Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.36, NO.1, Feb. 2026 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2026.36.1.37

KubeSmith: 클라우드 네이티브 환경의 파일 접근 제어를 강화하기 위한 프레임워크*

조 치 현,[1†] 박 현 준,[2] 이 승 수[3‡]
[1,2,3]인천대학교 (대학원생, 학생, 교수)

KubeSmith: A Framework for Hardening File Access Control in Cloud-Native Environments*

Chihyeon Cho,[1†] Hyeonjun Park,[2] Seungsoo Lee[3‡]
[1,2,3]Incheon National University (Graduate, Undergraduate student, Professor)

요 약

쿠버네티스 환경에서 컨테이너 기반 워크로드는 KubeArmor, Tetragon과 같은 클라우드 네이티브 런타임 보안 도구를 통해 보호된다. 그러나 이들의 파일 접근 제어는 컨테이너별 비일관적인 프로그램 경로, 경로 무결성 미보장, 심볼릭 링크 단위 접근 제어 불가 등으로 인해 공격자에 의해 무력화될 수 있다. 본 논문에서는 이러한 한계를 해결하기 위해 KubeSmith를 제안한다. KubeSmith는 (1) 파드 분석과 대규모 언어 모델(LLM)을 활용한 정책 경로 검증 및 보정, (2) 정책 우회 패턴 식별과 완화를 위한 규칙 추가, (3) LSM-BPF 기반 개별 심볼릭 링크 접근 제어 지원을 통해 파일 접근 제어를 강화한다. 실험 결과, KubeSmith는 잘못된 경로와 정책 우회 문제를 효과적으로 개선하며, 개별 심볼릭 링크 단위 접근 제어 적용 시에도 기존 파일 접근 제어 대비 약 6.23%의 오버헤드만 발생하여, 실용적 성능 수준에서 안전한 파일 접근 제어를 제공함을 확인하였다.

ABSTRACT

In Kubernetes environments, containerized workloads are protected by cloud-native runtime security tools such as KubeArmor and Tetragon, which enforce file access control policies. However, these controls often fail to operate correctly or can be bypassed by attackers due to inconsistent program paths across containers, lack of path integrity guarantees, and the inability to enforce access control at the granularity of individual symbolic links. To address these limitations, we present KubeSmith, a framework that strengthens file access control. KubeSmith enhances security by (1) validating and correcting policy paths using pod analysis and large language models (LLMs), (2) identifying and mitigating policy bypass patterns through rule augmentation, and (3) enforcing access control at the level of individual symbolic links via LSM-BPF programs. Experimental evaluation shows that KubeSmith effectively mitigates incorrect paths and policy bypasses, while introducing only a 6.23% overhead compared to standard file-based access control, demonstrating its practicality for secure containerized deployments.

Keywords: Container Security, Filesystem Access Control, Security Policy

Received(09. 29. 2025), Modified(12. 01. 2025), Accepted(12. 13. 2025)
* 본 논문은 2025년도 한국정보보호학회 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임.
* 이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원-학·석사연계ICT핵심인재양성 지원을 받아 수행된 연구임(IITP-2026-RS-2024-00437024).
† 주저자, gsm07231@inu.ac.kr
‡ 교신저자, seungsoo@inu.ac.kr(Corresponding author)

I. 서 론

쿠버네티스[1]는 컨테이너화된 워크로드를 관리하기 위한 오픈 소스 컨테이너 오케스트레이션 플랫폼으로, 클라우드 네이티브 환경에서 사실상 표준 인프라로 자리 잡고 있다. 이러한 환경에서는 수많은 컨테이너가 동시에 동작하며, 이들을 런타임 동안 보호하기 위해 KubeAmor[2], Tetragon[3] 등과 같은 클라우드 네이티브 런타임 보안 프레임워크들이 개발되어 왔다. 이들은 컨테이너의 capabilities 변경, 파일 접근과 같은 시스템 접근 행위를 사용자 정의 정책을 기반으로 제어한다.

그러나 본 연구에서는 이들의 파일 접근 제어 기능이 다음과 같은 요인으로 인해 무력화되는 사례를 식별하였다: (1) 컨테이너별 비일관적인 프로그램 경로 (2) 경로 무결성 미보장으로 인한 정책 우회 (3) 심볼릭 링크 개별 접근 제어 불가. 이러한 한계로 인해 관리자가 정의한 정책이 올바르게 작동하지 않거나 공격자에 의해 우회될 수 있다.

여러 선행 연구[19-22]가 컨테이너에 대한 파일 접근 제어 강화 기술을 제시해 왔지만, 이들은 모두 동적 분석을 기반으로 보안 정책을 자동 생성하는 방식에 초점을 맞추었다. 하지만, 동적 분석 기술은 테스트 품질에 의해 오탐(false positivies)이 크게 높아질 수 있어, 실제 산업 환경에서 널리 채택되지 못하는 근본적인 한계를 가진다.

이에, 본 논문에서는 보안 정책 검증 및 강화와 확장된 접근 제어 메커니즘을 활용하여 기존 파일 접근 제어를 강화하는 KubeSmith 프레임워크를 제안한다. 제안하는 프레임워크는 파드 분석 및 LLM을 활용하여 보안 정책 자체의 잘못된 경로를 보정하고, 정책에서 우회 패턴을 식별하고 이를 완화하기 위한 규칙을 정책에 추가함으로써 우회 가능성을 완화한다. 또한, LSM-BPF 프로그램을 자체적으로 실행하여 개별 심볼릭 링크의 접근 제어를 지원한다.

우리는 KubeSmith 프레임워크의 프로토타입을 구현하고 실험을 수행하여, 기존 파일 접근 제어 정책의 잘못된 경로와 우회 가능성 문제를 효과적으로 개선하는 것을 확인하였다. 또한 개별 심볼릭 링크 단위의 접근 제어 메커니즘을 적용하였을 때 기존의 파일 기반 접근 제어 대비 약 6.23%의 추가 오버헤드만 발생함을 실험적으로 확인하여, 제안 기법이 실용적인 성능 수준에서 효율적으로 동작함을 검증하였다.

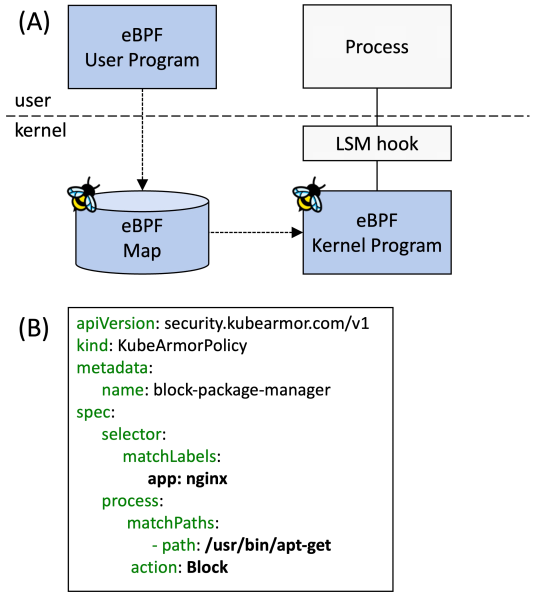

Fig. 1. (A) Architecutre of LSM-BPF, (B) A KubeArmorPolicy example

본 논문의 기여 사항은 다음과 같다:

- 클라우드 네이티브 런타임 보안 프레임워크의 파일 접근 제어 방식이 갖는 세 가지 한계를 제시하고 이를 완화하는 KubeSmith를 제안한다.
- KubeSmith의 프로토타입을 구현하고 실험적으로 평가하여, 파일 접근 제어를 효과적이고 효율적으로 강화할 수 있음을 입증한다.

II. 배경 지식

2.1 LSM-BPF

Linux Security Module(LSM)[4]은 리눅스 커널이 보안 기능을 확장할 수 있도록 제공하는 표준 인터페이스이다. LSM은 프로세스, 파일 등 커널이 관리하는 주요 객체에 대한 접근 요청을 가로채기 위해 다양한 보안 훅을 정의하며, 각 훅에는 보안 강화를 위한 모듈이 부착될 수 있다.

LSM-BPF[5]은 LSM 보안 훅에 eBPF 프로그램을 동적으로 연결할 수 있는 기반을 제공한다. 예를 들어 파일 열기 관련 LSM 보안 훅에 eBPF 프로그램을 연결하는 경우, 특정 프로세스가 민감한 파일에 접근을 시도할 때 이를 탐지하거나 차단할 수

있다. Fig 1(A)는 기본적인 LSM-BPF 프로그램의 구조를 보여준다. LSM-BPF 프로그램은 유저 영역과 커널 영역에서 각각 하나씩 실행된다. 유저 영역 프로그램은 커널 영역 프로그램을 로드하고, 지속적으로 커널 영역과 상호작용하는 인터페이스 역할을 수행한다. 커널 영역 프로그램은 실제 접근 요청에 대한 검증과 제어를 담당한다. 특히, eBPF 맵[6]은 유저 영역과 커널 영역 사이에서 데이터를 전달하는 매개체로 활용되며, 접근 제어를 위한 eBPF 프로그램에서는 주로 정책을 저장하고 참조하기 위한 저장소로 사용된다.

2.2 클라우드 네이티브 런타임 보안 프레임워크

클라우드 네이티브 런타임 보안 프레임워크는 관리자가 정의한 정책을 활용하여 런타임에 발생할 수 있는 보안 위협에 대응한다. KubeArmor와 Tetragon은 Cloud Native Computing Foundation[7]에서 관리하는 대표적인 [1] *시스템 보안 도구*이다. 이들은 LSM-BPF을 활용해 커널에 대한 컨테이너의 악의적인 요청을 탐지하거나 차단하는 기능을 제공한다. Fig 1(B)는 이들 중 KubeArmor가 제공하는 프로세스 실행 차단 정책을 나타낸다[8]. MatchLabels는 해당 정책의 집행 대상인 쿠버네티스 리소스를 나타내는 필드이며, 'app: nginx' 레이블을 가진 모든 쿠버네티스 워크로드 리소스에 대해 정책이 집행된다. Process 필드는 해당 정책이 프로세스 실행을 제어하는 정책임을 나타내며, path 필드는 제어할 프로세스의 경로를 나타낸다. 특히, *시스템 보안 도구*가 정책을 정확히 집행하기 위해서는 해당 경로가 심볼릭 링크가 가리키는 최종 대상 경로로 작성되어야 한다[9]. 마지막으로, Action 필드는 프로세스 실행을 차단하는 정책임을 나타낸다.

III. 관련 연구

컨테이너 보안 정책 생성. Mattetti[19] 등은 도커 컨테이너의 파일 접근 제어와 마운트 제어를 위한 보안 정책을 자동 생성하는 LicShield를 제안하였다. Loukidis-Andreou[20] 등은 컨테이너의 네트워크 접근 제어와 리눅스 capabilities를 제어하는 Docker-Sec을 제안했다. Zhu[21] 등은 컨테이너에서 발생한 시스템 로그를 AppArmor 정책으로 변환하는 시스템인 Lic-Sec을 제안하였으며, 이는 위의 두 연구의 한계를 개선한 통합 프레임워크이다. 또한, 이들은 이를 쿠버네티스 환경에 적합하도록 확장한 Kub-Sec[22] 또한 제안하였다.

이처럼 여러 연구들이 컨테이너의 보안을 강화하기 위해 다양한 보안 정책 자동 생성 기술을 탐구해 왔다. 그러나 동적 분석을 기반으로 생성된 정책은 오탐 가능성이 높아 실제 산업 환경에서는 널리 활용되지 못하고 있다. 이러한 한계 속에서, 관리자가 직접 생성한 보안 정책을 보강하는 KubeSmith는 보다 현실적인 대안으로 작용할 수 있다.

보안 정책 검증 및 수정. Gu[23] 등은 쿠버네티스 파드에 적용된 RBAC 정책이 실제 필요한 권한을 초과하여 부여하는지를 검증하는 EPScan을 제안하였다. 또한 일부 연구들[24-28]은 애플리케이션의 설명과 그들의 API 실제 사용을 분석하여 모바일 앱이 불필요하게 요청하는 권한을 식별하였다.

이처럼 보안 정책이 올바르게 집행되도록 실제 환경과 비교하여 검증 및 수정하는 연구는 꾸준히 이어져 왔다. 본 연구에서 제안하는 KubeSmith는 특히 파일 접근 제어 정책을 대상으로 부정확한 경로 지정 및 우회 가능성을 검증하고 보완하는 기능을 제공한다.

IV. 문제 정의

4.1 위협 모델 및 공격자 능력

우리는 컨테이너 내에서 실행 중인 애플리케이션의 취약점을 악용하여 원격 코드 실행을 수행할 수 있는 공격자를 가정한다. 공격 대상 파드는 *시스템 보안 도구*로부터 보호받으며, 파일 접근 제어 정책이 적용된 상태라고 가정한다. 주목할 점은, 공격자가 장악한 파드에 대한 특권 부여 여부가 본 논문에서 제시한 접근 제어 무력화에 영향을 미치지 않는다는 것이다. 다만 공격자가 장악한 파드가 특권을 보유한 경우, 접근 제어가 무력화된 이후 공격자가 시스템 전체에 미치는 잠재적 영향이 현저히 커질 수 있다.

4.2 컨테이너별 비일관적인 프로그램 경로

*시스템 보안 도구*는 파일 접근 제어 정책을 정의할 때 경로를 식별자로 활용한다. 이는 컨테이너의

1) 본 논문에서는 이후의 서술에서 간결함을 위해 클라우드 네이티브 런타임 보안 프레임워크를 '시스템 보안 도구'로 지칭한다.

파일 접근 요청 시, 해당 파일의 경로를 정책에 명시된 경로와 비교하여 접근 허용 여부를 결정한다는 의미이다. 하지만, 독립적인 루트 파일시스템을 가지는 컨테이너 환경에서 이러한 식별자를 정확하게 작성하는 것은 복잡한 작업이다.

이러한 복잡성은 두 가지 요인에서 비롯된다. 첫째, 동일한 프로그램이라도 컨테이너 이미지에 따라 경로가 다른 경우이다. Table 1에서 볼 수 있듯이, wget 프로그램은 Ubuntu 기반 컨테이너에서 "/usr/bin/wget"에 위치하지만 Busybox 기반의 컨테이너에서는 "/bin/wget"에 위치한다. 둘째, 동일한 유형의 프로그램이라도 이미지에 따라 프로그램명이 달라지는 경우이다. 예를 들어, 고수준 패키지 관리자 프로그램은 Ubuntu와 CentOS 기반의 컨테이너에서 "/usr/bin/apt" "/usr/bin/yum"으로 각각 상이하다.

관리자는 컨테이너마다 상이한 파일 경로로 인해 정책을 정의할 때마다 실제 경로를 확인해야 한다. 이러한 과정은 특히 다수의 컨테이너가 동시에 운영되는 쿠버네티스 환경에서 관리자의 정책 관리 부담을 가중시키며, 결과적으로 정책 오구성(misconfiguration)의 가능성을 높인다.

4.3 경로 무결성 미보장으로 인한 정책 우회

본 연구에서는 실험을 통해 *시스템 보안 도구*의 파일 접근 제어를 우회하는 새로운 공격 기법들을 식별하였다. 구체적으로 프로세스 실행 차단 정책이 우회가능하며, 식별된 공격 기법들은 파일 경로를 접근 제어의 식별자로 사용하면서도 경로 무결성을 보장하지 않는 설계적 취약성을 악용한다. 우리는 이러한 취약성을 악용하는 세 가지 기법 (1) 경로 변경, (2) 프로그램 복제, (3) 하드링크 생성을 확인하였으며, 이들 기법은 경로 기반 식별자에 의존하는 보안 정책을 무력화할 수 있다.

Fig 2는 컨테이너를 장악한 공격자가 프로그램의 경로를 변경함으로써 프로그램 실행 차단 정책을 우회하는 공격 사례를 보여준다. 해당 컨테이너에 대한 루트 권한 부여 여부는 공격의 성공 여부에 영향을 미치지 않는다. 환경에는 "/usr/bin/apt" 프로그램의 실행을 차단하는 KubeArmor의 보안 정책이 적용되어 있으며, 공격자의 apt update 요청이 차단되는 것을 확인할 수 있다. 그러나 공격자가 apt 프로그램의 경로를 manipulated-apt로 수정한 후, 동일한 프로그램의 실행을 요청할 경우 이를 정상적으로 차단하지 못하는 것을 확인할 수 있다.

Table 1. Container-specific program paths interpreted by system security tools

Program Type	Ubuntu	CentOS	Busybox
Package Manager (H)	/usr/bin/apt /usr/bin/apt-get	/usr/bin/yum /usr/bin/dnf-3	-
Package Manager (L)	/usr/bin/dpkg	/usr/bin/rpm	/usr/bin/rpm
Shell	/usr/bin/bash /usr/bin/dash	/usr/bin/bash	/bin/ash /bin/sh /bin/hush
Network Downloader	/usr/bin/wget /usr/bin/curl	/usr/bin/wget /usr/bin/curl	/bin/wget
File Editor	/usr/bin/vim.basic	/usr/bin/vi	/bin/vi

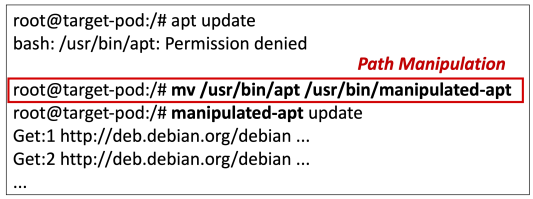

Fig. 2. Security policy bypass via path manipulation

4.4 심볼릭 링크 개별 접근 제어 불가

기존 파일 접근 제어 메커니즘은 제어 대상을 심볼릭 링크가 가리키는 최종 경로로 식별한다. 예를 들어, "/bin/apt"가 "/usr/bin/apt"를 가리키는 심볼릭 링크로 연결된 경우, "/bin/apt"의 실행 요청은 *시스템 보안 도구*에서 "/usr/bin/apt"로 해석된다. 이러한 식별 방식은 다수의 링크가 동일한 파일을 참조하는 경우, 특정 링크만을 제한하기 어려워 파일에 연결된 모든 링크에 대한 접근이 일괄적으로 차단되는 한계를 가진다. 이는 접근 제어의 정밀성을 저하시켜 실제 운영 환경에서 심각한 문제를 초래할 수 있다.

특히 Alpine 이미지 기반 컨테이너에서 이러한 문제가 두드러지는데, 이는 해당 이미지의 대부분의 바이너리가 busybox에 심볼릭 링크로 연결되어 있기 때문이다. 구체적으로, 최신 Alpine 이미지에서

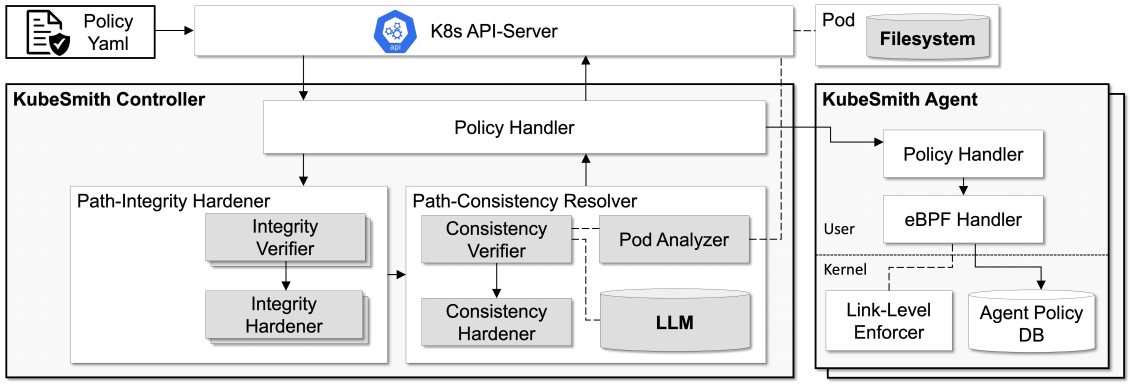

Fig. 3. Overall architecture of KubeSmith

는 총 304개의 바이너리가 busybox에 링크되어 있으며, 기존 *시스템 보안 도구*를 활용해 이들 중 하나의 실행을 차단하려면 모든 링크된 바이너리에 대한 실행을 동시에 차단해야 한다. 또한, Ubuntu 이미지에서는 단일 파일에 다수의 링크로 연결된 파일이 100개 이상, Nginx 이미지에서 500개 이상, Redis 이미지에서 200개 이상 발견되는 등 주요 컨테이너 이미지 전반에서 정책을 집행할 수 없는 동일한 문제가 확인되었다.

V. KubeSmith 설계

KubeSmith는 §4에서 제기한 파일 접근 제어 무력화 사례를 완화하기 위해 설계된 프레임워크이다. 프레임워크는 (1) 파드의 파일시스템 분석 및 LLM을 활용한 경로 보정 (2) 경로 조작 패턴 식별 및 대응 규칙 추가 (3) LSM-BPF를 활용한 추가적인 접근 제어를 기반으로 무력화 사례를 완화한다.

5.1 설계 고려 사항

R1: 비일관적인 프로그램 경로 보정. 정책의 경로가 현재 컨테이너에서 유효한지 검증해야 한다. 이를 위해 §4.2에서 언급한 두 가지 상황, 즉 (1) 동일한 프로그램이라도 컨테이너 이미지에 따라 경로가 달라지는 경우와 (2) 동일한 유형의 프로그램이라도 이미지에 따라 프로그램명이 달라지는 경우를 정확히 식별해야 한다. 나아가 이러한 식별 결과를 기반으로 정책을 보정하여, 해당 컨테이너 환경에서 유효하게 동작하는 강화된 정책으로 변환할 수 있어야 한다.

R2: 경로 조작 기반 정책 우회 방지. 정책이 공격자의 경로 조작 기법에 의해 우회될 수 있는지 검증해야 한다. 특히 §4.3에서 정의한 세 가지 기법 (1) 경로 변경, (2) 프로그램 복제, (3) 하드링크 생성에 대해 정책 우회 여부를 평가할 수 있어야 한다. 또한 검증 결과를 바탕으로 경로 조작 기법에 의한 우회가 불가능한 정책으로 보정할 수 있어야 한다.

R3: 심볼릭 링크 개별 접근 제어. 단일 파일에 연결된 다수의 심볼릭 링크를 개별적으로 접근 제어 가능해야 한다. 또한, 이러한 세분된 제어가 관리자의 추가적인 부담 없이 투명하게 운영될 수 있도록 보장해야 한다.

5.2 시스템 아키텍처 및 워크플로우

해당 섹션에서는 KubeSmith의 전체 아키텍처와 워크플로우를 설명한다. Fig 3에 나타난 바와 같이, KubeSmith는 전체 시스템의 동작을 담당하는 두 개의 주요 컴포넌트, Controller와 Agent로 구성된다. Controller는 마스터 노드에 배포되며 기존 *시스템 보안 도구*의 보안 정책을 수정하고 설치하며 심볼릭 링크 관련 제어가 필요할 경우, Agent로 관련된 규칙을 전달한다. Agent는 클러스터의 모든 노드에 배포되며, 심볼릭 링크를 제어하기 위한 정책을 집행하는 역할을 수행한다.

워크플로우는 다음과 같이 진행된다. 먼저, Policy Handler는 *시스템 보안 도구*의 정책이 생성되거나 변경되는 것을 감지하여 Path-Integrity Hardener로 전달한다. Integrity Verifier는 전달받은 정책에 대해 §5.1의 R2에 해당하는 경로 조

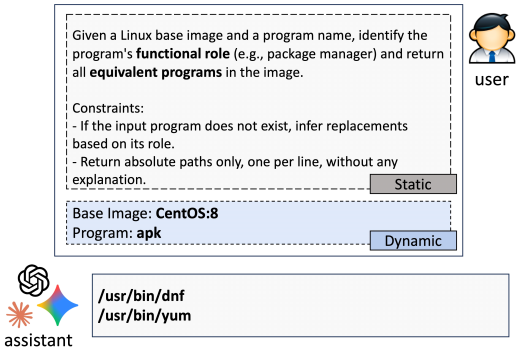

Fig. 4. Example prompt used to query programs of the same functional role

작 가능성을 검사한다. 이후 가능성이 식별된 정책은 Integrity Hardener로 전달되어 경로 조작 방지를 위한 대응 규칙이 추가된 후, Path-Consistency Resolver로 전달된다.

Consistency Verifier는 정책에 대해 §4.1의 R1에 해당하는 경로 유효성을 검사한다. 이 과정에서 Consistency Verifier는 Pod Analyzer와 LLM에 질의하고 그 결과를 검증함으로써, *시스템 보안 도구*가 올바르게 식별할 수 있는 경로를 결정한다.

동시에, §5.1의 R3를 위해 해당 경로가 다수의 심볼릭 링크를 가지는지 검사한다.

만약 경로가 다수의 심볼릭 링크를 가진다면, Consistency Verifier는 해당 경로와 관련된 규칙을 Agent로 전송하도록 Controller의 Policy Handler에 요청한다. Agent 내부의 Policy Hander는 전달받은 정책을 Agent Policy DB에 저장하며, 이후 컨테이너가 심볼릭 링크로 연결된 프로그램의 실행을 요청할 때 Link-Level Enforcer는 해당 DB를 조회하여 접근 허용 여부를 결정한다. 반대로, 경로가 다수의 링크를 가지지 않는 경우는 결정된 경로를 Consistency Hardener에게 전달한다. 이후 해당 경로를 기반으로 정책을 보정하고, 강화된 정책을 쿠버네티스 서버로 전송하도록 Policy Handler에게 요청한다.

VI. KubeSmith 시스템 세부사항

6.1 컨테이너 분석 기반 경로 유효성 검증 및 보정

Path-Consistency Resolver는 §5.1의 R1에서 언급한 경로 유효성 검증 및 보정을 위해 설계된 컴포넌트이다. 해당 컴포넌트는 컨테이너 파일시스템 정보 추출, 컨테이너-프로그램명 매칭, 파일시스템 조회 기반 경로 검증의 세 단계로 구성된다.

Table 2. Path manipulation pattern-rule mapping

Bypass Type	LSM hook	KubeArmor	Tetragon
Path Modification	path_rename	File	-
File Duplication	file_open		Hook: "file_open"
Hardlink	path_link		-

컨테이너 파일시스템 정보 추출. 정책에서 MatchLabels와 같은 적용 대상 정보를 추출하고, 이를 기반으로 해당 파드 목록을 조회한다. 이후, 파드에 포함된 컨테이너에서 Ubuntu, CentOS 등과 같은 리눅스 배포판 정보를 추출한다.

컨테이너-프로그램명 매칭. 컨테이너 이미지별로 서로 다른 동일 유형의 프로그램명을 식별하기 위해 LLM을 활용한다. 이는 LLM이 다양한 리눅스 배포판에서 사용되는 프로그램명을 학습하고 있다는 점에서 기반한다. Fig 4는 LLM 질의 시 사용되는 프롬프트를 보여주며, 크게 두 가지 영역으로 구성된다. 첫 번째는 요구 사항과 출력 형식 등 제약 조건이 명시된 Static 영역으로, 모든 질의에서 동일하게 유지된다. 두 번째는 컨테이너 파일시스템 정보와 프로그램명과 같은 데이터를 포함하는 Dynamic 영역으로, 질의마다 값이 변경된다. 해당 프롬프트를 활용한 LLM 질의를 통해, 정책 적용 대상 컨테이너 내에서 유효한 프로그램명을 정확하게 식별할 수 있다.

파일시스템 조회 기반 경로 보정. LLM이 출력한 프로그램명이 컨테이너 내에서 유효한지 검증하기 위해, 컨테이너 내부에서 해당 프로그램의 존재 여부를 확인한다. 프로그램이 존재하지 않는 경우에는 정책을 변경하지 않고, 존재하는 경우에는 정확한 정책 집행을 위해 경로가 심볼릭 링크인지 여부를 확인한다. 심볼릭 링크가 아닌 경우에는 경로 자체를 사용하고, 심볼릭 링크인 경우에는 링크가 가리키는 최종 경로를 탐색하여 정책 집행 대상 파일의 경로를 최종적으로 보정한다.

6.2 정책 분석 기반 경로 조작 가능성 검증 및 강화

Path-Integrity Hardener는 §5.1의 R2에서 언급한 경로 조작 가능성 검증 및 강화를 만족하기 위해 설계된 컴포넌트이다. 해당 컴포넌트는 경로 조작 패턴-대응 규칙 매칭, 형식 검증 기반 우회 취약 정책 식별, 규칙 보강 기반 정책 강화의 세 단계로 구성된다.

경로 조작 패턴-대응 규칙 매핑. Table 2는 공격자가 각 경로 조작 기법을 수행할 때 호출되는 LSM hook과 이에 대응하는 보안 정책의 규칙[10]을 나타낸다. KubeArmor의 경우 File 규칙을 사용하여 모든 경로 조작 기법을 방지할 수 있지만, Tetragon은 각 경로 조작 기법에 따라 다른 규칙 구성이 필요하다.

형식 검증 기반 우회 취약 정책 식별. 프로세스 실행 차단 정책의 규칙을 분석하여, 모든 차단 대상을 추출한다. 이후, 해당 대상들에 대한 경로 무결성 보장 정책이 집행되어 있는지 검증한다. 예를 들어, KubeArmor의 경우 모든 프로세스 차단 정책이 File 규칙과 함께 작성되었는지 정책의 필드를 검사하여 확인한다.

규칙 보강 기반 정책 강화. 식별된 취약한 정책에 대해, 경로 조작 패턴-대응 규칙 매핑을 기반으로 규칙을 보완한다. 이를 통해 해당 정책이 차단하는 프로그램에 대해 경로 조작을 원천적으로 차단할 수 있도록 강화한다.

6.3 에이전트 기반 심볼릭 링크 개별 제어

§5.1의 R3에서 제시한 개별 심볼릭 링크에 대한 접근 제어를 지원하기 위해, 우리는 LSM-BPF 기반의 확장 접근 제어 메커니즘을 채택하였다. 이때 추가적인 접근 제어로 인해 관리자의 정책 관리 부담이 증가하지 않도록, 대상 식별과 규칙 관리 과정을 투명하게 제공한다. 제안된 기능은 확장 접근 제어 메커니즘의 대상 식별, 규칙의 전달 및 저장, 규칙의 집행의 세 가지 요소로 구성된다.

확장 접근 제어 메커니즘의 대상 식별. 컨테이너의 파일시스템에서 다수의 심볼릭 링크를 가진 파일을 모두 파악한다. 이후 심볼릭 링크의 최종 경로를 탐색하는 과정에서, 해당 최종 경로가 다수의 심볼릭 링크를 가진 파일 목록에 존재하는지 검사한다.

규칙의 전달 및 저장. 다수의 심볼릭 링크와 연결된 파일에 대한 접근 제어 규칙은 KubeSmith Agent로 전달된다. 해당 컴포넌트는 전달받은 규칙을 확장 접근 제어를 위한 LSM-BPF 프로그램이 식별할 수 있는 형태로 변환해 eBPF Map에 저장한다.

eBPF Map은 단일 바이너리에 대한 다수의 심볼릭 링크를 관리하기 위해, 중첩 맵 유형을 선택했다. 외부 맵의 키는 다수의 심볼릭 링크를 가진 단일 바이너리 경로이고, 값은 내부 맵의 주소이다. 내부 맵의 키는 개별 심볼릭 링크 자체이며, 값은 0x01로 존재 유무를 간단히 나타낸다.

규칙의 집행. 개별 심볼릭 링크에 대한 접근 제어는 기존 *시스템 보안 도구*의 접근 제어 메커니즘 이후 보조적으로 집행된다. 구체적으로, eBPF 커널 프로그램은 “bprm_check_security” hook에 부착되어 프로세스 실행이 요청될 때마다 트리거된다[11]. 해당 커널 프로그램은 기존 *시스템 보안 도구*와 동일하게 컨테이너 식별을 위한 로직을 가진다.

개별 심볼릭 링크에 대한 접근 제어를 위해 커널 프로그램의 인자로 넘겨진 linux_binprm[12] 구조체의 file 필드와 filename 필드를 활용한다. file 필드는 심볼릭 링크의 최종 경로를 나타내고, filename 필드는 개별 심볼릭 링크의 경로를 나타낸다. 이 두 경로는 위 단계에서 저장된 eBPF Map을 탐색하는 키 값으로 활용되고, 규칙과 일치하는 심볼릭 링크에 대한 실행을 요청한 경우 그 실행이 차단된다.

VII. 구현 및 평가

7.1 실험 환경 및 구현

실험은 Proxmox VE가 설치된 Supermicro Super Server에서 수행되었다. 호스트 시스템의 하드웨어 사양은 Intel Xeon Silver 4210R 프로세서(10 physical cores, 2.40 GHz, 13.75 MB L3 캐시), 256 GB DDR4 메모리, 2 TB SSD로 구성된다. 본 환경 위에서 총 3개의 가상머신을 생성하였으며, 각 머신에 4 vCPU 코어와 8 GB 메모리를 할당했다. 또한, 가상머신에는 리눅스 커널 v6.11을 설치하였으며, 쿠버네티스 1.32.0 버전으로 세 머신을 클러스터링하였다. 클러스터에는 실험을 위해 KubeArmor 1.6.3을 설치하였다.

KubeSmith의 세부 구현 사항은 다음과 같다. 보안 정책의 감지, 수정 및 업데이트를 위해 KubeSmith 프레임워크는 client-go[13]를 기반으로 구현하였다. 컨테이너-프로그램 매칭 모듈은 구글의 Gemini API[14]를 활용해 구현하였다. 마지막으로, 확장 접근 제어 메커니즘은 cilium-ebpf[15] 패키지를 사용하여, eBPF 커널 프로그램과 사용자 공간 프로그램을 구현함으로써 구성하였다.

7.2 보안 정책 강화의 효과성

7.2.1 테스트 정책 수집

본 논문에서는 컨테이너의 비일관적인 경로로 인해 발생하는 잘못된 정책 구성 문제와 경로 무결성 미보장으로 인한 정책 우회 문제를 KubeSmith가 효과적으로 완화할 수 있는지를 평가하였다. 이를 위해 KubeSmith가 동작하지 않는 클러스터와 동작하는 클러스터에서 각각 동일한 보안 정책을 집행하고, §4에서 제시한 문제 상황이 개선되었는지 비교하였다. 평가에는 KubeArmor의 정책 저장소[16]에서 선별한 5개의 보안 정책[29-33]을 사용하여 데이터셋을 구성하였다.

7.2.2 LLM 프로그램 추정의 정확성

본 논문에서는 §6.1에서 제안한 컨테이너-프로그램명 매칭 기법의 정확성을 평가하였다. 평가를 위해 Ubuntu, CentOS, Busybox 이미지와 프로그램명을 조합하여 쿼리하였으며, 추론 결과와 grount truth로 선정한 프로그램명을 비교하였다. 프로그램명은 §7.2.1에서 선별한 5개의 보안 정책에 명시된 프로그램 중에서 Table 1의 프로그램 유형 별로 각 한 개씩 선별하였다(apt, dpkg, bash, wget, vi). Ground truth는 Table 1에서 나열한 경로를 사용하였으며, LLM이 출력한 프로그램 이름이 ground truth와 동일한 경우에만 올바른 추론으로 간주하였다.

추론 성능 평가는 기본 지표를 활용하였으며, LLM이 출력한 프로그램 이름과 ground truth를 비교하여 다음과 같이 정의하였다:

- True Positive(TP): LLM 출력이 ground truth에 존재하는 경우
- False Positive(FP): LLM 출력이 ground truth에 존재하지 않는 경우
- False Nagative(FN): Ground truth에 존재하지만 LLM이 출력하지 못한 경우

Table 3. Accuracy of KubeSmith's container-program matching

Program Type	Ubuntu		CentOS		Busybox	
	w/o	w/	w/o	w/	w/o	w/
Package Manager(H)	2/2	2/2	0/2	2/2	-	-
Package Manager(L)	1/1	1/1	0/1	1/1	0/1	0/1
Shell	1/2	2/2	1/1	1/1	1/3	2/3
Network Downloader	2/2	2/2	2/2	2/2	0/1	1/1
File Editor	0/1	1/1	0/1	1/1	1/1	1/1

Table 3은 평가 결과를 나타내며, FP가 FN에 비해 빈번하게 발생하는 것을 확인할 수 있다. FP가 발생한 대부분의 경우는 LLM이 출력한 프로그램이 컨테이너 내에 존재하지 않는 경우이다. 이는 §6.1에서 언급한 파일시스템 조회 기반 경로 보정을 통해서 정책에 포함되지 않도록 필터링할 수 있다. 일부 FP에서는 프로그램이 컨테이너 내에 존재하였는데, 이는 ground truth로 설정한 유사 프로그램의 범위와 LLM의 유사 프로그램 추론 사이의 간극으로 인해 발생한다. 추후에는 LLM의 프롬프트를 유사 프로그램에 대한 범위를 지정하는 방식으로 고도화하여 이러한 간극을 좁히는 연구를 진행할 계획이다.

7.2.3 경로 유효성 검증 및 보정의 효과성

각 환경에 배포된 정책 데이터셋이 다양한 컨테이너에 정확히 적용되는지 비교하여, 경로 유효성 검증 및 보정 기능의 효과성을 측정하였다. Table 4는 KubeSmith의 실행 여부에 따른 컨테이너 이미지별 정책 적용 결과를 프로그램 유형별로 나타낸 것이다. 예를 들어, Ubuntu 컨테이너에 File Editor를 차단하는 정책을 적용한 결과, 기본 환경에서는 잘못된 경로 구성으로 인해 1개의 프로그램 중 0개를 차단했지만 KubeSmith를 활용한 경우 컨테이너에 존재하는 프로그램을 정확하게 차단한 것을 확인할 수 있다. 또한, 파일시스템 조회 기반 경로 보정 기능의 효과를 검증하기 위해, 심볼릭 링크를 차

Table 4. Effectiveness of KubeSmith's misconfigured path detection and correction

Program Type	Ubuntu		CentOS		Busybox	
	FP	FN	FP	FN	FP	FN
Package Manager(H)	1	0	1	0	1	-
Package Manager(L)	2	0	2	0	1	1
Shell	1	0	1	0	0	1
Network Downloader	0	0	0	0	0	0
File Editor	2	0	2	0	0	0

단 대상으로 포함한 보안 정책을 Ubuntu 컨테이너에 적용하여 실험을 수행하였다. 실험에 사용한 심볼릭 링크는 "/bin" 디렉토리에 존재하는 프로그램들을 이름 기준으로 오름차순 정렬한 뒤, 상위 100개 항목을 선정하여 구성하였다. 실험 결과, KubeSmith를 적용하지 않은 경우 정책에 명시된 심볼릭 링크들이 모두 차단되지 않았으나, KubeSmith의 경로 보정 기능을 적용한 경우 정책에 포함된 모든 경로에 대한 접근이 정상적으로 차단되는 것을 확인하였다.

7.2.4 경로 조작 가능성 검증 및 강화의 효과성

각 환경에 배포된 임의의 파드에서 정책 적용 대상 프로그램에 대한 경로 조작을 시도하였다. Fig 5는 "/usr/bin/apt" 실행 차단 정책이 적용된 컨테이너에서, 공격자가 경로를 조작해 정책을 우회하는 사례를 나타낸다. Fig 5(A)는 KubeSmith가 적용되지 않은 환경으로, 공격자가 경로를 조작해 정책을 우회하는 사례를 나타낸다. 반면, Fig 5(B)에서는 KubeSmith가 적용된 환경에서 경로 조작이 성공적으로 차단되어 우회 시도가 실패하는 것을 확인할 수 있다. 하지만, KubeArmor는 모든 경로 조작 시도를 성공적으로 차단한 반면, Tetragon은 파일 복제를 제외한 나머지 경로 조작 시도를 차단하지 못하였다. 이는 Table 2에서 확인할 수 있듯이, Tetragon이 해당 경로 조작 유형을 차단할 수 있는 규칙을 현재 지원하지 않기 때문이다. 향후 Tetragon이 관련 규칙을 지원할 경우, 이는 자연스럽게 해결될 것으로 기대한다.

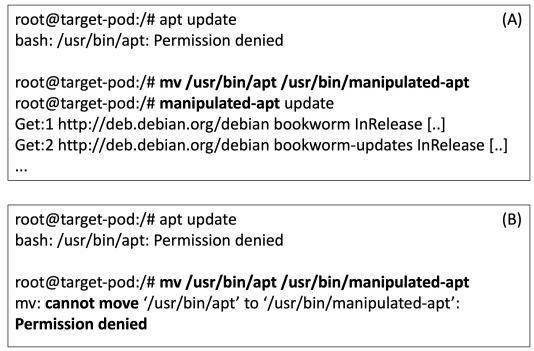

Fig. 5. Use case for preventing security policy bypass based on path manipulation

7.2.5 개별 심볼릭 링크 접근 제어의 효과성

개별 심볼릭 링크 차단 기능의 효과성을 평가하기 위해, 대부분의 실행 파일이 "/bin/busybox"와 연결된 Alpine 기반 컨테이너 환경에서 실험을 수행하였다. 실험은 도커 허브[17]에서 제공하는 3.22.1 버전의 Alpine 이미지로 생성된 컨테이너에서 진행되었으며, "/bin/busybox"에 연결된 총 304개의 링크 중 "/bin" 디렉토리 하위에 위치한 82개의 파일을 대상으로 선정하였다. 실험 결과, 기존 KubeArmor만 동작하는 환경에서는 어떠한 개별 심볼릭 링크도 차단할 수 없었다. 반면, KubeSmith를 함께 적용한 환경에서는 실험에 포함된 모든 대상 심볼릭 링크에 대해 접근 제어가 가능함을 확인하였다.

7.3 개별 심볼릭 링크 접근 제어의 효율성

본 논문에서는 개별 심볼릭 링크 차단 기능으로 인해 발생하는 추가적인 오버헤드를 측정하였다. 이를 위해 동일한 기능을 수행하지만 컨테이너 이미지에 따라 심볼릭 링크로 연결되어 있거나 그렇지 않은 다섯 개의 프로그램(ls, netstat, ps, mount, chown)을 선정하였다.

실험 환경은 도커 허브에서 제공하는 Ubuntu, CentOS, Alpine 기반 컨테이너를 각각 하나씩 실행하여 구성하였다. 특히, Alpine 기반 컨테이너는 KubeArmor와 KubeSmith가 동시에 실행 중인 클러스터에 배포하였고, 나머지 컨테이너는 KubeArmor만 실행 중인 클러스터에 배포하였다. 이러한 배포 방식은 KubeSmith가 적용된 경우와

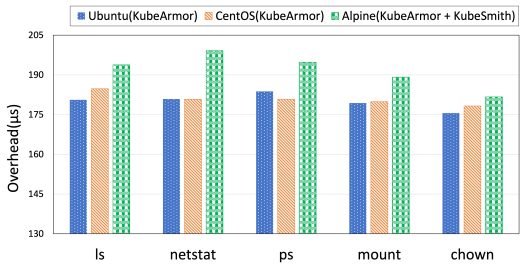

Fig. 6. Performance Comparison between KubeArmor and KubeArmor+KubeSmith

적용되지 않은 경우 간의 심볼릭 링크 단위 접근 제어 성능 차이를 비교하기 위함이다. 즉, KubeSmith가 제공하는 심볼릭 링크 단위 정책 집행 기능이 실제로 추가적인 오버헤드를 발생시키는지, 기존 접근 제어와 비교할 수 있도록 설계하였다.

모든 클러스터에는 다섯 개의 프로그램 실행을 차단하는 동일한 KubeArmor 정책을 배포하였다. 프로그램 실행 차단에 소요되는 시간은 strace[18]를 사용해 측정하였고, 동일 실험을 1,000회 반복하여 평균값을 산출하였다.

Fig 6은 실험 결과를 보여주며, 심볼릭 링크 차단 기능이 활성화된 경우 모든 프로그램에서 추가적인 오버헤드가 발생함을 확인할 수 있다. 또한, Ubuntu와 CentOS에서 측정된 차단 소요 시간의 평균과 Alpine에서 측정된 차단 소요 시간을 비교한 결과, 평균 약 6.23%의 오버헤드가 발생하였다. 그러나 해당 오버헤드는 매우 작은 수준으로, 제안 기법이 정책 집행 성능에 실질적인 영향을 미치지 않음을 확인할 수 있다.

VIII. 결 론

본 논문에서는 시스템 보안 도구의 보안 정책이 무력화될 수 있는 사례와, 심볼릭 링크를 개별적으로 제어하지 못하는 기존 보안 집행 메커니즘의 한계를 제시하였다. 이를 해결하기 위해, 보안 정책을 강화하고 기존 시스템 보안 도구의 집행 메커니즘을 확장한 프레임워크 KubeSmith를 제안하였다. 평가 결과, KubeSmith는 컨테이너 내의 유효하지 않은 경로를 효과적으로 보정하고, 공격자의 정책 우회를 차단하는 데 실질적인 효과가 있음을 입증하였다. 또한, 심볼릭 링크를 개별적으로 접근 제어할 수 있으며, 이에 따른 성능 오버헤드가 무시할 수 있을 정도로 낮음을 확인하였다.

References

[1] Kubernetes, "Kubernetes", https://kubernetes.io/, Accessed: Nov. 24, 2025.

[2] KubeArmor, "KubeArmor", https://kubearmor.io/, Accessed: Nov. 24, 2025.

[3] Tetragon, "Tetragon", https://tetragon.io/, Accessed: Nov. 24, 2025.

[4] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," in Proceedings of the 11th USENIX Security Symposium (USENIX Security 02), pp. 17 - 31, Aug. 2002.

[5] J. Her, C. H. Jo, T. Park, and S. Lee, "KUBEROSY: A Dynamic System Call Filtering Framework for Containers," IEEE Access, vol. 12, pp. 159889 - 159901, Oct. 2024.

[6] C. Liu, B. Tak, and L. Wang, "Understanding Performance of eBPF Maps," in Proceedings of the ACM SIGCOMM 2024 Workshop on eBPF and Kernel Extensions, pp. 9 - 15, Aug. 2024.

[7] Cloud Native Computing Foundation, "CNCF", https://cncf.io/, Accessed: Nov. 24, 2025.

[8] KubeArmor Documentation Quick Start, "KubeArmor process execution blocking policy," https://docs.kubearmor.io/kubearmor/quick-links/deployment_guide, Accessed: Nov. 24, 2025.

[9] KubeArmor Documentation FAQs, "Process Path and Symbolic Link Resolution in KubeArmor Policies," https://docs.kubearmor.io/kubearmor/documentation/faq, Accessed:

Nov. 24, 2025.
[10] LSM hook Definition, "LSM hooks for file and path operations," https://elixir.bootlin.com/linux/v6.11/source/include/linux/lsm_hook_defs.h, Accessed: Nov. 24, 2025.
[11] Linux Security Module Development, "bprm_check_security LSM hook," https://www.kernel.org/doc/html/v5.1/security/LSM.html, Accessed: Nov. 24, 2025.
[12] Linux binfmts Definition, "linux_binprm structure file and filename fields," https://elixir.bootlin.com/linux/v6.11/source/include/linux/binfmts.h, Accessed: Nov. 24, 2025.
[13] Client-go, "client-go", https://github.com/kubernetes/client-go, Accessed: Nov. 24, 2025.
[14] Google Gemini API, "google gemini api", https://ai.google.dev/gemini-api/docs, Accessed: Nov. 24, 2025.
[15] Cilium ebpf, "cilium ebpf", https://github.com/cilium/ebpf, Accessed: Nov 24, 2025.
[16] KubeArmor policy-templates, "Kube Armor security policy templates for container and system protection," https://github.com/kubearmor/policy-templates, Accessed: Nov. 24, 2025.
[17] Docker hub, "docker hub", https://hub.docker.com/, Accessed Nov. 24, 2025.
[18] Strace, "strace", https://man7.org/linux/man-pages/man1/strace.1.html, Accessed Nov. 24, 2025.
[19] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev, and L. Foschini, "Securing the Infrastructure and the Workloads of Linux Containers," in Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS), pp. 559–567, Sep. 2015.
[20] F. Loukidis-Andreou, I. Giannakopoulos, K. Doka, and N. Koziris, "Docker-sec: A Fully Automated Container Security Enhancement Mechanism," in Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1561–1564, Jul. 2018.
[21] H. Zhu and C. Gehrmann, "Lic-Sec: An Enhanced AppArmor Docker Security Profile Generator," Journal of Information Security and Applications, vol. 61, p. 102924, Aug. 2021.
[22] H. Zhu and C. Gehrmann, "Kub-Sec, an Automatic Kubernetes Cluster AppArmor Profile Generation Engine," in Proceedings of the 2022 14th International Conference on Communication Systems & NetworkS (COMSNETS), pp. 129–137, Jan. 2022.
[23] Y. Gu, X. Tan, Y. Zhang, S. Gao, and M. Yang, "EPScan: Automated Detection of Excessive RBAC Permissions in Kubernetes Applications," in Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP), pp. 3199–3217, May 2025.
[24] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permission Specification," in Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS), pp. 217–228, Oct. 2012.
[25] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," in Procee-

dings of the 18th ACM Conference on Computer and Communications Security (CCS), pp. 627–638, Oct. 2011.

[26] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards Automating Risk Assessment of Mobile Applications," in Proceedings of the 22nd USENIX Security Symposium (USENIX Security 13), pp. 527–542, Aug. 2013.

[27] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the Description-to-Permission Fidelity in Android Applications," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1354–1365, Nov. 2014.

[28] S. Zhang, H. Lei, Y. Wang, D. Li, Y. Guo, and X. Chen, "Apimind: API-Driven Assessment of Runtime Description-to-Permission Fidelity in Android Apps," in Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp. 427–438, Oct. 2023.

[29] GitHub, "ksp-block-sysrv-hello-malware," https://github.com/kubearmor/policy-templates/blob/38005efe72fc72a7bfbe806cedd18cd2b3d681d3/malware/system/ksp-block-sysrv-hello-malware.yaml#L31, Accessed: Nov. 25, 2025.

[30] GitHub, "ksp-teamtnt-tntbotinger-ddos-block," https://github.com/kubearmor/policy-templates/blob/38005efe72fc72a7bfbe806cedd18cd2b3d681d3/malware/system/ksp-teamtnt-tntbotinger-ddos-block.yaml, Accessed: Nov. 25, 2025.

[31] GitHub, "ksp-audit-stig-psql-v-214152.yaml," https://github.com/kubearmor/policy-templates/blob/38005efe72fc72a7bfbe806cedd18cd2b3d681d3/stigs/system/ksp-audit-stig-psql-v-214152.yaml#L25, Accessed: Nov. 25, 2025.

[32] GitHub, "ksp-nist-sc24-fail-in-known-state.yaml," https://github.com/kubearmor/policy-templates/blob/38005efe72fc72a7bfbe806cedd18cd2b3d681d3/nist/system/ksp-nist-sc24-fail-in-known-state.yaml#L19, Accessed: Nov. 25, 2025.

[33] GitHub, "ksp-audit-nist-7-untrusted-shell-execution-program.yaml," https://github.com/kubearmor/policy-templates/blob/38005efe72fc72a7bfbe806cedd18cd2b3d681d3/nist/system/ksp-audit-nist-7-untrusted-shell-execution-program.yaml, Accessed: Nov. 25, 2025.

〈저자소개〉

조 치 현 (Chihyeon Cho) 학생회원
2025년 2월 : 인천대학교 컴퓨터공학부 학사
2025년 3월~현재 : 인천대학교 컴퓨터공학과 석사과정
〈관심분야〉 클라우드 보안, 가상화 보안, 기밀 컴퓨팅

박 현 준 (Hyeonjun Park) 학생회원
2022년 3월~현재 : 인천대학교 컴퓨터공학부 학사과정
〈관심분야〉 클라우드 보안, 보안 정책

이 승 수 (Seungsoo Lee) 종신회원
2014년 2월 : 숭실대학교 컴퓨터학부 학사
2016년 2월 : KAIST 정보보호대학원 석사
2020년 8월 : KAIST 정보보호대학원 박사
〈관심분야〉 클라우드 보안, 네트워크 보안