안전한 컨테이너 이미지 레지스트리 제공을 위한 파이프라인 설계 방안에 관한 연구

고성재[1], 김선집[2*]
[1]씨엠티정보통신 책임컨설턴트, [2]한세대학교 ICT 융합학과 교수

A Study on Pipeline Design Methods for Providing Secure Container Image Registry

Seong-Jae Ko[1], Sun-Jib Kim[2*]
[1]Senior Consultant, CMT INFO&COMM CO.,LTD
[2]Professor, Division of ICT Convergence Engineering, Hansei University

요 약 애플리케이션의 개발 및 배포 방식이 모노리스에서 마이크로서비스로 전환되며 경량 가상화 기술인 컨테이너가 IT 핵심 기술로 자리 잡고 있다. 그러나 컨테이너 기술은 기존 하이퍼바이저 기반의 가상머신과 달리 동일한 커널을 공유하는 방식으로 구체적인 보안 경계를 제공하지 못한다. 다양한 선행연구에 따르면 현재 공유되는 대부분의 컨테이너 이미지에는 다수의 보안 취약점이 존재한다. 이에, 공격자들은 보안 취약점을 이용하여 익스플로잇을 시도할 수 있으며 이는 시스템 환경에 심각한 영향을 미칠 수 있다. 따라서 본 연구에서는 보안 취약점이 존재하는 컨테이너 이미지가 배포되는 것을 방지하기 위한 효율적인 자동화 배포 파이프라인 설계 방안을 제시한다. 이를 통해 안전한 컨테이너 환경을 제공할 수 있을 것이다.

주제어 : 클라우드, 마이크로서비스, 컨테이너, 하이퍼바이저, 파이프라인

Abstract The development and distribution approach of applications is transitioning from a monolithic architecture to microservices and containerization, a lightweight virtualization technology, is becoming a core IT technology. However, unlike traditional virtual machines based on hypervisors, container technology does not provide concrete security boundaries as it shares the same kernel. According to various preceding studies, there are many security vulnerabilities in most container images that are currently shared. Accordingly, attackers may attempt exploitation by using security vulnerabilities, which may seriously affect the system environment. Therefore, in this study, we propose an efficient automated deployment pipeline design to prevent the distribution of container images with security vulnerabilities, aiming to provide a secure container environment. Through this approach, we can ensure a safe container environment.

Key Words : Cloud, Microservices, Container, Hypervisor, Pipeline

*교신저자 : 김선집(kimsj@hansei.ac.kr)
접수일 2023년 3월 26일 수정일 2023년 5월 26일 심사완료일 2023년 5월 29일

1. 서론

1.1 연구 배경 및 목적

코로나 유행 이후 재택근무 확대와 온라인 수업 등의 폭발적인 수요 증가로 인해 국내 클라우드 서비스 활용이 급격히 증가하고 있다[1]. 또한, 애플리케이션의 개발 및 배포 방식이 모노리스(Monolith)에서 마이크로서비스(Microservice)로 전환되며 경량 가상화 기술인 컨테이너가 IT의 핵심 기술로 자리 잡고 있다[2]. 컨테이너는 하이퍼바이저(Hypervisor) 기반의 가상머신과 달리 동일한 운영체제(OS) 커널을 공유함에 따라 사용자가 애플리케이션을 배포하는데 필요한 리소스와 시작 시간을 단축해 클라우드 환경의 서비스 이용을 증대시키고 있다[3].

CNCF(Cloud Native Computing Foundation)의 2021년도 조사에 따르면 컨테이너 기술을 제공하는 오케스트레이션 도구인 쿠버네티스를 이미 사용하고 있거나 도입을 고려하고 있다고 응답한 조직이 무려 96%에 달한다[15]. 이처럼 다양한 장점들로 인해 컨테이너가 주목받고 있으나 컨테이너 기술은 호스트 운영체제의 커널을 공유함에 따라 가상머신만큼의 명확하고 구체적인 보안 경계를 제공하지는 못한다[4]. 따라서 컨테이너 환경에서는 클러스터 전반에 걸친 보안 설정 들을 준수해야 하며 특히나 안전한 컨테이너 환경을 위해서는 무엇보다 이미지에 대한 취약성 관리가 중요하다[2,3,5]. 이에 본 연구에서 안전한 컨테이너 환경을 제공함으로서 보안 취약점이 존재하는 컨테이너 이미지가 배포되는 것을 방지하기 위한 효율적인 자동화 배포 파이프라인 설계 방안을 제시하고자 한다.

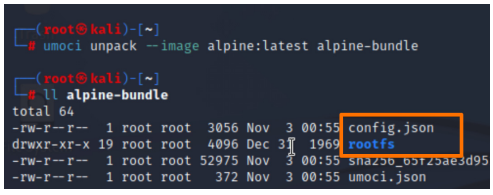
2. 이론 고찰

2.1 컨테이너 개요

컨테이너 기술은 개발과 운영 사이의 긴밀한 조율을 강조하며 애플리케이션의 빌드와 실행 간의 통합을 추구하는 데브옵스와 함께 크게 주목받았다. 컨테이너 기술의 이식성과 선언적 특성으로 인해 개발, 테스트, 운영 환경에서 높은 일관성을 유지할 수 있기 때문이다[6]. 운영체제 수준의 가상화 기술인 컨테이너는 ‘namespaces’, ‘cgroup’, ‘chroot’ 와 같은 리눅스 커널의 핵심 기능을 사용하여 프로세스를 격리된 환경에서 실행시키는 기술이다. 하드웨어 자원까지 가상화하는 가상머신과 달리 커널을 공유하는 방식이므로 실행속도가 매우 빠르며 컨테이너와 가상머신 간의 CPU 성능 비교를 분석한 Zhang 등의 연구에 따르면 하나의 운영체제에서 구동되는 컨테이너의 수가 증가하면 할수록 가상머신에 비해 CPU 성능이 월등히 높아진다[7].

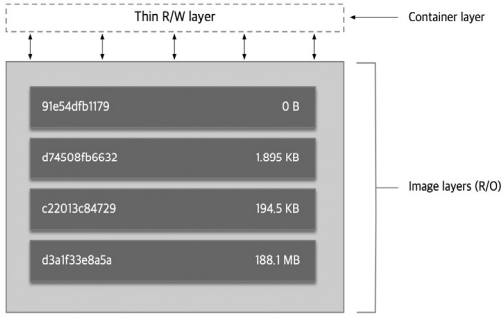
2.1.1 컨테이너 이미지

컨테이너 생명주기의 첫 단계는 애플리케이션의 구성요소를 빌드하여 이미지로 배치하는 것이다[6]. 컨테이너 이미지에는 컨테이너 실행 시 필요로 하는 모든 파일이 포함되어 있다. 아래[Fig. 1]는 umoci 라는 도구를 통해 알파인(Alpine) 컨테이너 이미지를 실제로 확인한 결과이다.



[Fig. 1] Alpine Image (by umoci)

알파인 리눅스 배포판 내용이 담긴 루트 파일시스템(rootfs)과 컨테이너 실행 시 적용할 설정 값(config.json)이 정의되어 있으며 이처럼 컨테이너 이미지는 크게 루트 파일시스템과 설정 정보로 구성된다[4]. 일반적인 컨테이너 이미지의 계층 구조는 아래 [Fig. 2]와 같다.



[Fig. 2] Image Layer

2.1.2 이미지 레지스트리

컨테이너 이미지는 일반적으로 호스트 간에 쉽게 공유하고 재사용할 수 있도록 중앙에 위치한다. 이를 위한 이

미지 저장소를 레지스트리(Registry)라고 말하며 애플리케이션 개발자가 이미지 빌드 시 쉽게 저장 및 식별할 수 있도록 태그나 카탈로그로 지정할 수 있다. 이미지 레지스트리의 대표적인 예로는 Amazon EC2 Container Registry, Docker Hub, Quay Container Registry가 있다[6].

2.1.3 컨테이너 런타임

컨테이너를 사용하게 되면 여러 애플리케이션이 동일한 운영체제 커널을 공유하며 모든 호스트 운영체제에는 컨테이너 런타임이라고 불리는 각 컨테이너에 대한 환경을 설정하고 유지하는 바이너리가 존재한다[6]. 컨테이너 런타임은 컨테이너 이미지에 포함된 루트 파일시스템과 설정 정보를 통해 컨테이너를 인스턴스화 한다[4]. 컨테이너 런타임의 종류로는 아래 〈Table 1〉과 같다.

〈Table 1〉 Container Runtime Type

Type	Container Runtimes
Low-Level	runC, crun, containerd
High-Level	Docker Engine, Podman, CRI-O, Mirantes
Sandboxed and Virtualized	gVisor, nabla-containers, kata-containers

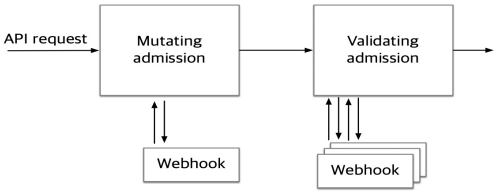
2.2 컨테이너 보안

컨테이너 기술은 컨테이너는 강력한 격리 수준을 제공하지만, 시야가 제한된 리눅스의 프로세스일 뿐이며 호스트 운영체제의 커널을 공유한다는 사실 하나만으로도 가상머신 격리보다 보안성이 약하다고 볼 수 있다. 따라서 부주의하게 구성된 환경에서 컨테이너는 가상머신보다 호스트 및 다른 컨테이너와 훨씬 더 쉽고 직접적으로 상호 작용할 수 있으므로 추가적인 보안 기능들과 샌드박싱을 적용하여 컨테이너 격리를 강화해야 한다[4].

2.2.1 컨테이너 보안 위협

컨테이너 기술의 핵심 컴포넌트에 대한 주요 위협으로는 크게 이미지, 레지스트리, 오케스트레이터, 컨테이너와 호스트 운영체제에 대한 위험이 있다[4].

위와 같은 보안 위협에 대한 보호 대책으로 CIS Benchmark 등에서 제공하는 보안 지침들을 준수함으로써 대부분 해결할 수 있으며 컨테이너 보호 대책의 경우, 컨테이너를 실행시키는 오케스트레이션에서 제공되는 보안 컨텍스트(Security Context)를 적절히 부여함으로써 보안 지침에서 요구하는 사항들을 준수하도록 통제할 수 있다[12,16]. 아래 [Fig. 3]은 대표적인 오케스트레이션 도구인 쿠버네티스의 승인 컨트롤러의 동작 방식으로써 API 서버로 전송된 요청을 검사 및 검증하고 변경할 방법을 제공하는데 승인 컨트롤러에 PSA(PodSecurityAdmission)와 같은 정책 엔진을 적용함으로써 컨테이너에 적용되는 보안 컨텍스트를 통제할 수 있다.



[Fig. 3] API Server's Admission Controller

이처럼 오케스트레이션에서 제공하는 정책 엔진을 통해 클러스터 관리자는 클러스터 내부에서 생성되는 모든 파드(Pod)의 보안 위배사항을 통제할 수 있으며 앞서 확인한 보안 위협의 많은 부분을 해결할 수 있다[11]. 그러나 이미지에 대한 보호 대책의 경우 오케스트레이션 도구 자체에서 제공되는 기능이나 런타임에서 제공되는 플러그인만으로는 해결시키는 데 어려움이 있으며 조직 내부의 개발단계부터 실제 배포단계까지의 적절한 파이프라인 배치 방법 등이 고려되어야 한다[6].

3. 관련 연구

3.1 컨테이너 이미지 보안성 동향 분석

3.1.1 컨테이너 이미지 보안성 분석에 관한 연구

You 등이 주장한 연구에서는 다양한 컨테이너 이미지를 공유할 수 있게 해주는 레지스트리 서비스인 도커 허브에서 공유되는 이미지의 취약성을 조사하기 위해 자동으로 컨테이너 이미지를 수집하고 분석하는 프레임워크를 설계하고 공식 이미지와 가장 많이 다운로드 된 1만 개의 커뮤니티 이미지를 분석하였다[3]. 연구 결과에 따르면 공식 이미지와 커뮤니티 이미지 모두 평균적으로 117개 이상의 보안 취약점을 가지고 있음을 알 수 있다. 또한, 공식 이미지 내에서 평균적으로 발견된 11개의 취약점 대부분은 커뮤니티 이미지에서도 동일하게 발견되었는데 이는 다수의 커뮤니티 이미지가 공식 이미지를 베이스 이미지로 사용하기 때문으로 추측된다.

즉, 도커 허브에서 직접 관리하는 이미지라고 하더라도 무분별하게 사용하는 경우 심각한 결과를 초래할 수 있으며 대부분의 커뮤니티 이미지가 공식 이미지를 상속받아 만들어지므로 공식 이미지의 취약점이 그대로 전파될 수 있음을 의미한다. 결론적으로 공식 이미지를 포함한 대부분의 컨테이너 이미지에서 다양한 취약점이 존재함을 알 수 있으며 컨테이너 이미지가 계층 구조로 구성됨에 따라 기본 베이스 이미지에서 발생하는 소수의 취약점이 하위의 수많은 다른 이미지에 영향을 미침을 알 수 있다.

3.1.2 과학적 데이터 분석을 위한 컨테이너 이미지 보안 취약성 분석

Kaur 등이 주장한 연구에서는 4종의 컨테이너 이미지 스캐너(Anchore, Vuls, Clair, Singularity Tools)를 통해 신경과학에 사용되는 2개의 컨테이너화된 프레임 워크를 대상으로 컨테이너 이미지 업데이트 및 이미지 축소가 보안 취약점에 미치는 영향을 분석하였다. 분석 결과, 검출된 보안 취약점 수와 컨테이너 이미지 내 존재하는 패키지 수 사이에 강한 선형관계가 있으며 평균적으로 하나의 패키지가 늘어감에 따라 1.7개의 취약점이 생겨남을 알 수 있으며 컨테이너 이미지를 최신 버전으로 업데이트하는 경우 패키지별 취약점 수가 평균 3배 감소하였다. 결론적으로 패키지 업데이트 및 이미지 축소를 통해 컨테이너 이미지에 존재하는 다양한 유형의 취약점을 제거할 수 있음을 알 수 있다[5]. 또한, 대부분의 컨테이너 이미지에서 상당한 양의 취약점이 발견되었으나 알파인 배포판을 기반으로 한 이미지에서 취약점이 가장 적은 편으로 확인되었는데 실제 사례를 살펴보면 현재 베이스 이미지로 가장 많이 사용되는 알파인 배포판 이미지에서도 2019년도에 CVSS 최고점에 달하는 취약점이 발견되었다. 해당 취약점은 알파인 이미지 기반의 리눅스 PAM을 사용하는 애플리케이션이 있는 컨테이너 또는 시스템 shadow 파일을 인증 데이터베이스로 사용하는 메커니즘을 이용하는 경우 슈퍼 유저인 루트 계정에 NULL 패스워드가 통과될 수 있다는 취약점이다[8].

이 취약점은 일반 사용자가 컨테이너에 접근하여 별다른 인증 없이 루트 계정을 획득할 수 있다는 것이며, 이러한 컨테이너 이미지 취약점의 문제점은 해당 알파인 이미지를 사용하는 경우뿐 아니라 대부분 운영 환경에서 사용되는 이미지가 이러한 베이스 이미지를 기반으로 만들어진 하위 이미지라는 특성 때문에 동일한 문제점을 가지게 된다.

4. 제안하는 방안

4.1 안전한 컨테이너 이미지 레지스트리 환경을 위한 파이프라인 설계

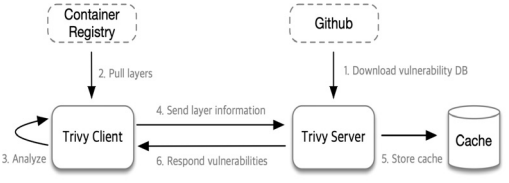
본 절에서는 안전한 컨테이너 이미지 레지스트리 환경을 제공하기 위해 자동화 배포 파이프라인 설계를 제안하고자 한다.

4.1.1 아르고 워크플로(Argo Workflow)

쿠버네티스 환경에서 지속적인 통합 및 배포를 수행할 수 있는 컨테이너 네이티브 워크플로 엔진인 아르고 워크플로는 기존 워크로드 사용 시 복잡한 작업이 맞물려 실행될 때 이를 유기적으로 연결하기 어렵다는 단점을 해소하였다. 아르고 워크플로를 사용하면 작업 실행의 단위가 컨테이너로 분리되므로 개별 작업마다 독립적인 환경을 제공할 수 있으며 DAG(Directed Acyclic Graph)를 사용하여 작업 간의 연결 상태를 확인하고 전체 워크로드 파이프라인을 시각화하여 관리할 수 있다[9]. 또한 아르고 워크플로는 규칙에 맞도록 YAML 파일만 생성할 수 있다면 어디서든 아르고 서버에 요청하여 워크플로를 실행시킬 수 있으며 아르고 규칙에 맞는 YAML 파일을 만들 수 있다면 어떠한 언어라도 워크플로를 실행할 수 있다는 장점이 있다.

4.1.2 이미지 스캐너 트리비(Trivy)

트리비는 아쿠아 시큐리티(Aqua Security)사에서 개발한 컨테이너 이미지에 대한 오픈 소스 스캐너로써 컨테이너 이미지에 존재하는 보안 취약점을 검출할 수 있다[13]. 아래 [Fig. 4]와 같이 트리비 클라이언트에서 컨테이너 이미지를 풀링한 이후 트리비 서버로 대상 이미지를 전달하면 트리비 서버는 깃허브로부터 내려받은 CVE 데이터베이스를 통해 스캐닝을 진행한 후 결괏값을 클라이언트로 응답한다.



[Fig. 4] Trivy Architecture

쿠버네티스 클러스터 내부에 트리비 서버를 파드형태

로 구성한 후 Job 형태로 스캐닝 대상 이미지를 전달한 후 파드의 로그를 확인하면 아래 [Fig. 5]와 같다. 이처럼 트리비를 통해 컨테이너 이미지에 존재하는 보안 취약점 정보를 모두 확인할 수 있다.

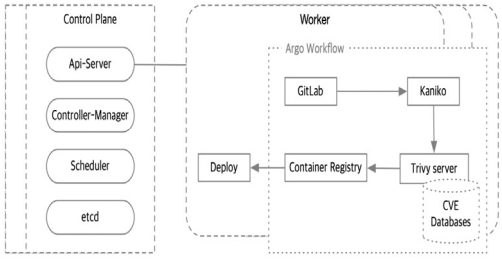
```
→ kubectl logs image-scan-vhks7
2022-12-04T07:28:47.226Z        WARN    'client' subcommand is deprecated now. S
2022-12-04T07:28:47.228Z        INFO    Vulnerability scanning is enabled
2022-12-04T07:28:47.228Z        INFO    Secret scanning is enabled
2022-12-04T07:28:47.228Z        INFO    If your scanning is slow, please try '--
2022-12-04T07:28:47.228Z        INFO    Please see also https://aquasecurity.git
ion
2022-12-04T07:28:49.957Z        WARN    This OS version is no longer supported b
2022-12-04T07:28:49.957Z        WARN    The vulnerability detection may be insuf

alpine:3.10.2 (alpine 3.10.2)
=============================
Total: 10 (HIGH: 9, CRITICAL: 1)
```

Library	Vulnerability	Severity	Installed Version	Fixed Version
apk-tools	CVE-2021-36159	CRITICAL	2.10.4-r2	2.10.7-r0
	CVE-2021-30139	HIGH		2.10.6-r0
busybox	CVE-2021-28831		1.30.1-r2	1.30.1-r5
libcrypto1.1	CVE-2020-1967		1.1.1c-r0	1.1.1g-r0
	CVE-2021-23840			1.1.1j-r0
	CVE-2021-3450			1.1.1k-r0

[Fig. 5] Image Scanning Result (by trivy)

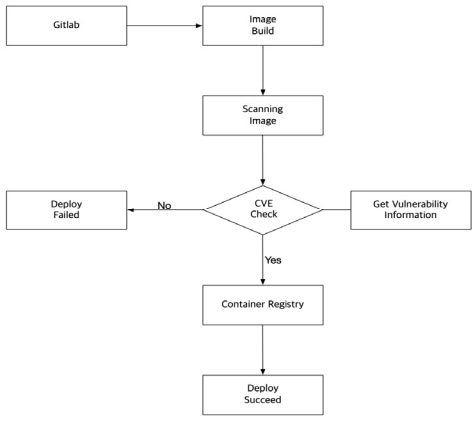
4.1.3 자동화 배포 파이프라인 설계

쿠버네티스 클러스터 환경에서 아르고 워크플로와 컨테이너 이미지 스캐너인 트리비를 활용하여 보안 취약점을 보유하고 있는 컨테이너 이미지가 배포되는 것을 방지하도록 자동화 파이프라인을 설계하였다.



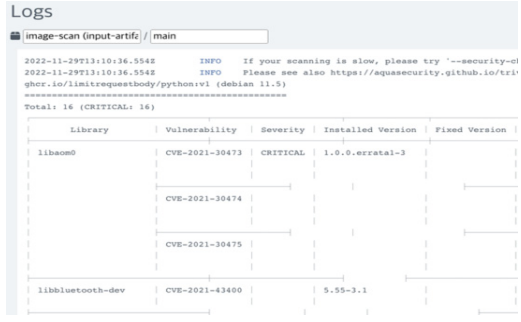
[Fig. 6] Automated Deploy Pipeline Architecture

아래 [Fig. 7]와 같이 컨테이너 배포를 위해 개발자가 작성한 소스 코드와 컨테이너 이미지를 깃랩(Gitlab) 서버로부터 풀링하여 카니코(Kaniko)를 통해 빌드한 후 빌드된 이미지를 트리비를 통해 스캐닝하여 보안 취약점 검출 여부에 따라 배포 진행을 결정하도록 구성하였다. 아르고 워크플로를 활용하면 워크플로에 트리비를 적용함으로써 이미지 스캐닝 결괏값 기반으로 배포 파이프라인을 구성할 수 있다.



[Fig. 7] Automated Deploy Pipeline Diagram

실제로 위와 같은 방식으로 워크플로를 생성하여 보안 취약점을 보유한 컨테이너를 배포하게 되면 배포가 실패됨을 알 수 있으며 아래 [Fig. 8]과 같이 로그를 확인할 수 있다. 로그 내용을 살펴보면 배포를 수행한 컨테이너의 libaom0 라이브러리에 존재하는 취약점(CVE-2021-30473)을 포함하여 16개의 CRITICAL 심각도의 취약점이 검출됨에 따라 실패했음을 알 수 있다.



[Fig. 8] Deploy Failed Logs

이처럼 아르고 워크플로와 컨테이너 이미지 스캐너인 트리비를 통해 쿠버네티스 환경에서 보안 취약점이 존재하는 컨테이너 이미지가 배포되지 못함에 따라 안전한 컨테이너 환경을 제공할 수 있음을 확인할 수 있었다.

5. 결론

본 연구에서는 컨테이너 환경의 보안 위협을 분석하고 컨테이너 이미지 보안성에 관한 선행연구들을 분석하여

안전한 컨테이너 이미지 레지스트리 제공하기 위한 파이프라인 설계 방안을 제안하였다. 개발자가 애플리케이션 소스 코드를 컨테이너 형태로 패키징 하여 빌드하는 시점부터 운영 환경에서 사용되는 이미지 레지스트리에 배포되는 파이프라인을 아르고 워크플로를 통해 구성하였으며 파이프라인 내 이미지 스캐너를 배치함으로써 특정 레벨 이상의 보안 취약점이 발견되는 경우 배포가 실패하도록 구성하였다. 이를 통해 공격자가 쉽게 악용할 수 있는 보안 취약점을 보유한 컨테이너가 운영 환경에 배포되지 못하도록 방지함으로써 안전한 컨테이너 환경을 제공할 수 있다.

REFERENCES

[1] M.H.Park, D.H.Kim, H.S.Han and Y.J.Lee, "Server Workload Security Trends in Cloud Environments," *Korea Institute of Information Security and Cryptology*, Vol.31, No.3, pp.39-44, 2021.

[2] B.Tak, "A Study on the Security Vulnerabilities of Container Images," *The Journal of Korean Institute of Next Generation Computing,* Vol.14, No.3, pp.7-15, 2018.

[3] M.S.You, J.H.Kim and S.W.Shin, "Revisiting Security Landscape of Docker Hub Container Images," *The Journal of Korean institute of communications and information sciences,* Vol.47, No.8, pp.1231-1243, 2022.

[4] L.Rice, *Container Security: Fundamental Technology Concepts that Protect Containerized Applications*, 1st ed., O'Reilly Media, 2020.

[5] B.Kaur, M.Dugr´e, A.Hanna and T.Glatard, "An analysis of security vulnerabilities in container images for scientific data analysis," *GigaScience*, Vol.10, 2021.

[6] M.Souppaya, J.Morello and K.Scarfone, "Application Container Security Guide," *NIST Special Publication 800-190,* 2017 September.

[7] Q. Zhang, L.Liu, C.Pu, Q.Dou, L.Wu and W.Zhou, "A Comparative Study of Containers and Virtual Machines in Big Data Environment," *2018 IEEE 11th International Conference on Cloud Computing*, 2018.

[8] Aquasec, CVE-2019-5021: Alpine Docker Image 'null root password' Vulnerability[Internet], https://blog.aquasec.com/cve-2019-5021-alpine-docker-image-vulnerability

[9] S.H. Kim and Y.Han. Kim, "A Structural Analysis of Argo Workflows for Task Automation in Kubernetes Environment," *The Journal of Korean institute of communications and information sciences*, pp.910-911, 2021.

[10] National Security Agency, Cybersecurity and Infrastructure Security Agency, "Kubernetes Hardening Guide," Cybersecurity Technical Report, 2022.

[11] M.Slik, Validating the replacement filtering features of popular alternative admission controllers for Pod Security Policies[Internet] https://rp.os3.nl/2020-2021/p76/report.pdf

[12] Container-security-checklist[Internet], https://github.com/krol3/container-security-checklist

[13] AquaSecurity, Trivy[Internet], https://github.com/aquasecurity/trivy

[14] Argo Workflow[Internet], https://argoproj.github.io/argo-workflows/

[15] CNCF Annual Survey 2021[Internet], https://www.cncf.io/reports/cncf-annual-survey-2021/

[16] CIS Benchmark[Internet], https://www.cisecurity.org/cis-benchmarks/

[17] Docker [Internet], https://docs.docker.com/

고 성 재(Seong-Jae Ko) [정회원]

■ 2023년 1월 ~ 현재 : 씨엠티정보통신(주) 컨설팅사업본부 책임컨설턴트

〈관심분야〉
정보보안, 클라우드, 컨테이너

김 선 집(Sun-Jib Kim) [정회원]

■ 2014년 3월 ~ 현재 : 한세대학교 IT학부 교수

〈관심분야〉
정보보안, 사물인터넷, 클라우드, 환경시스템