Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.4, Aug. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.4.637

클라우드 네이티브 환경을 위한 오픈소스 기반 모니터링 서비스 간편 배포 및 이미지 서명 검사기 구현*

곽 송 이,[1†] 응 웬 부 렁,[2] 정 수 환[3‡]
[1,2,3]숭실대학교 (대학원생, 박사후 연구원, 교수)

Implementation of Opensource-Based Automatic Monitoring Service Deployment and Image Integrity Checkers for Cloud-Native Environment*

Songi Gwak,[1†] Long Nguyen-Vu,[2] Souhwan Jung[3‡]
[1,2,3]Soongsil University (Graduate student, Postdoctoral Researcher, Professor)

요 약

클라우드 컴퓨팅은 수십 년을 걸쳐 인기를 얻고 있으며, 그에 따라 클라우드 네이티브 애플리케이션에 주요하게 사용되는 기술인 컨테이너 또한 주목을 받고 있다. 이러한 컨테이너 기술은 기존 VM보다 가볍고 성능이 뛰어나지만, 호스트 시스템과 커널을 공유하거나 이미지 레지스트리에서 이미지를 업/다운로드 하는 등의 문제로 여러 가지 보안상의 위협이 존재한다. 컨테이너의 보안 위협 중 하나로 컨테이너 생성의 소스가 되는 컨테이너 이미지의 무결성을 언급할 수 있다. 또한, 컨테이너 애플리케이션이 동작하는 동안의 런타임 보안이 매우 중요하며, 런타임에서 컨테이너 애플리케이션의 동작을 모니터링함으로써 컨테이너에서 발생하는 이상 행위를 탐지하는 데에 도움을 줄 수 있다. 따라서 본 논문에서는 첫째로, 컨테이너 이미지의 무결성을 보장하기 위해 기존의 Docker Content Trust (DCT) 기술을 기반으로 자동으로 이미지의 서명을 검사하는 서명 검사기를 구현한다. 다음으로 Cloud Native Computing Foundation (CNCF)의 오픈소스 프로젝트인 falco를 기반으로 falco 이미지의 배포 간편성을 위해 새로 생성한 이미지를 소개하고, 간편하게 모니터링 시스템을 구축할 수 있도록 돕는 docker-compose를 구현 및 패키지 구성을 제안한다.

ABSTRACT

Cloud computing has been gaining popularity over decades, and container, a technology that is primarily used in cloud native applications, is also drawing attention. Although container technologies are lighter and more capable than conventional VMs, there are several security threats, such as sharing kernels with host systems or uploading/downloading images from the image registry. one of which can refer to the integrity of container images. In addition, runtime security while the container application is running is very important, and monitoring the behavior of the container application at runtime can help detect abnormal behavior occurring in the container. Therefore, in this paper, first, we implement a signing checker that automatically checks the signature of an image based on the existing Docker Content Trust (DCT) technology to ensure the integrity of the container image. Next, based on falco, an open source project of Cloud Native Computing Foundation (CNCF), we introduce newly created image for the convenience of existing falco image, and propose implementation of docker-compose and package configuration that easily builds a monitoring system.

Keywords: Cloud Computing, Cloud Native Security, Container Image Integrity, Container Monitoring

Received(04. 14. 2022), Modified(1st: 05. 23. 2022, 2nd: 06. 29. 2022), Accepted(07. 28. 2022)

* 본 연구는 2022년 정부(과학기술통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2020-0-00952, 5G+ 서비스 안정성보장을 위한 엣지 시큐리티 기술 개발)

† 주저자, song2@soongsil.ac.kr

‡ 교신저자, souhwanj@ssu.ac.kr(Corresponding author)

I. 서 론

클라우드 컴퓨팅은 수십 년을 걸쳐 인기를 얻어오며, 많은 프로덕션에 실제로 적용되고 있다. 클라우드 컴퓨팅은 최소한의 관리 노력으로 신속한 프로비저닝 및 구성 가능한 컴퓨팅 리소스, 편리성 등을 가능하게 하는 모델이다[1]. 이러한 클라우드 컴퓨팅을 채택하여 애플리케이션을 구축, 최적화, 모든 환경을 연결하는 작업을 신속히 할 수 있도록 하는 방식이 클라우드 네이티브 애플리케이션이다 [2]. 그러나 이러한 클라우드 네이티브 애플리케이션이 기존의 애플리케이션 형태를 구축하는 인프라 구조에서는 불가능했던 편리성, 확장성 및 가용성을 가져다주는 만큼, 많은 보안 위협 또한 존재한다.

특히, 컨테이너 기술을 주요하게 활용하는 클라우드 네이티브 애플리케이션에서는 컨테이너의 소스가 되는 컨테이너 이미지의 무결성이 중요하다. 해당 무결성을 보장하기 위해 컨테이너 이미지에 대한 서명 기능을 사용할 수 있다. 본 논문에서는 첫 번째로, Docker Content Trust (DCT)를 이용해 무결성 보장을 위한 서명 검사 기능을 자동적으로 수행하는 방법을 제시한다. 두 번째로, 클라우드 네이티브 애플리케이션에서는 애플리케이션이 동작하는 동안의 런타임 보안도 중요하다. 본 논문에서는 여러 런타임 보안 중 컨테이너의 이상 행위 탐지를 위해 CNCF (Cloud Native Computing Foundation)의 오픈소스 프로젝트인 falco를 이용해 컨테이너 애플리케이션 모니터링을 간편하게 배포하도록 하는 방법을 소개한다.

논문은 2장에서 배경지식을 설명하며, 3장에서 기존 기술을 분석한다. 4장에서는 본 논문에서 제안하는 방법을 설명한다. 다음 5장에서는 구현한 툴에 대한 평가를 수행하며, 6장의 결론을 통해 끝맺는다.

II. 배경지식

2.1 Container Technology

컨테이너 기술은 가상화 기술의 하나로, 애플리케이션의 실행을 위해 관련 라이브러리나 구성 파일 등을 하나의 패키지로 구성해 구동하는 기술이다. 이러한 컨테이너는 물리적으로 머신을 분리하는 VM과는 달리 동일한 운영 체제 커널을 공유하고 애플리케이션 프로세스를 격리함으로써 기존 VM에 비해 가볍다는 장점이 있다. Fig. 1.은 컨테이너의 구조이다[3]. 클라우드 네이티브 애플리케이션은 이러한 컨테이너 기술을 채택하여 이상적인 애플리케이션 배포 및 독립적인 실행환경을 달성할 수 있다. 그러나, 컨테이너는 호스트 시스템 및 다른 컨테이너들과 호스트 운영 체제를 공유하기 때문에 VM 보다 격리성 측면에서 보안성이 떨어지는 문제가 있다.

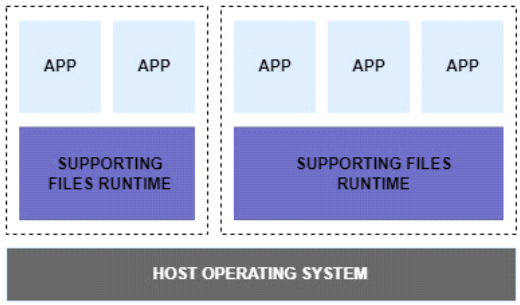

Fig. 1. Structure of Container

2.2 Container Image

컨테이너 이미지는 시스템에서 컨테이너를 만들도록 하는 실행 코드를 포함한 정적파일이다 [4]. 이 이미지는 애플리케이션을 구동하는 데 필요한 컨테이너 엔진, 시스템 라이브러리, 유틸리티, 설정 파일 및 특정 애플리케이션 코드 등 모든 정보를 포함하고 있다. 이렇게 한 번 배포된 컨테이너 이미지는 변경할 수 없으며, 어떠한 환경에서도 일관되게 배포할 수 있다.

한편, 컨테이너 이미지 및 관련된 아티팩트는 컨테이너 레지스트리에 저장 및 배포된다 [5]. 컨테이너 레지스트리의 예로는, Docker 컨테이너 이미지의 일반 카탈로그 역할을 하는 공용 컨테이너 레지스트리인 Docker Hub 외에도 Azure 및 github container registry 뿐만 아니라 private registry를 생성하여 사용할 수 있다. 안전하지 않은 채널을 통해 레지스트리에 연결하는 경우 이미지가 포함하는 내장된 시크릿이나 민감한 설정 등에 대한 기밀성이 노출될 위험이 있다 [6]. 또한, 네트워크 트래픽을 가로채고 트래픽 내의 개발자 또는 관리자의 자격 증명을 도용하거나 오케스트레이터에 가짜, 오래된 이미지를 제공하는 중간자 공격의 위험도 증가한다. 따라서, 해당 컨테이너 이미지가 변조되지 않았다는 것을 증명하기 위해 전자서명을 사용할 수 있다.

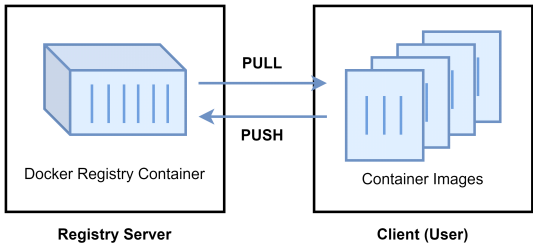

Fig. 2. Operation between Image Registry and Client

2.3 System Call

컴퓨팅에서 시스템 콜은 컴퓨터 프로그램이 실행되는 운영 체제의 커널로부터 서비스를 요청하는 프로그래밍 방식이다. 시스템 콜은 또한 프로그램이 운영 체제와 상호 작용하는 방법으로, 컴퓨터 프로그램은 운영 체제의 커널에 요청할 때 시스템 콜을 호출한다. 시스템 콜은 API (응용프로그램 인터페이스)를 통해 운영 체제의 서비스를 사용자 프로그램에 제공한다. 시스템 콜은 Fig. 3.와 같이 동작한다.

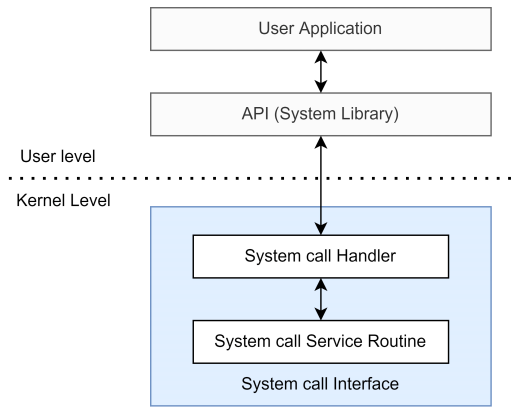

Fig. 3. Operation of System Call

III. 기존 기술

3.1 Docker Content Trust (DCT)

2.2절에서 언급한 컨테이너 이미지와 관련하여 Docker 이미지를 이용하는 경우, 이미지가 원격의 Docker 레지스트리에 저장되고 (Push), 이를 꺼내어 (Pull) 사용할 수 있다. 이러한 이미지를 공용 또는 개인 레지스트리에 저장하거나 꺼내는 경우, 네트워크 시스템 간에 데이터를 전송함으로 신뢰성 문제가 발생한다. 이를 위해 DCT를 사용하여 원격 Docker 레지스트리와 주고받는 데이터에 디지털 서명을 할 수 있다. 디지털 서명을 통해 클라이언트 또는 런타임에서 특정 이미지 태그에 대한 무결성 및 게시자를 확인할 수 있다 [7].

DCT는 특정 이미지에 대한 무결성 및 게시자를 확인하기 위해 특정 이미지의 태그를 이용한다. 개별적인 이미지 레코드에 대한 식별자는 Fig. 3.와 같은 형태이다.

DCT의 컴포넌트는 "Registry", "Notary Signer Server", "Notary Server", "Notary Database Server"로 이루어져 있다. 사용자는 Docker 클라이언트를 통해 Fig. 4.과 같이 태그가 존재하는 이미지의 서명을 확인할 수 있다 [8]. Fig. 5.에 따라, 이미지가 게시자에 의해 레지스트리에 저장되고, 게시자는 Notary Server를 활용함으로써 이미지에 서명한다. Notary Server에 의해 메타데이터와 타임스탬프가 확인되고, Notary Signer에 의해 서명된다.

DCT를 이용하는 경우, 시스템 관리자는 특정 태그를 가진 특정 이미지에 대한 서명을 확인할 수 있다. 그러나 이를 위해서는 로컬 레지스트리 서버에 존재하는 이미지 및 태그를 개별적으로 확인하는 작업이 선행되어야 하며, 다음으로 docker trust 명령을 통해 각 이미지에 대한 서명 정보를 요청 및 확인하여야 한다. 본 논문에서 수행한 연구에 따르면, 실제 컨테이너를 배포하는 경우 DCT를 사용하여 디지털 서명이 검증되지 않은 이미지를 실행되지 않도록 활용하는 사례는 존재하지만, 로컬 레지스트리에 존재하는 모든 이미지 및 태그의 서명 정보를 검사하

```
[REGISTRY_HOST[:REGISTRY_PORT]/]REPOSITORY[:TAG]
```

Fig. 4. Identifier of an individual image record

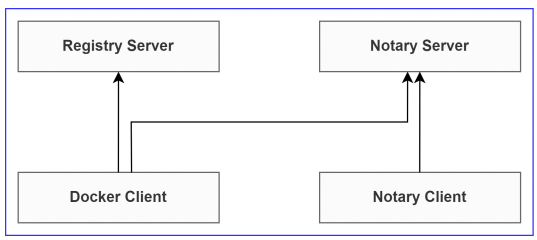

Fig. 5. Structure of DCT

는 사례는 아직 존재하지 않는다. 따라서 본 논문은 기존 기술의 불편성을 해소하는 유일한 구현이라고 할 수 있다.

3.2 Falco

Falco는 쿠버네티스 런타임 보안을 위한 CNCF의 프로젝트 중 하나이다. 이 기술은 호스트와 컨테이너 모두에 이상 행동에 대한 탐지 엔진을 제공하기 위해 런타임에 커널로부터 후킹한 리눅스 시스템 콜을 파싱하여 이벤트를 추출하고, 해당 이벤트를 필터링하는 강력한 규칙 엔진을 제공한다.

Falco는 기본적으로 리눅스 시스템 콜을 사용하여 이벤트를 탐지한다. 이를 위해 커널에 자체적으로 제작한 모듈이나 eBPF 프로브 등을 활용하여 시스템 콜을 후킹할 수 있다. 이는 쿠버네티스 네이티브를 지원하여, 모든 시스템 콜이 쿠버네티스 클러스터의 컨텍스트를 태그할 수 있다. Namespace, deployment, daemonset 및 pod 등의 쿠버네티스 리소스 또한 falco를 통해 추적할 수 있다. 컨테이너 애플리케이션 또한 결론적으로 호스트 커널에 시스템 콜을 요청하기 때문에, falco는 호스트 및 컨테이너에 관계 없이 모든 이벤트를 탐지할 수 있다. falco는 Fig. 6.와 같은 구조를 가진다 [9].

Falco는 사용자가 모니터링 시스템을 편리하게 배포하고, 시스템에서 직접 실행시키는 것을 피할 수 있도록 Docker를 이용한 배포를 제공하지만, 이를 하나의 가시화된 모니터링 시스템으로 구축하기 위해

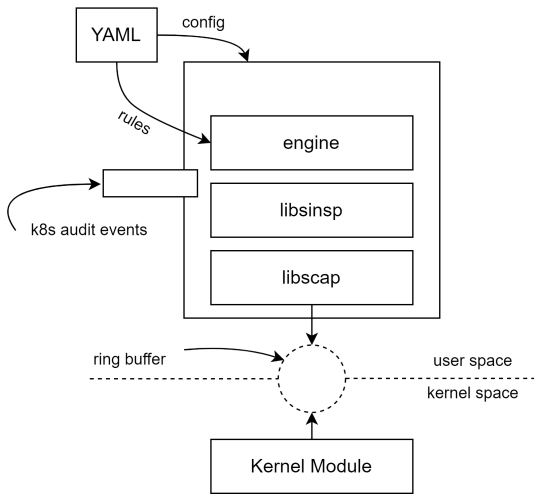

Fig. 6. Architecture of Falco

서는 많은 노력이 필요하다. 이를테면, 이를 가시화하기 위해서는 다른 많은 오픈소스들과 통합이 필요하며, 모니터링 시스템 자체가 컨테이너 위에서 동작하는 경우 이 구현은 더욱 복잡해진다. 그러나 이러한 falco 및 여러 오픈소스들은 각각이 독립적인 프로젝트이므로 이를 통합하여 간편하게 배포하도록 하는 구현은 아직 존재하지 않는다. 따라서 본 논문은 기존 오픈소스들을 통합하며, Docker 컨테이너 위에서 편리하게 배포할 수 있도록 하는 고유한 패키지를 제공할 수 있다.

IV. 제안 모델

4.1 서명 검사기

본 절에서는 DCT를 활용한 간편하고 자동적인 서명 검사기를 제안한다. 3.1절에서는 기존 DCT 기술이 어떠한 방식으로 이미지를 저장하고, 이에 대해 서명을 제공하는지 확인하였다. 또한, 이러한 이미지에 대한 서명은 각 태그에 대하여 발생하므로, 같은 이미지에 대해 태그가 다를 경우 각각 고유한 서명을 가진다는 것을 확인하였다.

4.1.1 제안 모델

본 절에서는 3.1절에서 설명한 DCT 기술을 활용하여 레지스트리에 존재하는 모든 이미지에 대한 서명 검사를 자동으로 수행하는 서명 검사기를 제안한다. 기존 DCT 사용의 불편성을 해결하기 위해 주기적으로 Docker 레지스트리를 스캔하고 서명 및 서명되지 않은 이미지를 자동적으로 확인하는 서명 검사기를 개발한다. 이를 통해 서명된 이미지와 서명되지 않은 이미지에 대한 정보를 확인하고, 시스템 관리자가 레지스트리를 관리하는데 도움을 줄 수 있다.

4.1.2 서명 검사기의 작업 흐름

4.1.1절에서 제안하는 서명 검사기의 작업 흐름도는 Fig. 7.와 같다.

서명 검사기는 다음과 같이 동작한다. 먼저, 서명 검사기 내부에서 로컬 레지스트리에 대해 존재하는 모든 리파지토리를 가져온다. 다음으로 모든 리파지토리에 대해 태그를 가진 이미지의 체크 리스트를 생성한다. 모든 체크리스트에 대해 Notary 서버에 해

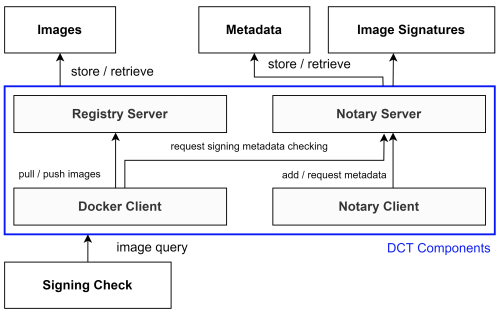

Fig. 7. Workflow of Signing Check using DCT

당 태그를 가진 이미지가 서명되었는지, 혹은 서명되지 않았는지 확인한다.

4.2 모니터링 시스템

본 절에서는 falco 프로젝트를 활용한 간편한 모니터링 시스템 배포를 위한 방법을 제시한다. 3.2절에서는 falco 프로젝트가 커널 모듈 등을 이용하여 컨테이너가 사용하는 모든 시스템 콜을 후킹함으로써 시스템에서 일어나는 모든 일을 관찰할 수 있음을 확인하였다. 또한, 강력한 규칙 엔진을 이용해 위험한 이벤트가 발생할 때 경고를 받을 수 있음을 확인하였다.

4.2.1 모니터링 시스템 예제

3.2절에서 분석한 falco 모니터링 시스템은 강력하지만, 이를 편리한 방법으로 구축하기 위해서는 많은 노력이 필요하다. 이 절에서는, falco 모니터링 시스템을 다른 오픈소스 프로젝트를 사용해 웹 대시보드를 제공하는 서비스를 통해 가시화한다고 가정한다. 또한, 호스트 시스템에 로드를 부과하지 않기 위해 Docker 컨테이너를 통해 모든 서비스를 동작시킨다고 가정한다. 이를 위해서는 falco의 stdout 로그 뿐만 아니라, 이를 metrics 형태로 추출하여야 한다. 추출한 metrics를 local의 Prometheus 서비스와 연결해야 하며, 이는 가시화를 위해 Grafana 서비스와 연결되어야 한다.

4.2.2 제안 모델

보안 관리자에게 요구되는 이러한 부가적인 노력은 결론적으로 모니터링 시스템의 구축을 어렵게 한다. 따라서 본 절에서는 위에서 언급한 falco를 이용한 모니터링 시스템을 구축할 때의 불편성을 해소하기 위한 방안을 제시한다.

4.2.1절에서 설명한 falco 및 오픈 소스 기반의 모니터링 시스템의 구조는 Fig. 8.과 같다.

먼저 falco 이벤트를 prometheus metrics로 추출하기 위해서는 falco-exporter를 사용한다. falco-exporter는 falco 서비스와 로컬 유닉스 소켓을 공유한다. 따라서 개별적인 Docker 컨테이너에서 falco와 falco-exporter를 사용하는 경우 파일 시스템으로 구성된 유닉스 소켓을 공유할 수 없다. 이를 해소하기 위해 기존의 falco 컨테이너 이미지를 수정하여 falco-exporter를 falco 서비스의 백그라운드에서 수행하도록 하는 새로운 이미지를 만든다. 새로운 이미지는 Fig.9.와 같은 구조를 가진다.

다음으로 falco를 이용한 모니터링 시스템을 간편하게 배포할 수 있도록 하는 docker-compose 및 설정 파일을 포함하는 패키지의 형태를 제안한다. 간편화된 모니터링 시스템을 위한 패키지는 Fig. 10.와 같은 구조를 가진다.

Fig. 10.과 같은 패키지 구조를 활용하는 docker-compose.yml 파일은 Fig. 11.와 같이 구성할 수 있다.

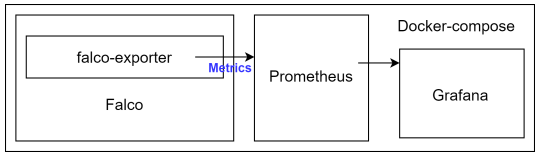

Fig. 8. Structure of Monitoring System using Open Source Projects

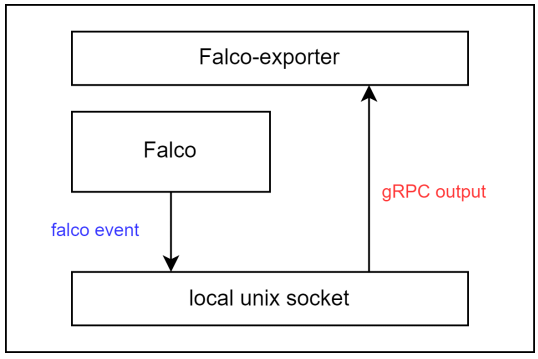

Fig. 9. Structure of New Falco Image

Fig. 11.의 설정 파일은 실제로 동작하지 않으며, Docker 컴포즈 버전, 해당 이미지들을 적절하게 구동시키기 위한 추가적인 볼륨 마운트 및 네트워크 설정 등을 필요로 한다. 여기서는 본 논문에서 생성한 새로운 컨테이너 이미지로 기존의 falco 이미지를 대체하였으며, 패키지의 경로를 이용하여 필수적인 규칙 파일, prometheus와 grafana의 설정 파일 등을 구성하였다.

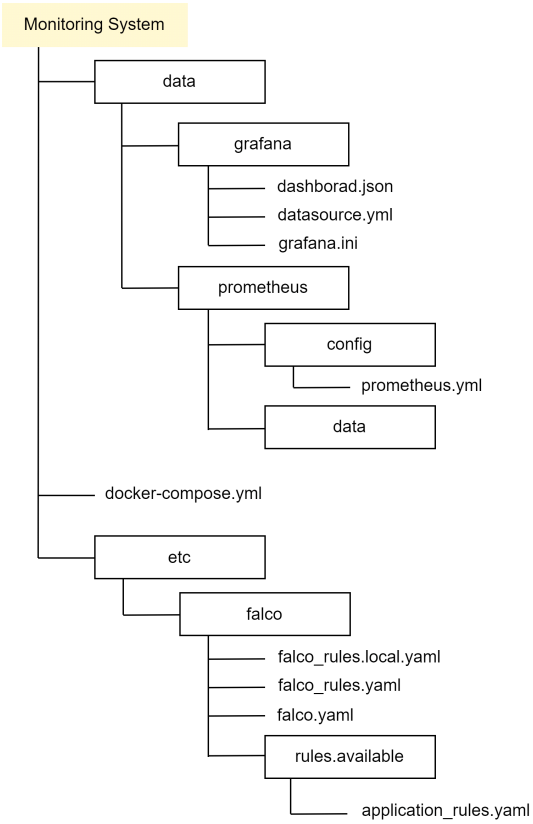

Fig. 10. Structure of Simplified Monitoring System Package

```
services:
  falco:
    image: song22/falco-and-exporter:v2
    volumes:
      - $HOME/falco-monitoring/.falco:/root/.falco
      - $HOME/falco-monitoring/etc/falco/:/etc/falco/

  prometheus:
    image: prom/prometheus:latest
    volumes:
      - $HOME/falco-monitoring/data/prometheus/config:/etc/prometheus
      - $HOME/falco-monitoring/data/prometheus/data:/prometheus

  grafana:
    image: grafana/grafana:latest
    volumes:
      - $HOME/falco-monitoring/data/grafana/grafana.ini:/etc/grafana/grafana.ini
      - $HOME/falco-monitoring/data/grafana/datasource.yml:/etc/grafana/provisioning
/datasources/datasource.yaml
      - $HOME/falco-monitoring/data/grafana/dashboard.json:/etc/grafana/provisioning
/dashboards/dashboard.json
```

Fig. 11. docker-compose configuration

V. 평 가

5.1 서명 검사기

본 절에서는 4.1절에서 제안한 서명 검사기에 대한 결과를 확인한다. Fig. 12와 같이 로컬 레지스트리에 네 개의 이미지가 존재하는 상황을 가정한다.

서명 검사기가 없는 경우, 시스템 관리자는 직접 Curl 명령어를 사용하여 로컬 레지스트리 서버에 어떠한 이미지가 있는지 검색하여야 한다. 다음으로, 검색한 이미지에 대한 모든 태그를 검색하여 확인하고자 하는 〔특정 이미지 이름:태그 이름〕 쌍을 찾는다. 최종적으로 위를 이용해 “docker trust” 명령으로 서명을 확인한다. 만약 레지스트리에 확인하고자 하는 이미지가 많을 경우, 모든 이미지에 위와 같은 작업을 반복 수행함으로써 서명을 확인할 수 있다. 반면 서명 검사기를 사용하는 경우, 시스템 관리자가 리파지토리 및 이미지를 검색하고 각 태그에 대한 서명을 요청할 필요 없이, 서명 검사기를 통해 레지스트리에 존재하는 모든 이미지에 대한 서명을 검사할 수 있다.

서명 검증 기능을 평가하기 위해 Connaisseur라는 쿠버네티스의 Admission Controller를 이용한다. Connaissuer는 쿠버네티스에서 컨테이너 이미지에 서명을 확인하기 위한 관리 도구이며, 이는 서명되지 않은 이미지를 클러스터에 배포할 수 있는지 여부를 결정한다.

벤치마크는 다음과 같이 진행된다. 우선 Connaisseur를 이용해 단일 이미지 배포에 대한 실행시간을 측정한다. 이를 위해 Fig. 12.의 4개의 이미지의 서명을 확인하기 위해 서명 검사기를 사용하였다. 그 결과는 Table. 1.과 같이, 사용자와 시스템 런타임에서는 서명 검증 기능이 더 오랜 시간이 걸리지만, 코드의 단순성 덕에 서명 검증 기능의 모든 하위 프로세스를 포함하는 실제 경과 시간을 확인했을 때 훨씬 빠른 속도를 보이는 것을 확인하였다.

```
{"repositoties" : ["apline","golang","jenkins","mariadb"]}
```

Fig. 12. Images of Registry

Table. 1. Execution time of Connaisseur and Signing Check (ms)

	Connaisseur	Signing Checker
Average User Runtime	0.115	0.195
Agerage System Runtime	0.035	0.16
Total Elapsed Time	3.8	0.25

5.2 모니터링 시스템

4.2절에서 제안하는 모니터링 시스템은 "docker-compose up" 명령 하나로 간단하게 수행될 수 있으며, 그 결과는 Fig. 14.와 같이 Grafana 웹 서비스를 통해 로컬에서 확인할 수 있다. 이 결과는 시스템 콜을 후킹해 규칙에서 설정된 규칙에 따라 이를 필터링하여 가시화한 결과이며, 웹 대시보드의 일부이다.

job	priority	rule	source
falco-exporter	3	Create files below dev	SYSCALL
falco-exporter	3	Mkdir binary dirs	SYSCALL
falco-exporter	3	Modify binary dirs	SYSCALL
falco-exporter	3	Write below binary dir	SYSCALL

Fig. 13. Result of Monitoring System

5.2.1 모니터링 시스템을 활용한 악성 행위 탐지

본 절에서는 제안한 모니터링 시스템을 통해 호스트 시스템에서 발생하는 악성 행위를 탐지하는 예를 보인다. 먼저 Falco의 Event-generator 프로그램을 이용해 민감한 시스템 동작을 수행시킨 결과는 Table. 2.와 같다. 이 표에서의 Operation은 테스트를 위해 수행시킨 Operation의 종류를 의미하며, Observed Attributes는 모니터링의 결과로 확인 가능한 커널 오브젝트이다. 표 내부의 Attiribute는 파싱 가능한 속성을 분류한 것이며, Example은 실제로 파싱된 결과의 예이다.

이 외에도 모니터링 시스템은 시스템 콜을 사용하는 모든 행위를 커널 레벨에서 탐지할 수 있으며, 이는 컨테이너에서도 수행될 수 있다. 이때 어떤 커널 오브젝트를 파싱할지는 사전에 Rule을 통해 정의하

Table. 2. Monitoring Result against to sensitive system operation

Operation	Observed Attributes	
Network Activity	Attribute	Example
	User	root
	User_loginuid	-1
	Command	Sha1sum -loglevel info run ^helper.NewNetworkActivity$
	Connection	172.17.0.2:50-> 10.2.3.4:8192
	Container _id	2bbeacd45b1e
NonSudo Setuid	User	bin
	User_loginuid	-1
	Cur_uid	2
	Parent	child
	Command	child --loglevel info run ^syscall.NonSudoSetuid$
	Uid	root
	Container _id	2bbeacd45b1e
System User Interactive	User	bin
	User_loginuid	-1
	Command	login
	Container_id	2bbeacd45b1e

여야 한다.

5.2.2 모니터링 시스템을 활용한 CVE-2022-0492 탐지

본 절에서는 모니터링 시스템의 효용성을 보이기 위해 모니터링 시스템을 이용해 CVE를 탐지하는 예를 보인다. 본 실험에서는 Ubuntu 18.04 LTS 환경에서 Docker 19.03.10 버전을 사용하였다.

본 CVE는 Docker 컨테이너에서 Cgroup에 존재하는 취약점을 이용해 컨테이너를 탈출하고 권한을 상승하는 취약점이다.

해당 취약점을 테스트하기 위해 악용될 Cgroup은 control group으로, Linux kernel feature로써 프로세스를 계층적인 그룹으로 구성하여 관리자가 프로세스 집합의 자원 사용을 제한 및 고립할 수 있다. 이러한 Cgroup은 v1과 v2가 존재하며, Cgroup v1에 해당 취약점이 존재한다. 본 CVE는 컨테이너가 Cgroup의 v1 구조를 가지고, CAP_SYS_ADMIN capability를 사용할 수 있으

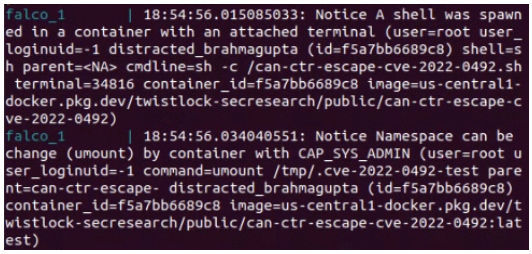

Fig. 14. CVE-2022-0492 Detection Experimental Result

며, AppArmor나 security option인 Seccomp 등이 비활성화된 컨테이너에서 umount 혹은 unshare 등의 명령으로 컨테이너를 탈출할 수 있다. Fig. 14.는 본 CVE를 악용하여 컨테이너 탈출을 시도했을 때, 모니터링 시스템을 활용해 이를 탐지한 결과이다.

실험 결과, 터미널을 포함한 컨테이너가 생성됨을 탐지함과 더불어, umount를 수행하는 악의적인 이미지에 대해 각각 실제로 실행된 커맨드 및 이미지, 컨테이너 등을 탐지할 수 있음을 확인하였다.

5.2.3 성능 벤치마크

벤치마크는 향후 연구에서 사용될 Falco 내부의 eBPF 모듈에 대한 퍼포먼스를 테스트함으로써 진행된다. 이는 Falco를 이용한 모니터링 시스템 위에 Sysdig, Inc.에서 제공하는 default kernel 모듈을 사용하는 경우의 성능을 비교한다. 두 모듈 간의 하드웨어 사용량을 VM 환경에서 측정해 비교하며, 그 결과는 Table. 3.과 같다. Table. 3.의 결과를 통해 향후 연구에서 주요하게 사용될 기술인 eBPF를 Kernel 모듈 대신 사용하는 경우에도 시스템 사용량 측면에서는 큰 오버헤드가 없음을 검증하였다.

Table. 3. Hardware Usage of eBPF module and kernel module

	Kernel	eBPF
CPU	about 2%	about 2%
Memory	about 2.7%	about 2.9%

VI. 결 론

본 논문에서는 클라우드 네이티브 애플리케이션에서 발생하는 여러 보안적 위험을 완화하기 위한 툴들을 제안 및 구현하였다. 첫 번째로, 클라우드 네이티브 애플리케이션의 주요하게 사용되는 기술인 컨테이너의 소스가 되는 이미지의 무결성을 확인하기 위해 DCT 기술을 이용한 서명 검사기를 제안하였다. 해당 서명 검사기는 DCT의 불편성을 해소하며, 주기적으로 Docker 레지스트리를 스캔 및 레지스트리에 존재하는 전체 이미지에 대한 서명을 자동적으로 확인한다. 이는 시스템 관리자가 서명 정보 및 레지스트리 이미지를 관리하는 데에 도움을 줄 수 있다.

다음으로, 컨테이너 애플리케이션이 동작하는 런타임에서의 보안을 위해 기존 프로젝트의 배포와 통합 면에서의 어려움을 해결하기 위해, 본 논문에서는 이를 쉽게 배포하고 통합할 수 있게 하는 이미지를 구현, docker-compose 및 패키지의 구조를 제안하였다. 이를 통해 시스템 관리자는 더욱 편리하게 서비스에 대한 모니터링 시스템을 구축하고 관리할 수 있다.

본 논문에서 제안한 모델을 이용하여 클라우드를 사용하는 경우, 이미지의 무결성 및 런타임 보안 측면에서 더욱 편리하게 보안성을 제공할 수 있다. 또한, 간편하게 모니터링 시스템을 배포함으로써 시스템에서 일어나는 일에 대한 가시성 및 관찰 가능성을 얻을 수 있다. 향후 이미지 무결성 및 런타임 보안 측면뿐 아니라, 클라우드 네이티브 환경에 대한 다른 보안적 측면을 해결하는 연구를 추진할 계획이다.

References

[1] M. Peter and G. Timothy, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, Sept 2011

[2] CLOUDMATE, "What is cloud native?", https://cloudmt.co.kr/?p=3927, Last Accessed 12 Fed. 2022

[3] RedHat, "Comparison between container and VM", https://www.redhat.com/ko/topics/containers/containers-vs-vms, Last Accessed 01 Fed. 2022

[4] Aqua, "Container Images: Architecture and Best Practices", https://www.aquasec.com/cloud-native-academy/container-security/container-images/, Last

Accessed 07 Jan. 2022
〔5〕 Microsoft, "Information of registry, repository, artifacts", https://docs.microsoft.com/ko-kr/azure/container-registry/container-registry-concepts, Last Accessed 14 Jan. 2022
〔6〕 S. Murugiah, M. John and S. Karen, "Application Container Security Guide", NIST Special Publication 800-190, Sept 2017
〔7〕 Docker, "Content trust in Docker", https://docs.docker.com/engine/security/trust/, Last Accessed 04 Jan. 2022
〔8〕 Trendmicro, "Docker Content Trust: What It Is and How It Secures Container Images", https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/docker-content-trust-what-it-is-and-how-it-secures-container-images, Last Accessed 01 Fed. 2022
〔9〕 Falco, "Getting Started", https://falco.org/docs/getting-started, Last Accessed 07 Jan. 2022

〈저자소개〉

곽 송 이 (Songi Gwak) 학생회원
2021년 2월: 창원대학교 컴퓨터공학과 졸업
2021년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈관심분야〉 클라우드 보안, AI 보안, 네트워크

응웬부렁 (Long Nguyen-Vu) 정회원
2012년 2월: National University of Information Technology, Ho Chi Minh 졸업
2014년 3월~2016년 2월: 숭실대학교 정보통신공학과 석사
2016년 3월~2022년 2월: 숭실대학교 정보통신공학과 박사
2022년 3월~현재: 숭실대학교 통신망보안연구실 연구원
〈관심분야〉 클라우드 보안, AI 보안

정 수 환 (Souhwan Jung) 종신회원
1985년 2월: 서울대학교 전자공학과 졸업
1987년 2월: 서울대학교 전자공학과 석사
1996년 6월: University of Washington 박사
1988년~1991년: 한국통신 전임 연구원
1997년~현재: 숭실대학교 전자정보공학부 교수
〈관심분야〉 AI 보안, 모바일 보안, 클라우드 보안, 네트워크 보안