및 분석할 수 있다. 이를 통해 프로세스 실행, 네트워크 트래픽, 파일 시스템 활동 등 다양한 커널 레벨의 정보를 수집할 수 있다. 이러한 특성 덕분에 eBPF는 컨테이너 환경에서의 활동을 모니터링하고 분석하는 데 유용한 도구로 사용될 수 있다[4].

본 논문에서는 eBPF를 활용한 컨테이너 활동 분석 시스템을 제안한다. 제안하는 시스템은 컨테이너 내부 및 호스트 시스템의 커널 영역에서 발생하는 이벤트를 실시간으로 모니터링하고, 수집된 데이터를 기반으로 컨테이너 활동을 분석하는 시스템이다. 이를 토대로 컨테이너 환경 운영자는 운영 환경에서 발생할 수 있는 다양한 문제를 조기에 발견하고 대비할 수 있다. 따라서 본 논문은 제안한 프로그램을 통해 eBPF 기반 모니터링 시스템의 효용성을 입증한다.

본 논문의 구성은 다음과 같다. 2장에서는 컨테이너 기술과 eBPF에 대한 배경지식을 소개한다. 3장에서는 eBPF 기술을 활용해 컨테이너 보호를 제공하는 솔루션인 Tetragon, Falco, Cilium을 여러 관점에서 비교 분석한다. 이후 4, 5, 6장에서는 본 연구의 동작 흐름 설명, 구현 및 평가에 대해 다루며, 7장에서는 eBPF를 활용하여 컨테이너를 모니터링하는 기술을 본 연구와 비교한다. 마지막으로 8장에서 결론을 맺는다.

2. 배경 지식

2.1 컨테이너 시스템

컨테이너 시스템은 애플리케이션과 그 종속성을 하나의 패키지로 묶어 일관된 실행 환경을 제공하는 기술이다. 컨테이너는 가상 머신과 달리 호스트 운영 체제의 커널 자원을 공유하며, 경량화된 특성으로 인해 짧은 수명 주기를 가지며, 빠르게 생성되고 삭제될 수 있다.

컨테이너를 운영하는 호스트 컴퓨터에서는 호스트 프로세스와 컨테이너 프로세스가 동시에 실행되고 있으며, 컨테이너 내부에도 다양한 프로세스가 동작한다. 호스트 프로세스와 컨테이너 프로세스의 주요 차이점은 실행 환경과 자원 격리에 있다. 호스트 프로세스는 호스트 운영체제의 자원을 직접 사용하며, 전체 시스템의 자원에 접근할 수 있다. 반면, 컨테이너 프로세스는 컨테이너 내에서 격리된 환경에서 실행되며, 호스트 운영체제의 자원을 제한적으로 사용한다. 이처럼 독립적인 실행 환경을 제공하기 위해서 컨테이너는 Cgroups와 Linux 네임스페이스를 활용한다. Cgroups는 프로세스 그룹의 자원 사용량(CPU, 메모리, 디스크 I/O 등)을 제한하고 추적하는 기능을 제공한다. Linux 네임스페이스는 컨테이너가 고유한 프로세스 ID, 파일 시스템 등을 가지도록 한다. 특히 PID(Process ID)와 MNT(Mount) 네임스페이스는 컨테이너 환경에서 호스트 프로세스와 컨테이너 프로세스를 구분하는 데 중요한 역할을 한다. PID 네임스페이스는 각 프로세스에 고유한 프로세스 ID 공간을 제공한다. 호스트 시스템에서 실행되는 프로세스는 호스트의 PID 네임스페이스에 속하며, 컨테이너 내에서 실행되는 프로세스는 컨테이너의 PID 네임스페이스에 속한다. 이를 통해 같은 PID를 가지더라도 네임스페이스가 다르면 서로 다른 프로세스로 취급된다. MNT 네임스페이스는 파일 시스템의 마운트 지점을 격리한다. 각 네임스페이스는 고유한 파일 시스템 구조를 지니며, 특정 마운트 포인트를 독립적으로 관리할 수 있다. 호스트 시스템의 파일 시스템과 컨테이너의 파일 시스템은 다른 MNT 네임스페이스에 속하기 때문에, 컨테이너 내 파일 시스템 변경이 호스트 시스템에 영향을 미치지 않는다. 이를 통해 호스트 프로세스와 컨테이너 프로세스는 파일 시스템 수준에서 격리된 환경을 가진다[5].

2.2 eBPF

eBPF는 리눅스 커널 내에서 작동하는 경량 프로그램을 작성하고 실행할 수 있는 기술이다. eBPF 프로그램은 커널의 특정 이벤트가 발생할 때 실행되며, 실시간으로 데이터를 수집하고 분석할 수 있다. 그뿐만 아니라 커널 내에서 직접 실행되므로 사용자 공간에서 실행되는 프로그램에 비해 오버헤드가

적으며, 커널의 안정성을 보장하기 위해 엄격한 검증 절차를 거친다.

eBPF는 kprobe와 kretprobe를 활용하여 커널 함수의 진입점과 반환점을 추적한다. kprobe는 커널 함수가 호출될 때 실행되는 프로브로 함수 호출 시점에 접근한 프로세스의 데이터를 수집할 수 있다. kretprobe는 커널 함수가 반환될 때 실행되는 프로브로 함수가 반환되었을 때 반환값을 수집할 수 있다. 이 두 가지 디버깅 메커니즘을 통해 컨테이너 프로세스가 동작하는 방식을 상세히 모니터링할 수 있다[6].

3. 컨테이너 활동 모니터링 시스템

3.1 기존 컨테이너 보호 시스템의 특징

리눅스 커널 컨테이너 기술은 애플리케이션의 배포와 관리에 큰 유연성을 제공하지만, 그 특성상 새로운 보안 문제를 초래할 수 있다. 전통적인 보안 솔루션의 경우 주로 물리 서버 환경을 대상으로 설계되었기 때문에 컨테이너 환경에 도입할 경우 오버헤드가 크게 발생하는 등 여러 한계점을 지닌다. 전통적인 보안 솔루션은 호스트 기반 침입 탐지 시스템(HIDS), 엔드포인트 보호 플랫폼(EPP), 파일 무결성 모니터링(FIM) 등 다양한 역할을 수행해왔다. 예를 들어, OSSEC[7]는 시스템 파일의 무결성, 로그 파일, 설정 파일 등을 모니터링하여 비정상적인 활동을 탐지하고, Symantec Endpoint Protection[8]는 악성 소프트웨어 탐지, 방화벽, 침입 방지 등을 제공하여 엔드포인트를 보호한다. 그러나 이러한 솔루션들은 컨테이너 환경에 도입될 경우 여러 한계점을 가진다. 컨테이너는 경량화된 특성으로 인해 짧은 수명 주기를 가지며, 빠르게 생성되고 삭제된다. 전통적인 보안 솔루션은 이러한 동적 특성에 적응하기 어렵고, 컨테이너의 빠른 생성 및 삭제에 따른 보안 정책 적용의 유연성이 부족하다. 또한, 컨테이너는 호스트 운영 체제의 커널 자원을 공유하기 때문에 커널 레벨에서의 모니터링이 중요한데, 전통적인 보안 솔루션은 커널 내부의 이벤트를 실시간으로 모니터링하는 데 한계가 있다[9]. 이러한 이유로 인해, 컨테이너 환경에 특화된 다양한 보안 솔루션이 개발되고 있으며, 그중에서도 Tetragon, Falco, Tracee는 대표적인 도구로 주목받고 있다. 해당 절에선 해당 도구들의 동작 방식과 주요 특징을 분석한다.

1) Tetragon

Tetragon[10]은 실시간 eBPF 기반 보안 가시성과 런타임 정책 집행을 제공하는 도구이다. 이는 정책과 필터링을 직접 eBPF로 적용하여 관찰 오버헤드를 줄이고, 모든 프로세스를 추적하며, 정책을 실시간으로 강제 실행할 수 있다. 또한, tracepoint와 kprobe를 활용하여 리눅스 커널의 다양한 호출을 관찰하고, 함수의 인자 및 반환 값을 포함한 다양한 메타데이터를 수집한다. 이를 통해 프로세스의 생명 주기를 분석하여 실행과 종료를 추적하며, 파일 접근 경로를 모니터링하고,

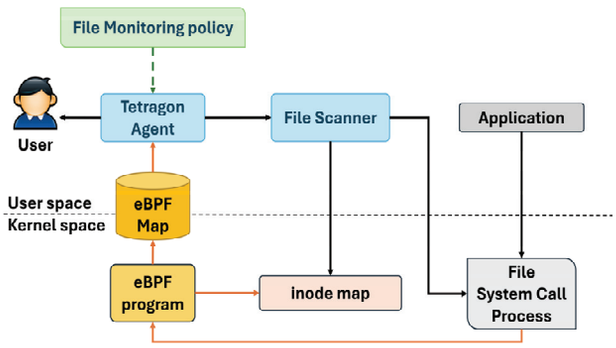

Fig. 1. Tetragon Architecture

TCP 연결을 통해 네트워크 가시성을 제공한다. 또한, Linux 프로세스의 사용자 정보(UID, GID 등)에 접근하여 사용자 정책과 결합하고, 커널에 실시간으로 정책을 생성하고 적용할 수 있다.

Tetragon은 Fig. 1의 구조로 되어 있다. 예를 들어 애플리케이션이 특정 파일에 접근한다면 이를 감시하기 위해 파일 스캐너가 파일 시스템을 스캔하여 인오드를 eBPF 맵에 저장한다. eBPF 프로그램은 이를 참조하여 정책에 따라 접근을 모니터링하고, 해당 접근에 대해 적절한 조치를 해 사용자에게 이벤트를 알린다.

2) Falco

Falco[11]는 리눅스 커널과 컨테이너 런타임의 행동을 모니터링하여 의심스러운 활동을 탐지하고 경고를 제공하는 클라우드 네이티브 보안 도구이다. 해당 도구는 eBPF를 활용하여 커널 이벤트와 시스템 콜을 실시간으로 모니터링하고 사용자가 정의한 정책에 따라 비정상적인 활동을 플래그로 표시하여 경고를 나타낸다. 또한, 클라우드 네이티브 시스템에서 규정을 준수하도록 설계되어 있으며, 규칙 기반 탐지를 통해 비정상적인 활동 및 잠재적인 위협을 식별한다. 그 밖에 플러그인을 사용하여 기능을 확장할 수 있으며, 외부 시스템과 통합하여 알림을 전송하거나 추가 데이터를 수집할 수 있다.

Falco는 (Fig. 2)의 구조로 되어있다. 커널 영역에서 발생하는 이벤트는 eBPF 프로그램을 통해 비동기식으로 수집된다. 이후 Libscap와 Libsinsp 라이브러리[12]에서 캡처 관리, OS 상태 수집 및 출력 형식화 등을 수행한다. 최종적으로 Falco 애플리케이션이 의심스러운 이벤트 정보를 비사용자에게 제공한다.

3) Tracee

Tracee[13]는 시스템과 애플리케이션의 동작을 이해할 수 있도록 돕는 런타임 보안 및 가시성 도구이다. eBPF 기술을 사용하여 시스템에서 발생하는 다양한 이벤트를 실시간으로 추적하고, 이를 기반으로 의심스러운 행동 패턴을 감지한다. 해당 도구는 tracepoints와 kprobe를 통해 330개 이상의 시

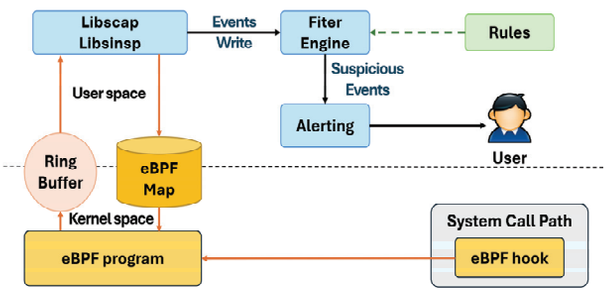

Fig. 2. Falco Architecture

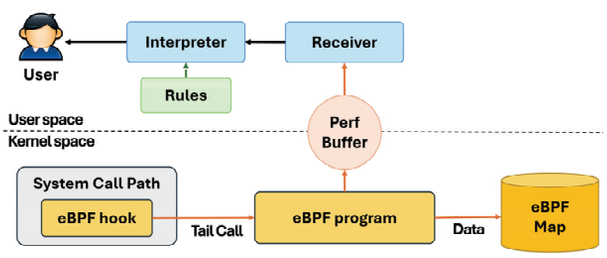

Fig. 3. Tracee Architecture

스템 호출과 이벤트를 실시간으로 생성하고 추적할 수 있으므로 커널에서 발생한 시스템 활동을 상세히 모니터링할 수 있다. 이를 통해 프로세스 실행, 파일 접근, 네트워크 활동 등 다양한 커널 레벨의 정보를 수집한다. 또한, 탐지된 이벤트를 관련성과 우선 순위에 따라 필터링하고 JSON 등 원하는 형식으로 결과를 출력할 수 있다.

Tracee는 Fig. 3의 구조로 되어있다. 여러 CPU가 동시에 동일한 데이터에 접근하거나 수정하는 경쟁 조건을 피하기 위해 Percpu map을 사용한다. 또한, 큰 데이터를 효율적으로 처리하기 위해 Tail Call과 Perf Buffer를 이용하여 반복적으로 데이터를 전송한다.

3.2 서비스 비교 분석

Tetragon, Falco, Tracee 모두 커널 내부에서 발생하는 이벤트를 감지하며 이를 바탕으로 다양한 보안 기능을 제공한다. 그러나 각 컨테이너 보호 시스템이 이벤트를 식별하는 방식이나 제공하는 서비스는 상이하다. 이번 절에선 이벤트 탐지 방식, 기본 정책 적용 여부, 이벤트 유형 식별, 시스템 콜 차단 및 경고 생성 기능에서 이들 시스템을 비교·분석하며 〈Table 1〉은 해당 내용을 정리한 것이다.

1) 이벤트 탐지 방식

컨테이너 프로세스는 컨테이너를 운영하는 호스트의 리눅스 커널을 공유하기 때문에, 컨테이너 내부에서 발생하는 다양한 이벤트와 동작을 추적하려면 호스트 시스템의 커널에 대한 이해가 필요하다. 따라서 리눅스 시스템의 추적 도구와 기술을 활용하여 컨테이너와 관련된 보안 및 성능 이벤트를 모니터링하고 탐지할 필요가 있다.

Tetragon, Falco, Tracee는 이러한 요구를 충족시키기 위해 eBPF 기술을 활용한다. 이를 통해 커널 내부에서 직접 실행되는 프로그램을 통해 커널의 동작을 실시간으로 모니터링하고, 시스템 호출 등 다양한 작업을 효율적으로 추적할 수 있게 해준다. 이에 더하여 Falco는 eBPF 외에도 커널 모듈 또한 활용하여 커널에서 발생한 이벤트를 식별한다. 커널 모듈은 커널과의 직접적인 통신을 가능하게 하며, 커널 레벨에서의 이벤트 추적 및 필터링을 지원한다. 이로 인해, Falco는 커널 모듈을 통해 커널 레벨에서의 이벤트를 수집하고 분석할 수 있는 능력을 갖추고 있다. 이러한 접근 방식은 eBPF 기반의 접근 방식과 상호 보완적으로 작용할 수 있다.

2) 기본 정책 적용 여부

컨테이너의 효과적인 운영을 위해, 컨테이너와 연관된 정책을 이해하고 적용하는 것은 중요한 고려사항이다. 기본 정책 적용 여부는 시스템이 제공하는 보안 기본 설정과 기본적인 규칙의 유용성을 결정짓는 요소로, 이를 통해 정책이 기본적으로 어떻게 설정되는지를 이해할 수 있다.

Falco의 경우, 사용자에게 기본 정책을 제공한다. 여기서 기본 정책이란, 시스템의 보안 상태를 실시간으로 모니터링하고 다양한 보안 이벤트를 탐지하기 위해 자체 시스템에서 미리 정의된 규칙 세트를 의미한다. 이러한 기본 정책은 악성 활동, 비정상적인 시스템 호출, 권한 상승 시도 등 다양한 보안 위협을 식별하는 데 유용하다. Falco의 기본 정책은 커널 이벤트와 시스템 호출을 분석하여, 의심스러운 행동을 탐지하고 경고를 발생시키는 기능을 지원한다. 이로 인해 사용자는 신속하게 보안 위협에 대응할 수 있으며, 기본 정책을 통해 높은 수준의 보안을 유지할 수 있다. 반면, Tetragon과 Tracee는 기본 정책을 제공하지 않는다.

3) 이벤트 유형 식별

각 시스템은 이벤트 유형을 다양한 카테고리로 분류하여

Table 1. Comparison of Container Security Monitoring Systems

Systems	Observation Mechanism	Pre-configured Rule Set	Event Type Classification	Supports Event Alerting and Blocking
Tetragon	eBPF	None	Self-Categorization using System Call Names	Yes (Alerting and Blocking)
Falco	eBPF and Kernel Module	Comprehensive Default Rules	Only System Call Names	Only Alerting
Tracee	eBPF	None	Only System Call Names	Only Alerting

```
"timestamp": "2024-06-27T12:34:56Z",
"event_type": "PROCESS_KPROBE",
"pid": 1234,
"uid": 1001,
"process_name": "/usr/bin/python3",
"parent_pid": 567,
"cwd": "/home/user/projects",
"kprobe_function": "do_sys_open",
"args": ["/usr/bin/python3", "script.py"]
```


Fig. 4. Tetragon Event Classification

로그에 반영한다. 이러한 분류는 사용된 시스템 콜에 따라 파일, 네트워크, 프로세스 등 각기 다른 이벤트를 명확히 식별하고 분석할 수 있도록 돕는다.

Tetragon은 Fig. 4와 같이 다양한 시스템 콜을 감지하고 이를 분류하여 기록한다. 주요 이벤트 유형으로는 PROCESS_EXEC, PROCESS_EXIT 등이 있다.

Falco와 Tracee도 시스템 콜 이벤트를 식별하지만, Tetragon과는 다르게 단순히 시스템 콜 명만을 로그에 출력한다. Falco와 Tracee는 사용된 시스템 콜 명을 통해 이벤트를 식별할 수 있지만, 추가적인 분류 기능을 제공하지 않는다. 이는 사용자가 시스템 콜 명만을 가지고 이벤트를 분류해야 한다는 것을 의미한다.

4) 시스템 콜 차단 및 경고 생성 기능

시스템 콜은 사용자 공간의 프로그램이 운영체제의 커널 공간과 상호작용할 수 있도록 하는 인터페이스이다. 시스템 콜을 통해 프로그램은 파일 시스템 접근, 프로세스 제어, 메모리 관리와 같은 저수준 작업을 수행할 수 있다. 각 시스템은 특정 시스템 콜 접근 시 행동을 차단하거나 경고를 생성할 수 있는 기능을 제공한다. 이는 시스템 보안을 강화하고, 비정상적인 활동을 실시간으로 감지하고 대응하기 위해 중요한 요소이다. 여기서는 차단 및 경고 생성 기능을 두 가지 관점에서 비교한다.

Tetragon은 Fig. 5와 같이 eBPF를 활용하여 특정 시스템 콜 차단과 경고 생성이 가능하다. 이는 사용자가 작성한 정책을 통해 설정되며, 명시한 조건을 충족할 경우 해당 기능이 동작한다. 반면 Falco와 Tracee는 차단 기능을 제공하지 않지만 특정 행위를 수행했다는 경고를 생성하기 위해 두 시스템은 YAML 및 관련 CLI를 사용할 수 있다.

4. eBPF를 활용한 컨테이너 활동 분석 시스템

3장에서 살펴보았듯이 Tetragon, Falco, Tracee 세 가지 시스템 모두 컨테이너 프로세스의 활동을 추적하여 컨테이너 운영 환경의 보안 수준을 높이려는 공통된 목표를 가지고 있었으나 각 시스템이 이벤트를 식별하는 방식과 제공하는 서비스에선 차이가 존재했다. 이러한 차이점을 고려하여 각 시스템의 장점을 조합하면 더욱 효과적인 컨테이너 활동 분석 시스템을 만들 수 있다. 특히, 해당 시스템이 일반 호스트 프로


```
apiVersion: cilium.io/v1alpha1
kind: TracingPolicyNamespaced
spec:
 kprobes:
 - call: "security_file_permission"
  syscall: false
  return: true
  args:
  - index: 0
   type: "file"
  returnArgAction: "Post"
  selectors:
  - matchArgs:
   - index: 0
    operator: "Prefix"
    values:
    - "/boot"
   matchActions:
   - action: Sigkill
```

```
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
spec:
 kprobes:
 - call: "security_file_permission"
  syscall: false
  return: true
  args:
  - index: 0
   type: "file"
  returnArgAction: "Post"
  selectors:
  - matchArgs:
   - index: 0
    operator: "Prefix"
    values:
    - "/boot"
```


Fig. 5. Blocking and Monitoring in Tetragon Using YAML

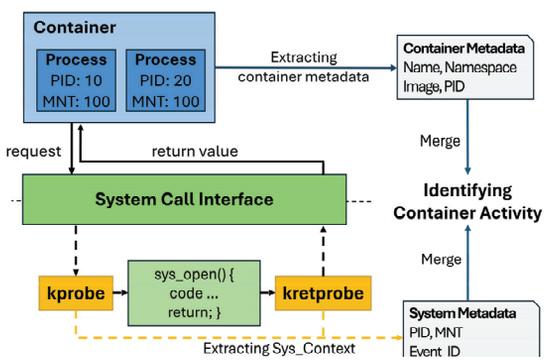

Fig. 6. The Overall Workflow

세스가 아닌 컨테이너 프로세스만을 추적하도록 설계하면 성능 저하를 줄일 수 있으며, 서비스 활동에 따라 시스템 콜을 분류하고 특정 시스템 콜에 접근한 프로세스의 정보를 획득하면 컨테이너 운영 환경을 더욱 효과적으로 보호할 수 있다.

따라서 본 연구에서는 eBPF를 활용하여 호스트 커널 영역에서 발생하는 컨테이너 내부 프로세스의 활동을 감시하는 시스템을 제안한다. 해당 시스템의 전체 구조는 Fig. 6과 같으며 이를 구현하기 위해 다음과 같은 3단계를 거쳐 컨테이너 모니터링 시스템을 구축한다.

4.1 컨테이너 메타데이터 수집

호스트 내에서는 다수의 컨테이너가 동작한다. 호스트 커널 영역에서 컨테이너 프로세스의 활동을 모니터링하기 위해서는 실시간으로 상태가 변화하는 모든 컨테이너의 정보를 수집할 필요가 있다.

Fig. 7과 같이 호스트에서는 일반 호스트 프로세스와 컨테이너 프로세스가 함께 동작한다. 이때 두 프로세스는 같은 커널을 이용하여 실행되며, 커널에서는 두 프로세스를 동일하게 취급하기 때문에 이들을 구분하는 것이 중요하다. 따라서 컨테이너 런타임 인터페이스를 통해 현재 호스트에서 운영 중인 컨테이너의 메타데이터(Namespace, Pod, Container 이름 등)을 수집하는 과정을 거친다. 또한, 컨테이너 메타데이터는

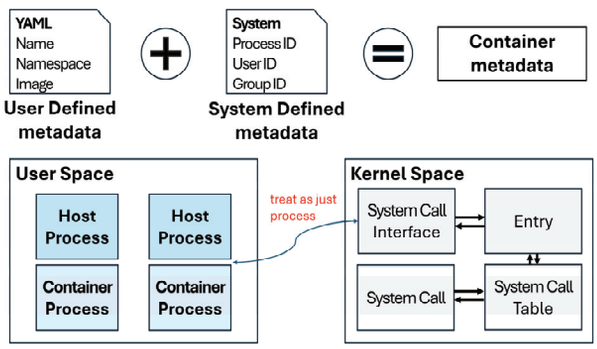

Fig. 7. Container Metadata Collection

사용자가 YAML 파일 등을 통해 작성한 사용자 정의 메타데이터이므로, 추후 시스템 메타데이터(PID, 시스템 콜 정보 등)와의 연계를 위해 시스템이 부여한 개별 컨테이너의 Init 프로세스 PID 정보도 수집한다. 이를 통해 특정 컨테이너에 대한 사용자 메타데이터와 시스템 메타데이터를 연관시킬 수 있다[14,15].

4.2 컨테이너 내부 활동 모니터링

컨테이너의 프로세스는 가상 머신과 달리 호스트 커널을 공유하기 때문에 호스트 커널 영역에 훅을 삽입함으로써 컨테이너의 프로세스 동작을 모니터링할 수 있다. 따라서 시스템 콜 모니터링을 위해 eBPF 프로그램의 kprobe와 kretprobe를 이용하여 훅을 호스트 커널 영역에 삽입한다. Fig. 8에서 볼 수 있듯이 kprobe와 kretprobe는 프로세스가 특정 시스템 콜을 호출할 때와 반환할 때의 이벤트 정보를 얻을 수 있는 메커니즘이다. 이를 통해 호스트 내에서 실행 중인 컨테이너가 호출한 시스템 콜을 모니터링하여 컨테이너 내부 프로세스의 동작 흐름을 추적할 수 있다.

호스트 커널을 공유하는 컨테이너의 특징으로 인해 단순히 시스템 콜 정보만으로는 시스템 콜을 호출한 프로세스가 일반 호스트 프로세스인지, 컨테이너 프로세스인지 식별하기 어렵다. 이는 시스템 입장에서는 컨테이너 프로세스와 호스트 일반 프로세스 모두를 동일하게 단순 프로세스로 취급하기 때문이다. 따라서 호스트 커널 내에서 프로세스와 파일 시스템을 고유하게 식별할 수 있는 PID와 MNT 네임스페이스

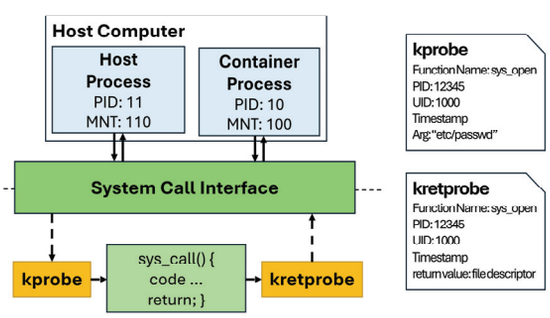

Fig 8. Host Computer Process Monitoring

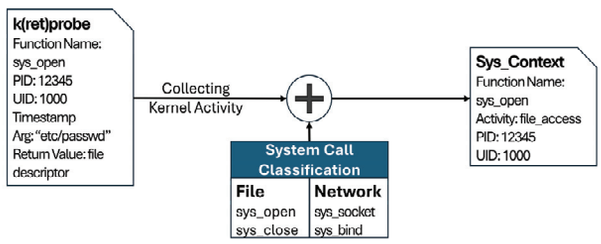

Fig. 9. Activity Classification Based on System Calls

Fig. 10. Results of System Call Classification

정보를 활용한다. PID 네임스페이스를 통해 동일한 PID를 가지더라도 서로 다른 네임스페이스에 속하는 프로세스를 구분할 수 있다. MNT 네임스페이스는 파일 시스템 마운트 지점을 격리하여, 다른 네임스페이스에서 독립된 파일 시스템 구조를 유지하도록 한다. 이를 활용함으로써 일반 프로세스에 대한 불필요한 모니터링을 회피하고, 시스템 콜을 호출한 컨테이너 프로세스가 어떤 컨테이너에 속해 있는지를 확인할 수 있다[16].

Fig. 9에서 k(ret)probe를 통해 식별한 프로세스의 정보와 시스템 콜에 해당하는 유형 정보를 더함으로써 컨테이너 프로세스의 활동을 더 자세히 관찰할 수 있다. 본 논문에선 파일 조작, 네트워크 조작, 시그널 처리 등 총 9가지의 기준을 통해 64개의 시스템 콜을 분류하였다.

Fig. 10과 같이 각 시스템 콜의 유형을 분류하고, 이를 통해 컨테이너의 활동을 세밀하게 모니터링할 수 있다. 이러한 분류를 통해 컨테이너 내부의 활동을 더욱 세밀하게 모니터링할 수 있으며, 이를 통해 컨테이너 운영 환경의 보안과 효율성을 높일 수 있다.

4.3 컨테이너와 시스템 메타데이터 연계

Fig. 11에서 볼 수 있듯이 앞서 컨테이너 런타임 인터페이스를 통해 수집한 컨테이너 메타데이터와 eBPF 프로그램을 활용하여 수집한 시스템 메타데이터를 연계하여 어떠한 컨테이너 프로세스가 어떠한 활동을 하고 있는지를 특정한다. 이때 개별 컨테이너에 해당하는 PID와 MNT 네임스페이스 정보를 확인하기 위해 컨테이너 메타데이터 수집 시 획득하였던 개별 컨테이너의 Init 프로세스 PID 정보를 이용한다. 이를 통해 시스템 콜 모니터링 과정에서 수집한 PID의 값을 사전에 수집하였던 컨테이너의 PID와 비교함으로써 일반 호스트 프로세스와 컨테이너 내부 프로세스를 구분하여 불필요한 모니터링을 회피할 수 있다[17]. 또한, 버퍼 오버플로우와 데이터 손실 가능성을 최소화하기 위해 링 버퍼를 사용하여 사용자 공간으로 식별한 데이터를 전달한다.

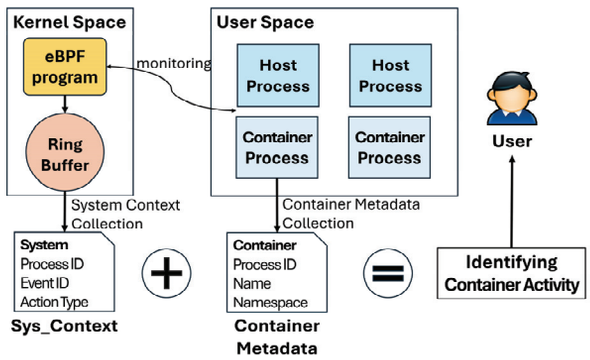

Fig. 11. Integration of Container and System Metadata

최종적으로 컨테이너 메타데이터와 시스템 메타데이터를 종합하여 호스트 커널 내부에서 발생한 개별 컨테이너의 내부 활동을 모두 감시한다.

5. 구 현

본 장에서는 제안된 eBPF 기반 컨테이너 활동 감시 시스템의 구현 과정을 상세히 기술한다. 구현 과정은 컨테이너 메타데이터 수집, 컨테이너 내부 활동 모니터링, 컨테이너와 시스템 메타데이터 연계의 세 가지 주요 단계로 나누어 진행된다.

5.1 컨테이너 메타데이터 수집

컨테이너 메타데이터를 수집하기 위해서는 호스트 내에서 Kubernetes[18], Docker[19]와 같은 컨테이너 오케스트레이션 도구 및 런타임 환경을 식별해야 한다. 각 런타임과 오케스트레이션 도구는 제공하는 리소스와 메타데이터가 다르기 때문에, 이를 효과적으로 수집하기 위해 각 시스템의 API와 통신이 필요하다. 호스트의 환경을 식별했다면 관련 API를 활용하여 현재 호스트에서 실행 중인 모든 컨테이너의 메타데이터를 수집한다. 이러한 메타데이터에는 Namespace, Pod 등이 포함되며, 이는 컨테이너의 상태를 실시간으로 추적하고 보안 및 성능 모니터링을 위한 기초 데이터를 제공하는 데 사용된다.

동시에 컨테이너의 Init 프로세스 PID 정보도 수집한다. 이는 “/proc/⟨pid⟩/ns/pid” 와 “/proc/⟨pid⟩/ns/mnt” 파일을 사용하여 프로세스의 네임스페이스 정보를 확인할 수 있다. “/proc/⟨pid⟩/ns/pid”는 해당 프로세스가 속한 PID 네임스페이스를 가리키며, 이를 통해 동일한 PID를 가지더라도 서로 다른 네임스페이스에 속하는 프로세스를 구분할 수 있다. 마찬가지로, “/proc/⟨pid⟩/ns/mnt”는 해당 프로세스가 속한 마운트 네임스페이스를 가리키며, 이를 통해 파일 시스템 마운트 지점의 격리를 확인할 수 있다. 수집된 메타데이터는 eBPF Map에 저장된다. 이는 PID와 MNT 네임스페이스 정보를 포함하며, 이후 발생하는 시스템 콜 이벤트를 모니터링할 때 참조된다. 이를 통해 어떤 프로세스가 컨테이너에 속해 있는지를 정확하게 식별할 수 있다.

5.2 컨테이너 내부 활동 모니터링 구현

해당 단계에서는 eBPF 프로그램을 작성하여 호스트 커널 영역에 kprobe와 kretprobe 훅을 삽입한다. 시스템 콜 호출로 인한 프로그램 동작 시 기본적으로 이벤트 발생 시각, PID, UID 등을 수집하며, 이후 시스템 콜 타입에 따라 추가적인 정보를 식별한다. 예를 들어, 파일 접근과 관련된 이벤트 발생 시 디렉터리 이름과 파일 이름을 추가로 수집하며 네트워크 관련 이벤트 발생 시 목적지 IP와 포트 번호를 수집한다. 이벤트가 발생할 때, 5.1 단계에서 eBPF Map에 저장된 PID와 MNT 네임스페이스 정보를 참조하여 해당 이벤트가 모니터링 대상인 컨테이너 프로세스와 관련이 있는지를 확인한다. 이를 통해 관련 데이터 수집을 지속할지 여부를 결정한다. 만약 해당 이벤트가 컨테이너 프로세스와 관련이 있다면, 추가적인 정보를 수집하여 분석에 활용한다. 수집된 데이터는 버퍼 오버플로우와 데이터 손실을 최소화하기 위해 링버퍼를 사용하여 사용자 공간으로 전송된다. 이후 bpf2go[20]를 사용하여 eBPF 프로그램을 변환하고, Go 언어로 작성된 사용자 공간 프로그램과 통합한다.

5.3 컨테이너 메타데이터 수집

해당 단계에서는 앞서 수집한 컨테이너 메타데이터와 eBPF 프로그램을 통해 수집한 시스템 메타데이터를 통합한다. 링버퍼를 사용하여 사용자 공간으로 전송된 데이터를 기반으로, 컨테이너의 메타데이터와 시스템 콜 이벤트를 연계한다. 이를 통해 컨테이너의 특정 활동을 실시간으로 모니터링하고 분석할 수 있다. 최종적으로, 이 정보를 활용하여 컨테이너 운영 환경의 보안을 강화하고 효율성을 높일 수 있다.

6. 평 가

eBPF 기반 컨테이너 활동 감시 시스템의 성능과 효과를 평가하기 위하여, sys_openat 시스템 콜을 통한 파일 읽기 및 sys_execve 시스템 콜을 통한 프로세스 생성 식별에 관한 실험을 진행하였다. 실험 환경은 Table 2와 같다.

6.1 파일 읽기

해당 실험은 sys_openat 시스템 콜을 사용하여 파일을 읽는 과정을 모니터링하는 것이다. sys_openat 시스템 콜은 sys_open과 유사하지만, 더 유연하고 안전한 파일 열기 방법

Table 2. Experimental Environment

Type	Details
CPU	Intel(R) Xeon(R) Gold 5220R CPU @ 2.20GHz (4C 4T)
RAM	64GB
OS/SW	Ubuntu 5.15.0-117-generic Docker 24.0.7 Kubernetes Client Version v1.23.0

```
ContainerName: k8s_ubunut-container1,  Namespace: default
PidId: 4026532995,  MntId: 4026532993
HostPpid: 73262,  HostPid: 73222,  Pid: 31,  Uid: 0
Syscall: sys_openat, EventType: File Operation
Activity: File Open,  Dirname: /mnt,  Filename: test.txt
```

Fig. 12. File Open Using sys_openat

```
ContainerName: k8s_ubunut-container2,  Namespace: default
PidId: 4026302895,  MntId: 4026558143
HostPpid: 73842,  HostPid: 73694,  Pid: 32,  Uid: 0
Syscall: sys_execve, EventType: Process Manage
Activity: Process Create,  Comm: bash,  Execfile: /usr/bin/cat
```

Fig. 13. Process Create Using sys_execve

을 제공하므로 Ubuntu와 같은 최신 리눅스 배포판에서는 더 선호된다.

6.2 프로세스 생성

해당 실험은 sys_execve 시스템 콜을 사용하여 새로운 프로세스를 실행하는 과정을 모니터링하는 것이다. 해당 시스템 콜은 새로운 프로그램을 실행하기 위해 현재 프로세스를 대체하는 시스템 콜로 다양한 어플리케이션 실행과 스크립트 구동에 필수적이다.

6.3 평가 결과

(Fig. 12)와 (Fig. 13)에서 확인할 수 있듯이 두 실험 모두에서 eBPF 기반 컨테이너 활동 감시 시스템은 높은 정확도를 보여주었으며, 컨테이너 운영 환경의 모니터링에 매우 효과적임을 입증하였다. 이를 통해 제안된 시스템이 컨테이너의 내부 활동을 실시간으로 모니터링하고, 보안 위협을 신속하게 감지하며, 운영 효율성을 높일 수 있음을 확인할 수 있었다.

7. 관련 연구

eBPF를 활용하여 컨테이너 활동을 모니터링하는 연구들이 진행되었다. 이러한 연구들은 eBPF의 성능 모니터링, 네트워크 트래픽 분석 등 다양한 적용 가능성을 탐구한다. 해당 장에서는 주요 연구들을 소개하고 본 연구와 비교를 한다.

Liu et al. [21]은 eBPF를 사용하여 성능 모니터링을 구현한 연구를 발표하였다. 이 연구에서는 eBPF의 경량성과 실시간 데이터 수집 능력을 활용하여 시스템 성능을 모니터링하는 방법을 제시하고, 이를 통해 성능 저하 없이 상세한 성능 데이터를 수집할 수 있음을 입증하였다. 특히, eBPF를 사용하면 프로세스 실행, 네트워크 트래픽 등 다양한 커널 레벨의 정보를 실시간으로 수집하고 분석할 수 있어, 성능 문제를 신속하게 감지하고 해결할 수 있다. Liu et al.의 연구와 유사하게, 성능 저하 없이 데이터를 수집할 수 있음을 확인하였으나 본 연구는 컨테이너 프로세스와 호스트 프로세스를 구분하여 불필요한 모니터링을 줄이는 데 초점을 맞추었다는 점에서 차이가 있다.

opportunistic_monitoring[22]는 eBPF를 이용한 프로토콜 독립적인 컨테이너 네트워크 관측 분석 시스템을 제안하였다. 이 시스템은 네트워크 인터페이스에서 발생하는 모든 트래픽을 캡처하고 분석하여, 다양한 네트워크 활동을 실시간으로 모니터링하고 이상 징후를 감지할 수 있다. 이 연구는 eBPF의 네트워크 패킷 필터링 기능을 활용하여 효율적인 네트워크 모니터링을 구현한 사례로 주목받고 있다. 반면 본 연구는 파일 접근 등의 다양한 시스템 콜 이벤트를 모니터링하여 컨테이너의 보안과 성능을 종합적으로 관리할 수 있는 시스템을 제안하였다.

Soldani and Jafarizadeh (2023)[23]은 eBPF의 클라우드 네이티브 네트워킹, 보안 및 관찰에 대한 새로운 접근법을 제안하였다. 이 논문에서는 eBPF가 어떻게 5G 및 향후 6G 네트워크에서 클라우드 네이티브 애플리케이션의 에너지 소비를 추정하고, 성능 수준을 도출하며, 비인가된 접근을 실시간으로 감지하고 대응할 수 있는지에 대해 설명한다. 본 논문 또한 이러한 목표를 공유하며, eBPF를 활용하여 컨테이너 내부 활동을 실시간으로 모니터링하고 분석하는 시스템을 제안함으로써, 네트워크뿐만 아니라 시스템 전반에 걸친 보안과 성능을 종합적으로 관리하는 데 중점을 두었다.

8. 결 론

본 연구에서는 eBPF를 활용하여 호스트 커널 영역에서 발생하는 컨테이너 내부 프로세스의 활동을 감시하는 시스템을 제안하였다. 제안된 시스템은 컨테이너 메타데이터 수집, 컨테이너 내부 활동 모니터링, 컨테이너와 시스템 메타데이터 연계의 3단계를 통해 구성되며, 이를 통해 실시간으로 컨테이너 내부 활동을 효율적으로 분석하고 관리할 수 있다. 특히, PID와 MNT 네임스페이스 정보를 활용한 프로세스 식별 방식을 통해 일반 프로세스와 컨테이너 프로세스를 구분하여 불필요한 모니터링을 줄이고 시스템 콜에 따른 프로세스 활동을 식별하여 분석할 수 있었다. 기존의 Tetragon, Falco, Tracee와 같은 대표적인 eBPF 기반 감시 시스템들을 이벤트 탐지 방식, 기본 정책 적용 여부, 이벤트 유형 식별, 시스템 콜 차단 및 경고 생성 기능의 4가지 관점에서 비교 분석함으로써, 제안된 시스템의 필요성과 타당성을 뒷받침하였다. 실험 결과, 제안된 시스템은 호스트 커널 영역에서 발생하는 이벤트를 실시간으로 모니터링하고, 수집된 데이터를 기반으로 컨테이너 활동을 분석함으로써 컨테이너 환경에서 발생할 수 있는 문제를 조기에 발견하고 대응할 수 있는 효용성을 입증하였다. 본 논문은 클라우드 환경에서의 컨테이너 관리와 모니터링을 향상시키는 데 이바지할 것이며, 이를 통해 클라우드 네이티브 애플리케이션의 안정성과 보안을 강화하는 데 중요한 역할을 할 것으로 기대된다. 향후 연구에서는 제안된 시스템의 성능 최적화와 더욱 깊이 있는 분석 방식을 구현하여 컨테이너 운영의 효율성을 한층 더 높일 것이다.

References

[1] V. Singh and S. K. Peddoju, “Container-based microservice architecture for cloud applications,” *International Conference on Computing, Communication and Automation*, IEEE, 2017.

[2] S. Sultan, I. Ahmad, and T. Dimitriou, “Container Security: Issues, Challenges, and the Road Ahead,” *IEEE Access*, Vol.7, pp.52976-52996, 2019.

[3] eBPF [Internet], https://ebpf.io/

[4] D. Soldani et al., “eBPF: A new approach to cloud-native observability, networking and security for current (5G) and future mobile networks (6G and Beyond),” *IEEE Access*, Vol.11, pp.57174-57202, 2023.

[5] M. A. Babar and B. Ramsey, “Understanding container isolation mechanisms for building security-sensitive private cloud,” Technical Report. CREST - Centre for Research on Engineering Software Technologies, University of Adelaide, Australia, 2017.

[6] N. Hedam, “eBPF - From a Programmer’s Perspective,” EasyChair Preprint no. 5198 (March 23, 2021).

[7] OSSEC - Open Source HIDS - FIM, Rootkit Detection, Malware Detection [Internet], https://www.ossec.net/

[8] Symantec Endpoint Protection [Internet], https://sep.securitycloud.symantec.com/v2/landing/

[9] M. Souppaya, J. Morello, and K. Scarfone, “Application container security guide,” Special Publication (NIST SP). National Institute of Standards and Technology, Gaithersburg, MD, 2017.

[10] Tetragon [Internet], https://tetragon.io/

[11] Falco [Internet], https://falco.org/

[12] Libsinsp [Internet], https://github.com/falcosecurity/libs/

[13] Tracee by Aqua Security [Internet], https://aquasecurity.github.io/tracee/

[14] F. J. ertinatto, D. Arioza, J. C. Nobre, L. Z. Granville, “Container-level auditing in container orchestrators with eBPF,” AINA, 2024.

[15] Isovalent, “Detecting a Container Escape with Cilium and eBPF,” Last modified July 25, 2024.

[16] Devi Priya V. S, Sibi Chakkaravarthy Sethuraman, and Muhammad Khurram Khan, “Container Security: Precaution Levels, Mitigation Strategies, and Performance Evaluation.,” Computers & Security, vol. 135, 2023.

[17] Yang, Seungyong, Brent Byunghoon Kang, and Jaehyun Nam, "Optimus: Association-Based Dynamic System Call Filtering for Container Attack Surface Reduction.", Journal of Cloud Computing, 2024.

[18] kubernetes [Internet], https://kubernetes.io/

[19] docker [Internet], https://www.docker.com/

[20] bpf2go [Internet], https://github.com/cilium/ebpf/tree/main/cmd/bpf2go

[21] C. Liu, Z. Cai, B. Wang, Z. Tang, and J. Liu, “A protocol-independent container network observability analysis system based on eBPF,” IEEE, 2020.

[22] S. Magnani, F. Risso, and D. Siracusa, “A control plane enabling automated and fully adaptive network traffic monitoring with eBPF,” IEEE, 2022.

[23] D. Soldani, and S. Jafarizadeh, “eBPF: A new approach to cloud-native observability, networking, and security for current 5G and future mobile networks (6G and Beyond),” IEEE, 2023.

김 지 수

https://orcid.org/0009-0006-9287-7775
e-mail : imjs0807@dankook.ac.kr
2024년 단국대학교 컴퓨터공학과(학사)
2024년 ~ 현 재 단국대학교 인공지능융합학과 석사과정
관심분야 : 클라우드 컴퓨팅, 시스템 보안, 네트워크 보안

남 재 현

https://orcid.org/0000-0001-8907-5495
e-mail : namjh@dankook.ac.kr
2013년 서강대학교 컴퓨터공학과(학사)
2020년 한국과학기술원 정보보호대학원(석 · 박사)
2020년 ~ 2022년 AccuKnox 수석연구원
2022년 ~ 현 재 단국대학교 컴퓨터공학과 조교수
관심분야 : 클라우드 컴퓨팅, 시스템 보안, 네트워크 보안