논문 22-47-08-21 The Journal of Korean Institute of Communications and Information Sciences '22-08 Vol.47 No.08
https://doi.org/10.7840/kics.2022.47.8.1231

# 컨테이너 이미지 보안성 분석에 관한 연구: Docker Hub 이미지를 중심으로

유 명 성◆, 김 재 한*, 신 승 원°

# Revisiting Security Landscape of Docker Hub Container Images

Myoungsung You◆, Jaehan Kim*, Seungwon Shin°

요 약

경량 가상화 기술인 컨테이너가 클라우드 환경에서 활발히 사용되면서 개발자들이 다양한 컨테이너 이미지를 공유할 수 있게 해주는 레지스트리 서비스인 Docker Hub가 등장했다. Docker Hub는 서비스 개발자들에게 컨테이너를 활용한 효율적인 개발 환경을 제공하지만 취약한 컨테이너 이미지의 공유로 인한 전례 없는 심각한 보안 위협도 함께 가져왔다. 본 논문에서는 Docker Hub에서 공유되는 공식 및 커뮤니티 이미지의 취약성에 대해 상세히 조사한다. 이를 위해 자동으로 Docker Hub 컨테이너 이미지를 수집 및 분석하는 프레임워크를 설계했고 모든 공식 이미지와 가장 많이 다운로드된 1만 개의 커뮤니티 이미지를 분석하여 컨테이너 이미지 보안에 대한 다음과 같은 결론을 도출하였다. (1) 공식/커뮤니티 이미지 모두 평균 117개 이상의 보안 취약점을 가지고 있다. (2) 컨테이너 이미지 내의 취약점은 심각한 취약점일지라도 해당 취약점이 공개된 뒤 평균적으로 약 3일 이후에야 패치된다. (3) 컨테이너 이미지 간의 취약점 전파는 매우 빈번하지만 이에 대한 대응에는 큰 비용이 소모된다. 본 논문이 제공하는 결과는 향후 컨테이너 보안 연구에 유용하게 활용될 수 있을 것이라 기대한다.

**키워드 :** 클라우드 컴퓨팅, 정보보안 및 개인정보보호, 클라우드 보안, 가상화, 컨테이너 보안
**Key Words :** Cloud computing, Security and privacy, Cloud security, Virtualization, Container security

ABSTRACT

Containerization has recently become a de facto standard of virtualization technology in modern cloud environments. Its popularity has led to the development of various container engines and remote registry services. Docker Hub, the largest container registry, distributes numerous official and community container images. It provides agile ways to develop services using containers but at the same time poses new security threats by sharing vulnerable images. In this paper, we investigate the current state of vulnerabilities in container images shared on Docker Hub. We design an automated security assessment framework that discovers and examines container images on Docker Hub. We obtain the following insights by analyzing all the official images and the 10,000 most downloaded community images through our framework. (1) Both the official and community images have an average of 117 or more vulnerabilities. (2) Vulnerabilities in images are patched

※ 본 연구는 방위사업청과 국방과학연구소가 지원하는 미래전투체계 네트워크기술 특화연구센터 사업의 일환으로 수행되었습니다.(UD190033ED)
◆ First Author : KAIST School of Electrical Engineering, famous77@kaist.ac.kr, 학생회원
° Corresponding Author : KAIST School of Electrical Engineering, claude@kaist.ac.kr, 정회원
* KAIST School of Electrical Engineering, rlawogks145@kaist.ac.kr
논문번호 : 202205-084-0-SE, Received April 30, 2022; Revised May 14, 2022; Accepted May 14, 2022

on average three days after the vulnerabilities are disclosed. (3) Propagation of vulnerability between images is prevalent, but countermeasures against the propagation are costly. We believe that this paper will be utilized as a good foundation in future work on container security.

## Ⅰ. 서 론

운영체제 수준의 경량 가상화 기술인 컨테이너(container)[1]는 마이크로서비스의 신속한 배포와 효율적인 운영을 제공해 데이터 센터 및 클라우드 환경에서 가상화 기술의 사실상 표준(de facto standard)이 되었다. 기존 가상 머신과 달리 컨테이너는 응용프로그램 실행을 위한 격리된 런타임 환경을 제공하면서도 컨테이너화된 응용프로그램들이 호스트 시스템의 자원을 효율적으로 공유하도록 해준다. 이런 자원 공유 모델은 컨테이너를 가상 머신보다 더 빠르고 효율적으로 동작할 수 있게 했고, 이는 클라우드 환경에서 컨테이너의 대대적인 도입을 이끌었다.

컨테이너가 활발히 사용됨에 따라 이를 위한 다양한 기술이 파생되었다. Docker[2]는 대표적인 컨테이너 런타임 엔진으로 컨테이너에서 실행할 수 있는 응용프로그램을 이미지 형태로 배포한다. 각 이미지는 응용프로그램 실행을 위한 모든 구성 요소(파일시스템 , 라이브러리 등)를 포함하고 있다. Docker가 컨테이너로 응용프로그램을 배포하는 손쉬운 방법을 제공함에 따라 개발자들이 컨테이너 이미지를 효율적으로 공유할 수 있게 해주는 레지스트리 서비스인 Docker Hub[3]가 등장했다.

Docker Hub 내의 이미지는 공식(official) 이미지와 커뮤니티(community) 이미지로 나뉘며 커뮤니티 이미지는 공유 및 업로드에 제한이 없다. Docker Hub은 컨테이너를 활용한 효율적인 서비스 개발 환경을 제공하지만, 동시에 취약한 컨테이너의 공유로 인한 전례 없던 새로운 보안 위협도 함께 가져왔다. Docker Hub은 업로드되는 컨테이너 이미지에 대해 제한적인 취약성 검사 절차만을 제공하기 때문에[4] 공유되는 이미지 내부에 알려진 보안 취약점이 존재할 수 있으며 이에 대한 위협은 모두 해당 컨테이너를 내려 받아(pull) 사용하는 사용자에게 전가된다.

보안 전문 업체인 Tripwire 사가 300개의 IT 전문 기업을 대상으로 조사한 2019년도 컨테이너 보안 분석 보고서[5]에 따르면 94%의 기업이 컨테이너 사용에 있어 보안이 가장 큰 관심사라고 밝혔으며, 60% 기업이 컨테이너로 인한 보안 사고로 피해를 본 경험이 있다. 따라서 신뢰할 수 있는 컨테이너 환경 구축을 위해 Docker Hub 내 컨테이너 이미지에 대한 체계적인 보안 분석 연구가 필요하다.

본 논문에서는 Docker Hub 내에서 공유되는 공식 이미지 및 커뮤니티 이미지의 보안성을 다음과 같은 목표를 갖고 종합적으로 분석한다. (i) 컨테이너 이미지에 존재하는 잠재적인 보안 취약점을 도출하고 이들의 특성 및 분포를 분석한다. (ii) 컨테이너 이미지 공급자(vendor)가 새로운 보안 취약점에 얼마나 민첩하게 대응하는지 그 속도를 측정한다. (iii) 컨테이너 이미지 간의 보안 취약점의 전파 과정과 및 그 영향도를 분석한다.

이를 위해 본 논문에서는 Docker Hub 컨테이너 이미지를 자동으로 수집 분석하는 프레임워크를 설계했다. 본 논문에서는 더욱 명확한 분석을 위해 다운로드 횟수를 기준으로 1만 개의 컨테이너 이미지를 선정했고, 설계한 프레임워크로 해당 이미지들의 최신 버전을 수집 및 분석하여 다음과 같은 결론을 도출했다.

(i) 공식 이미지와 커뮤니티 이미지 모두 평균적으로 117개 이상의 알려진 취약점을 가지고 있다. (ii) 컨테이너 이미지 내부의 취약점은 심각한 취약점일지라도 해당 취약점이 공개된 뒤 평균적으로 3일 이후에 패치된다. (iii) 일부 컨테이너 이미지 내부의 취약점은 해당 이미지를 참조하는 많은 수의 다른 이미지에 심각한 영향을 줄 수 있다.

논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장에서는 논문의 내용을 이해하기 위한 배경지식과 컨테이너 이미지에 대한 잠재적인 보안 위협을 소개한다. 논문의 3장에서는 본 논문에서 설계한 Docker Hub 컨테이너 이미지 자동 분석 프레임워크의 구조와 동작 과정을 상세히 설명한다. 4장에서는 분석 프레임워크를 통해 도출한 결과를 보여준다. 마지막으로 5장에서 컨테이너 이미지 보안에 대한 기존 연구에 대해 소개한 뒤 6장에서 논문의 내용을 결론짓는다.

## Ⅱ. 관련 연구 및 배경

### 2.1 리눅스 컨테이너

리눅스 컨테이너는 운영체제 수준의 경량 가상화 기술로, 프로세스(응용프로그램) 실행을 위한 격리되고 통합된 환경을 제공한다. Fig. 1에 나타나 있는 것

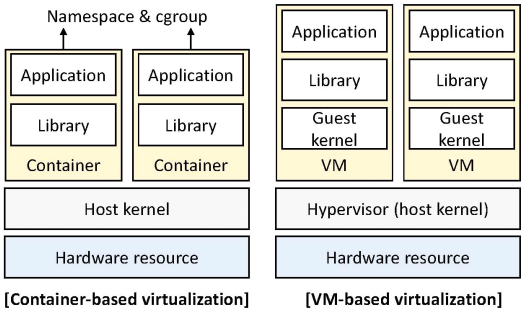



Fig. 1. The overview of the container/VM-based virtualization.

처럼 하나의 호스트 운영체제에서 실행되는 컨테이너들은 호스트 커널과 하드웨어 자원을 공유하면서 효율적이고 유연하게 동작한다[26]. 컨테이너 내부의 프로세스는 다른 컨테이너 환경의 존재를 알 수 없으므로 자신이 독립된 환경에서 실행되는 것으로 판단하게 된다. 이러한 프로세스 격리는 네임스페이스(namespace)와 컨트롤 그룹(control group) 기술을 통해 이루어진다[6]. 예를 들어 호스트 시스템은 컨테이너별로 독립된 네트워크 네임스페이스[7]를 생성하여 각 컨테이너가 독립된 네트워크 인터페이스를 사용할 수 있도록 해준다.

컨테이너는 기존 가상화 기술인 가상 머신(virtual machine)과 달리 가상화된 커널 및 하드웨어 자원을 구성하지 않으므로 응용프로그램을 더욱 빠르고 효율적으로 배포 및 운영이 가능하다[25]. 그러나 컨테이너는 본질적으로 커널의 기능으로 논리적 격리 환경을 구성한 것이기 때문에 가상 머신과 비교하여 상대적으로 약한 격리 수준을 가지며, 이는 심각한 보안 위협을 야기하게 되었다. 예컨대, 컨테이너 내부의 공격자는 컨테이너 내부에 악성코드 등을 포함해 컨테이너 환경뿐 아니라 컨테이너가 상주하는 호스트 시스템 전체에 심각한 보안 위협을 가할 수 있다.

### 2.2 Docker 컨테이너 런타임

Docker[2]는 대표적인 컨테이너 런타임으로 컨테이너화된 응용프로그램의 효율적인 실행 및 배포를 위한 다양한 기능을 제공한다. Docker는 컨테이너를 컨테이너 이미지의 형태로 배포한다. 각 컨테이너 이미지는 실행할 응용프로그램의 바이너리 파일뿐 아니라 이를 위한 라이브러리, 설정 파일 등 응용프로그램 실행을 위한 모든 구성 요소를 포함하고 있다.

개발자들은 새로운 컨테이너 이미지를 밑바닥부터 다시 만들지 않고, 기존에 있는 다른 이미지들을 확장하여 새 이미지를 적은 비용으로 개발할 수 있으며 이를 통해 컨테이너 이미지 간의 부모-자식 관계(parent-child relationship)가 형성된다. 컨테이너 이미지에 대한 의존성 트리의 최상위 계층(root)에는 응용프로그램 실행 환경을 제공하는 운영체제(예: Ubuntu)나 많은 응용프로그램에서 자주 사용되는 라이브러리(예: libc) 또는 서버 프로그램(예: Apache server)과 같은 베이스 이미지들이 위치하고 있다.

### 2.3 Docker Hub

Docker가 활발히 사용되면서 Docker 컨테이너 이미지의 효율적인 공유를 위한 레지스트리 서비스가 개발되었다. 대표적으로 2014년에 등장한 클라우드 레지스트리인 Docker Hub[3]가 있다. Docker Hub에서 컨테이너 이미지는 이미지의 개발 및 운영에 대한 형상 관리(versioning)가 가능한 레퍼지토리 형태로 공유된다.

Docker Hub에는 두 가지 종류의 레퍼지토리가 존재한다. 먼저 공식 레퍼지토리의 경우 인증된 제공사(예: Oracle, Microsoft, and RedHat 등)로부터 인계받은 컨테이너 이미지들이 속해 있다. 공식 레퍼지토리의 컨테이너 이미지는 대부분 베이스 이미지이기 때문에 Docker Hub에서 직접 관리한다. 본 논문을 작성하는 시점에서 공식 레퍼지토리는 약 170개가 존재한다[8]. 반면 커뮤니티 레퍼지토리는 별도의 제한 없이 모든 사용자가 이미지를 생성하고 배포할 수 있으며 그 리스트와 수는 공식적으로 공개되지 않고 있다[9].

### 2.4 Docker Hub 이미지 내의 보안 위협

Docker Hub에는 다양한 공급처 및 사용자로부터 업로드된 수많은 컨테이너 이미지가 존재하며, 이들은 다양한 서비스 구성을 위해 활발히 사용되고 있다. 일반적으로 컨테이너 이미지는 다수의 소프트웨어 패키지 및 실행 파일뿐만 아니라 다른 이미지에서 가져온 파일도 사용한다. 이러한 복잡성은 컨테이너 이미지의 생성 및 관리 측면에서 다음과 같은 보안 위협을 야기한다.

**(1) 컨테이너 내부의 취약점**. 컨테이너 이미지가 내부의 응용프로그램 및 소프트웨어 패키지에 알려진 취약점이 존재할 수 있다. 이 경우 공격자가 해당 취약점을 악용하여 컨테이너화된 서비스를 공격할 수 있다.

**(2) 민감한 컨테이너 실행 명령어**. 컨테이너 이미지는 설정 파일에 명시된 docker run 명령어[10]를 통

해 실행된다. docker run 명령어는 다양한 옵션을 사용하는데, 여기에는 컨테이너의 격리 수준 및 권한을 조절하는 옵션도 포함된다. 부주의한 옵션 사용으로 컨테이너에 낮은 격리 수준을 사용하거나 필요 이상의 권한을 부여하면, 공격자가 해당 컨테이너를 탈취할 경우보다 심각한 결과를 초래할 수 있다.

**(3) 컨테이너 간 취약점 전파**. 컨테이너 이미지는 계층 구조를 이루고 있기 때문에 부모 컨테이너 이미지에 존재하는 취약점이 자식 컨테이너에도 동일하게 존재할 수 있다. 즉 공격자는 특정 이미지의 취약점을 이용해 해당 이미지를 확장하여 사용하는 자식 이미지 들을 공격할 수 있다.

**(4) 취약점 패치 지연**. 컨테이너 이미지에서 사용되는 소프트웨어에 대한 취약점 패치가 공개되어도 해당 패치가 곧바로 컨테이너 이미지에 적용되지는 않는다. 이는 패치 내용에 맞게 컨테이너 이미지 내의 다른 설정 파일들도 새롭게 구성하고 이미지를 다시 빌드해야 하기 때문이다. 이러한 취약점 패치의 지연은 공격자가 해당 취약점을 악용할 기회를 제공한다.

위와 같은 보안 위협은 컨테이너를 활용하는 서비스와 시스템에 심각한 영향을 줄 수 있다. 예를 들어, 공격자는 컨테이너에서 실행되는 응용프로그램 대한 원격 코드 실행 취약점을 악용하여 서비스 데이터를 유출하거나 시스템을 마비시킬 수 있다[6]. 이러한 위협을 예방하기 위해 본 논문에서는 Docker Hub 내의 컨테이너 이미지를 대상으로 위에서 언급한 네 가지 보안 위협에 대해 면밀히 조사한다. 본 논문이 컨테이너 이미지 보안성 분석의 첫 번째 연구는 아니다. 하지만 기존 연구는 대부분 단일 이미지 내부에 존재하는 취약점을 찾고 이를 분석하는 것에 초점을 맞췄다. 이와 달리 본 논문은 단일 이미지의 보안성 분석은 물론 이미지 간의 취약점 전파에 대한 조사 및 취약점 패치 반영 시간까지 고려하여 컨테이너 이미지 생태의 전반적인 보안성을 분석한다.

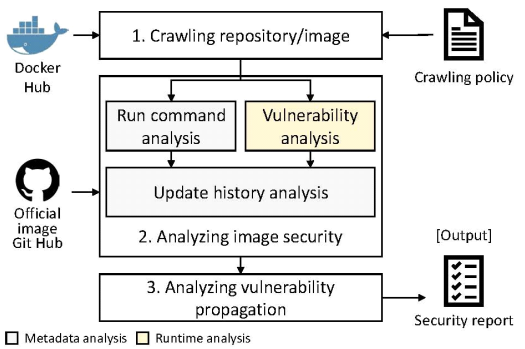



Fig. 2. The overall work flow of container image security analysis framework.

## Ⅲ. 연구 방법

### 3.1 분석 프레임워크 개요

본 논문에서는 Docker Hub에서 수집한 컨테이너 이미지를 2.4절에서 언급한 네 가지 보안 위협을 중점으로 분석한다. 이를 위해 본 논문에서는 Fig. 2에 설명된 것과 같은 자동화된 컨테이너 이미지 보안 분석 프레임워크를 설계하였다. 설계한 분석 프레임워크는 다음 세 가지 단계를 통해 컨테이너 이미지를 분석한다: (i) 데이터 수집, (ii) 보안성 분석, (iii) 취약성 전파 분석.

데이터 수집 과정에서 분석 프레임워크는 Docker Hub에 업로드된 컨테이너 레퍼지토리의 웹 사이트와 컨테이너 이미지를 수집한다. 이때 관리자는 프레임워크가 수집할 컨테이너 이미지에 대한 구체적인 수집 정책(이미지 pull 횟수, 이미지의 카테고리 등)을 지정하여 수집 과정의 효율을 높일 수 있다. 데이터 수집 과정이 끝난 뒤, 분석 프레임워크는 수집한 레퍼지토리와 이미지에 대해 각각 아래와 같은 보안 분석을 진행한다. 분석 프레임워크는 먼저 레퍼지토리에서 컨테이너 이미지 실행 명령어(docker run)[10]를 추출하고 명령어에 민감한 옵션을 사용했는지 여부를 검사한다. 이와 별개로 해당 레퍼지토리의 업데이트 기록을 분석하여 보안 취약점에 대해 얼마나 빠르게 대응하고 있는지 그 대응 시간을 도출한다.

컨테이너 이미지에 대한 보안 분석으로 분석 프레임워크는 수집한 컨테이너 이미지를 직접 실행시킨 뒤 알려진 취약점이 존재하는지 검사한다.

다음으로 분석 프레임워크는 각 컨테이너 이미지가 사용하는 베이스 이미지를 구하고 취약점 분석 결과와 결합하여 컨테이너 이미지 사이의 취약성 전파 관계를 분석한다. 구체적으로 특정 이미지의 보안 취약점이 다른 컨테이너에 얼마나 많은 영향을 줄 수 있는지 등 컨테이너 이미지 간의 보안성 관계를 측정한다.

Table 1. Collections of Docker container images.

| | # of images | # of vendors |
|---|---|---|
| Official image | 161 | 1 (Docker Hub) |
| Community image | 10,000 (most popular) | 8,405 |
| Total | 10,161 | 8,406 |

마지막으로 분석 프레임워크는 분석한 결과를 종합하여 수집한 컨테이너 이미지에 대한 보안 분석 보고서를 생성한다.

### 3.2 데이터 수집 및 전처리

컨테이너 이미지를 분석을 위해 Docker Hub를 이용하여 다양한 컨테이너 이미지를 수집하였다. Docker Hub가 제공하는 이미지는 공식(official) 이미지와 커뮤니티(community) 이미지로 분류되어 있으며, 이미지 명칭 리스트를 제공하는 형태에 차이가 있어 각기 다른 방법을 통해 데이터 수집을 진행하였다. 본 논문에서는 인증된 사용자(verified publisher)의 이미지는 커뮤니티 이미지로 분류하였다. 컨테이너 이미지 명칭과 관련 정보 수집을 위해 Docker Hub API를 이용하였다.

공식 컨테이너 이미지의 경우 도커 허브에서 이미지 명칭 리스트를 제공한다. 또한 해당 이미지에 대한 레퍼지토리가 저장된 GitHub의 Docker 자사의 레퍼지토리에서도 리스트를 얻을 수 있다. 본 수집 과정에서는 GitHub Docker 라이브러리의 리스트를 바탕으로 174개의 공식 이미지 명칭 리스트를 구성하고 docker pull 명령어를 통해 해당 명칭을 가진 공식 이미지를 수집하였다. 이후 전처리 과정을 통해 latest 태그를 가지고 있지 않은 이미지를 제외하여 총 161개의 공식 이미지를 구성하였다. 이와 달리 커뮤니티 컨테이너 이미지는 Docker Hub에서 이미지 리스트를 공개하지 않아 키워드 기반 검색을 통해서 해당 이미지의 명칭과 정보를 파악할 수 있다. Docker Hub은 길이가 1인 문자열에 대한 검색을 허용하지 않기 때문에, 최대한 다양한 커뮤니티 이미지 수집을 위해 숫자, 알파벳, ‘ ’, ‘-’ 문자들 무작위로 조합해 검색을 위한 길이 2의 문자열을 생성하였다. 이를 통해 검색한 이미지 중 다음과 같은 전처리 과정을 거쳐 분석을 위한 데이터를 선별했다. 먼저, 삭제되었거나 만료된 다수의 이미지와 latest(최신 버전) 태그를 가지고 있지 않은 이미지를 제외하였다.

그리고 분석의 효율성을 높이기 위해 사용자들에게 영향을 미칠 가능성이 높은(pull 횟수가 많은) 1만 개의 이미지를 분석에 활용하였다. 전처리 전후의 데이터 수집 현황은 Table 1과 같다. Docker Hub API는 컨테이너 이미지 파일 이외에도 이미지에 대한 메타데이터를 제공한다. 다양한 보안성 분석을 위해 전처리 이후의 공식 및 커뮤니티 컨테이너 이미지의 생성 및 최근 업데이트날짜, pull 횟수, 설명문서 등의 메타데이터를 추가로 수집하였다.

### 3.3 이미지 보안성 분석

컨테이너 이미지에 대한 종합적인 보안성 분석을 위해 분석 프레임워크는 수집한 이미지를 대상으로 다음 세 가지 분석을 진행한다.

**민감한 명령어 옵션 분석.** 일반적으로 레퍼지토리의 웹 페이지는 해당 컨테이너 이미지를 처음 사용하는 사용자를 위해 컨테이너 실행을 위한 docker run 예제 명령어를 제공한다. 분석 프레임워크는 수집한 레퍼지토리 페이지에서 docker run 명령어를 추출한 뒤 보안에 민감한 옵션을 포함하고 있는지 검사한다. 컨테이너 이미지의 실행을 위한 docker run 명령어에는 약 100여 개의 옵션을 지정하여 사용할 수 있다[10]. 유감스럽게도 명령어에 사용되는 옵션 중 어떤 것이 악의적으로 사용될 수 있는지는 공개된 자료는 존재하지 않는다. 이 문제를 해결하기 위해 본 논문에서는 docker run 명령어의 옵션을 모두 분석했고 다음과 같은 기준을 통해 악의적으로 사용될 수 있는 민감한 명령어 옵션 목록을 도출했다.

(i) 컨테이너의 기본 원리는 네임스페이스를 통해 컨테이너의 자원(파일시스템, 네트워크 인터페이스 등)을 호스트로부터 논리적으로 격리하는 것이다. 따라서 본 논문에서는 컨테이너가 호스트의 자원에 접근할 수 있게 해주는 옵션을 민감한 옵션으로 정의했다. 대표적으로 컨테이너에서 호스트 장치에 접근할 수 있게 해주는 옵션(device="path")이나, 호스트 시스템과 특정 네임스페이스를 공유할 수 있게 해주는 옵션(network="host")이 포함된다.

(ii) 보안을 위해 컨테이너는 일반적으로 관리자 권한으로 실행되지 않는다. 이는 관리자 권한으로 실행된 컨테이너가 탈취될 경우 시스템에 심각한 영향을 줄 수 있기 때문이다. 본 논문에서는 컨테이너에 관리자 권한(혹은 관리자 권한의 일부분)을 부여하는 옵션을 민감한 옵션으로 정의했다. 예로, 컨테이너에 기본적으로 없는 특정 권한을 부여하는 옵션(add-cap="NET ADMIN")이 포함된다.

**컨테이너 내부 취약점 분석.** 컨테이너 이미지에 포함된 응용프로그램, 소프트웨어 패키지, 라이브러리 등에 알려진 취약점이 존재할 경우 공격자는 이를 악용해 컨테이너를 손쉽게 탈취할 수 있다. 본 논문에서는 소프트웨어 보안 취약점에 대한 자세한 정보를 제공하는 Common Vulnerabilities and Exposures (CVE) ID[11]를 기준으로 컨테이너 이미지에 존재하는 보안 취약점을 식별하고 분석했다. 이를 위해 분석 프레임워크는 내부적으로 Snky[12]를 활용하였다. Snyk는 Docker Hub에서 컨테이너 이미지 보안성 검

증을 위해 사용되는 개발자 보안 플랫폼으로, CVE ID를 기준으로 컨테이너 이미지에서 사용되는 소프트웨어 패키지와 라이브러리를 분석하여 알려진 취약점을 찾아준다. 분석 프레임워크는 Synk를 활용하여 각 컨테이너 이미지에 존재하는 취약점의 CVE ID 및 심각도, 취약점의 분류(버퍼 오버플로우 등)를 수집한다. 본 논문에서는 커뮤니티 이미지뿐 아니라 대부분의 공식 이미지에서도 다수의 알려진 취약점을 식별했다.

**취약점 노출 기간 분석.** 컨테이너 이미지에서 식별한 보안 취약점의 라이프 사이클을 분석하기 위해 본 논문에서는 (1) 취약점 공개 시기, (2) 이미지 업데이트 시기를 비교 분석하였다. 취약점 공개 시기는 컨테이너 이미지 내부에 존재하는 취약점이 공개된 시점으로 취약점에 대해 CVE ID가 부여된 시기를 의미한다[13]. 이미지 업데이트 시기는 컨테이너 이미지 개발자가 해당 취약점에 대해 소프트웨어 벤더가 제공하는 보안 패치를 컨테이너 이미지에 반영한 시점을 말한다. 본 논문에서는 취약점 공개 시기와 이미지 업데이트 시기 간의 차이를 분석하여 컨테이너 이미지가 공격에 무방비로 노출되어 있는 기간인 취약점 노출 기간을 산출한다. 본 논문에서 취약점 노출 기간 분석은 공식 이미지를 대상으로만 진행한다. 이는 공식 이미지는 Docker Hub에서 직접 관리하기 때문에 이미지 업데이트 기록을 공개하고 있지만, 커뮤니티 이미지의 업데이트 기록이 공개되어 있지 않아 분석이 사실상 불가능하기 때문이다.

Docker Hub은 공식 이미지에 대한 업데이트 기록을 모두 자사 GitHub[8]에 커밋(commit)하여 유지하고 있다. 분석 프레임워크는 Docker GitHub의 커밋 로그를 수집하여 공식 이미지에 대한 업데이트 기록을 모두 수집한다. 이중 보안 취약점 패치에 대한 커밋 로그는 CVE ID를 태그로 포함하고 있다. 이를 이용해 분석 프레임워크는 수집한 업데이트 기록에서 보안 취약점 목록 및 해당 취약점에 대한 패치 시기를 구한다. 이후 수집한 취약점 목록을 바탕으로 CVE 데이터베이스[13]를 조회하여 취약점 공개시기를 구한 뒤 최종적으로 각 취약점에 대한 취약점 노출 기간을 산출한다.

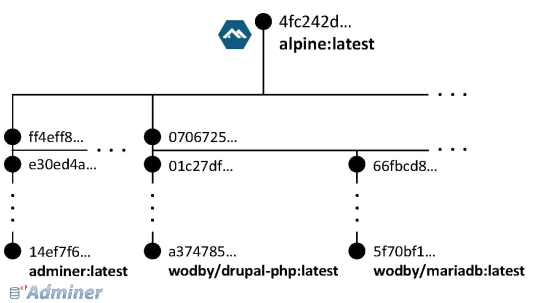



Fig. 3. An example of dependency analysis.

### 3.4 이미지 간 취약성 전파 분석

Docker 컨테이너는 기존 이미지를 베이스(base image)로 택하여 새로운 계층을 쌓아 다른 컨테이너 이미지를 생성할 수 있는 구조를 제공한다. 이는 새로운 이미지 개발에 소모되는 비용을 최소화해주지만 기존 이미지 내부의 보안 취약점이 새로운 이미지로 전파될 수 있는 위험성을 내포하고 있다. 본 논문에서는 심층적인 보안성 분석을 위해 이와 같은 이미지 간 의존성을 파악하는 방법을 제시한다.

먼저, docker inspect 명령어를 통해 모든 컨테이너 이미지의 각 계층 이미지 정보를 얻어낸다. 각 컨테이너 이미지에 대해 계층 이미지들의 고유 식별자인 digest ID를 이용하여 이미지의 부모-자식 관계를 반영한 일자 모양의 트리를 생성한다. 이때 각 트리의 최상위 계층에는 해당 이미지가 사용하는 베이스 이미지가 위치하게 된다. 이후 모든 컨테이너 이미지의 트리 중 두 트리를 선택하여 digest ID의 가장 긴 매칭(longest matching)을 가지고 있는 서브 트리(subtree)를 병합하고 매치되지 않은 다른 자식 노드들을 이어 붙인다. 위 과정을 반복하여 트리를 병합하면 특정 베이스 이미지를 최상위 계층으로 하고 이를 참조하는 컨테이너 이미지들을 노드로 가지는 의존성 트리를 최종적으로 얻게 된다. Fig. 3은 Alpine Linux[14] 이미지를 베이스 이미지로 하는 컨테이너 이미지들에 대한 의존성 분석 예시이다.

베이스 이미지의 취약점이 해당 이미지를 사용하는 커뮤니티 이미지에 전파되는 것을 분석하기 위해, 생성된 이미지 의존성 트리와 GitHub의 Docker 레퍼지토리 커밋 기록을 활용하였다. 각 의존성 트리의 최상위 계층이 가지고 있는 digest ID를 레퍼지토리의 과거 정보와 매칭하여 특정 컨테이너 이미지가 사용한 베이스 이미지의 정보를 조사하였다. 해당 베이스 이미지가 공식 이미지가 아닌 경우 위의 방법이 제한되므로, 분석 데이터 내 이미지의 digest ID와 매치하여 이미지의 정보를 파악하였다. 이미지 취약점 식별을 위해 Synk의 보안 테스트 결과를 바탕으로 이미지의 태그 혹은 특정 소프트웨어 버전에 따라 이미지가 가지고 있는 취약점을 CVE ID 형식으로 추출하고 해당 베이스 이미지를 사용하는 이미지를 분류하였다.

Table 3. Top-10 vulnerabilities and affected images.

| Official images | | | Community images | | |
|---|---|---|---|---|---|
| Vulnerability | Severity | # of images | Vulnerability | Severity | # of images |
| CVE-2019-20838* | High | 112 | CVE-2018-25032 | High | 5,013 |
| CVE-2017-11164* | High | 111 | CVE-2022-1271* | High | 4,757 |
| CVE-2016-2781* | Medium | 111 | CVE-2016-2781* | Medium | 4,685 |
| CVE-2013-4235* | Medium | 110 | CVE-2017-11164* | High | 4,667 |
| CVE-2022-1304* | High | 107 | CVE-2022-0778 | High | 4,482 |
| CVE-2022-1271* | High | 106 | CVE-2022-1304* | High | 4,405 |
| CVE-2020-16156 | High | 104 | CVE-2020-14155 | Medium | 4,348 |
| CVE-2021-3608 | Low | 103 | CVE-2019-18276 | High | 4,259 |
| CVE-2018-5709 | High | 97 | CVE-2019-20838* | High | 4,199 |
| CVE-2005-2541 | Critical | 90 | CVE-2013-4235* | High | 4,152 |

## Ⅳ. 연구 결과

본 장에서는 설계한 분석 프레임워크를 사용해 수집한 컨테이너 이미지를 대상으로 진행한 실험 결과와 이를 통해 도출한 시사점에 대해 서술한다. 본 논문에서는 실험을 위해 Docker Hub 내의 모든 공식 이미지와 가장 많이 다운로드 된 1만 개의 커뮤니티 이미지를 수집했다. 명확한 결과를 위해 소프트웨어 버전별 취약점 식별을 제외한 모든 분석에서 컨테이너 이미지는 해당 레퍼지토리의 최신 버전(latest)만을 대상으로 하였다.

Table 2. The statistics of vulnerabilities.

| Image | Mean | Std. | Median | Max | Min |
|---|---|---|---|---|---|
| Official | 117.3 | 192.4 | 56 | 1,360 | 0 |
| Community | 206.4 | 331.1 | 54 | 3,140 | 0 |

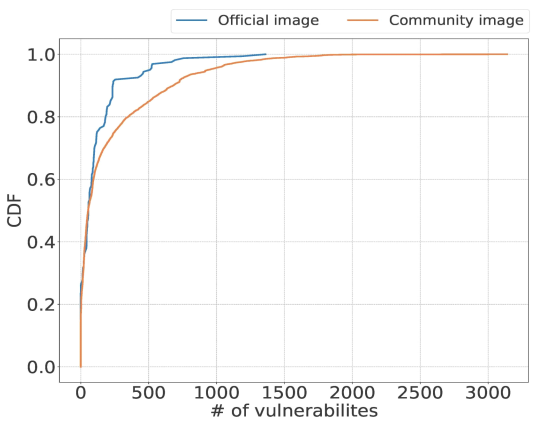



Fig. 4. CDF of the number of vulnerabilities in a image.

### 4.1 이미지별 취약점 분포

이미지별 보안 취약점의 수에 대한 분포는 Docker Hub에 내재된 보안 취약점의 전반적인 특징을 나타낸다. 본 논문에서 진행한 실험 결과 공식 이미지에서 2,855개, 커뮤니티 이미지에서 13,033개의 취약점을 식별하였다. 공식 이미지의 취약점 중 거의 대부분(2,835개)은 커뮤니티 이미지에서도 발견되었는데, 이는 다수의 커뮤니티 이미지가 공식 이미지를 베이스 이미지로 사용하기 때문으로 추측된다.

Table 2는 공식/커뮤니티 이미지별 취약점의 수와 이에 대한 평균, 중간값, 최대, 최소 및 표준편차를 보여주며 Fig. 4는 이미지별 취약점 수의 누적 분포 함수를 나타낸다. 전체적으로 각 공급자에 의해 자율적으로 관리되는 커뮤니티 이미지가 Docker Hub에서 직접 관리하는 공식 이미지보다 많은 수의 취약점을 가지고 있다. 하지만 중간값을 고려했을 때 54개와 56개로 두 이미지 종류 간의 큰 차이가 없으며, 공식 이미지 내에도 평균적으로 117개의 취약점이 존재한다. 이는 공식 이미지를 사용할 때도 취약점에 대한 주의가 필요함을 의미한다.

### 4.2 이미지별 취약점의 심각도 및 종류

컨테이너 이미지에서 발견된 보안 취약점의 심각도를 분석하기 위해 본 논문에서는 NVD에서 제공하는 CVE에 대한 다섯 단계의 심각도 지표[15]를 활용했다: “None”, “Low”, “Medium”, “High”, “Critical”. Fig. 5는 이미지별로 발견된 취약점의 심각도 분포를 보여준다. 심각도가 Critical에 해당하는 취약점은 CVSS 점수 9.0 이상의 취약점으로, 공격자가 해당 취약점을

익스플로잇할 경우 시스템의 관리자 권한에 버금가는 권한을 얻거나, 취약점을 익스플로잇하기 위해 별도의 인증 절차나 추가 정보가 필요하지 않은 취약점이다. 커뮤니티 이미지의 경우 약 60%의 이미지가 Critical 심각도의 취약점을 한 개 이상 보유하고 있다. 이는 대부분의 커뮤니티 이미지가 심각한 보안 취약점을 가지고 있다는 것을 의미한다. 더욱 심각한 것은 공식 이미지 역시 약 20%가량의 이미지가 Critical 심각도에 해당하는 취약점을 가지고 있다는 점이다. 즉 Docker Hub에서 직접 관리하는 이미지라고 해서 무분별하게 사용할 경우 심각한 결과를 초래할 수 있음을 의미한다.

본 논문에서는 취약점 심각도와 더불어 최빈출 취약점에 대해서도 조사했다. Table 3은 공식/커뮤니티 이미지에서 발견된 10개의 최빈출 취약점을 보여준다. 공식 이미 와 커뮤니티 이미지 모두에서 최빈출 취약점들은 대부분 High 등급의 심각도를 가지고 있다. High 심각도의 취약점은 일반적으로 익스플로잇하기 어렵지만 취약점 공격할 경우 계정 권한 상승 등이 가능한 위험한 취약점이다. 흥미로운 것은 공식/커뮤니티 이미지별 최빈 취약점 10개 중 6가지가 동일하다는 점이다. 이는 앞서 언급한 것처럼 대부분의 커뮤니티 이미지가 공식 이미지를 상속받아 생성되어 공식 이미지의 취약점이 그대로 전파될 수 있다는 것을 의미한다. 공식 이미지에서 발견된 최빈 취약점 중 2005년이나 2013년에 발견된 상당히 오래된 취약점도 존재한다. 특히 CVE-2005-2541 [16]의 경우 CVSS 점수 10.0(최고점)으로 시스템 전체에 영향을 주는 매우 심각한 취약점이다. 공개된 지 16년이 넘은 취약점이 아직도 공식 이미지에서 다수 발견되고 있다는 점은 컨테이너 이미지에 대한 보안 패치가 잘 이루어지고 있지 않다는 사실을 의미한다.

### 4.3 이미지별 민감한 명령어 옵션 분포

Table 4는 컨테이너 각 이미지 레퍼지토리에서 발견된 민감한 명령어 옵션의 분포를 보여준다. 전체 이미지 중 약 1%의 레퍼지토리가 민감한 옵션이 포함된 docker run 예제 명령어를 제공하고 있다. 흥미로운 점은 약 50개의 커뮤니티 이미지의 레퍼지토리에서 docker run 명령어에 컨테이너를 시스템 관리자 권한으로 실행시키는 privileged 옵션을 사용하고 있다. 이 옵션을 통해 실행되는 컨테이너가 탈취될 경우 시스템 관리자 계정이 탈취된 것과 동일하기 때문에 레퍼지토리에서 제공하는 docker run 예제 명령어를 사용할 때는 사용자의 각별한 주의가 필요하다.

Table 4. Distribution of sensitive docker run command arguments in Docker Hub container images; *Including the "docker in docker" container image.

| Category | Example | Official | Community |
|---|---|---|---|
| PID isolation violation | pid="host" | 0 | 0 |
| UTS isolation violation | uts="host" | 0 | 0 |
| IPC isolation violation | ipc="host" | 0 | 2 |
| NET isolation violation | network= "host" | 2* | 13 |
| Sensitive capability | cap-add= "NET RAW" | 3 | 21 |
| Privileged | privileged | 1* | 50 |
| Host device access | device="<path>" | 3 | 23 |

### 4.4 이미지별 취약점 노출 기간

Docker Hub은 161개의 공식 이미지에 대한 업데이트가 있을 때마다 해당 내용을 자사 GitHub 레퍼지토리[8,17]에 커밋하여 기록하고 있다. 본 논문에서는 공식 이미지에 대한 취약점 노출 기간을 산출하기 전에 먼저 각 이미지가 얼마나 자주 관리되고 있는지 조사했다. 분석 프레임워크로 수집한 80,964개의 커밋 로그를 분석하여 각 공식 이미지별 업데이트 주기를 산출했고 Fig. 6과 7에 정리하였다. 분석 결과 공식 이미지는 평균적으로 약 10일을 주기로 업데이트되고 있지만 심한 경우 100일 이상 업데이트가 이루어지지 않고 있는 이미지도 존재한다. 반면, 다른 이미지에서 많이 참조되는 이미지의 경우 대부분 평균 미만의 업데이트 주기로 관리되고 있다. 이는 컨테이너 이미지

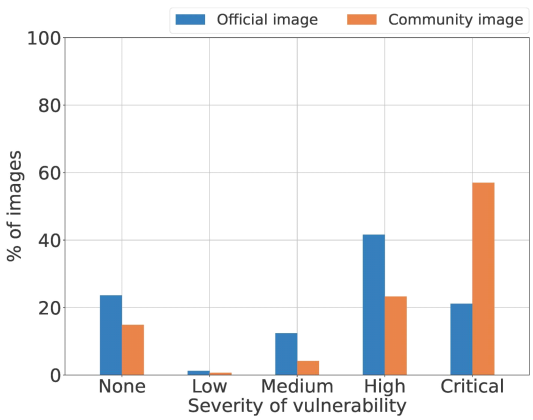



Fig. 5. Percentage of images inspected by the most severe vulnerability.

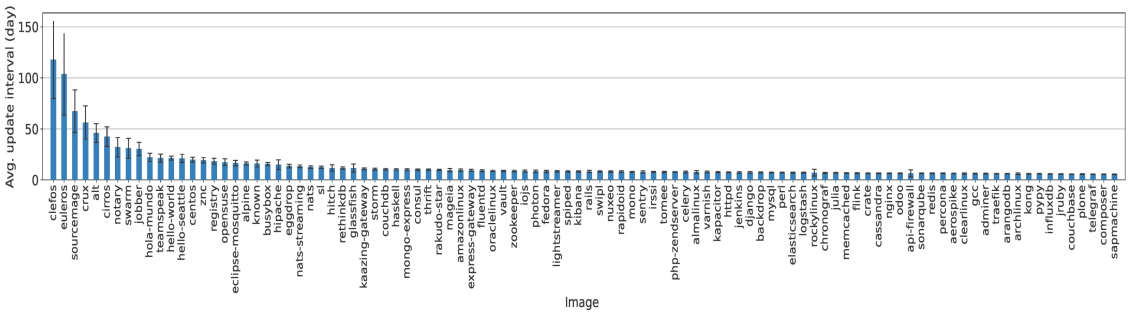



Fig. 6. Update intervals of Top-10 official images sorted by updated interval; The error bar denotes the standard errors.

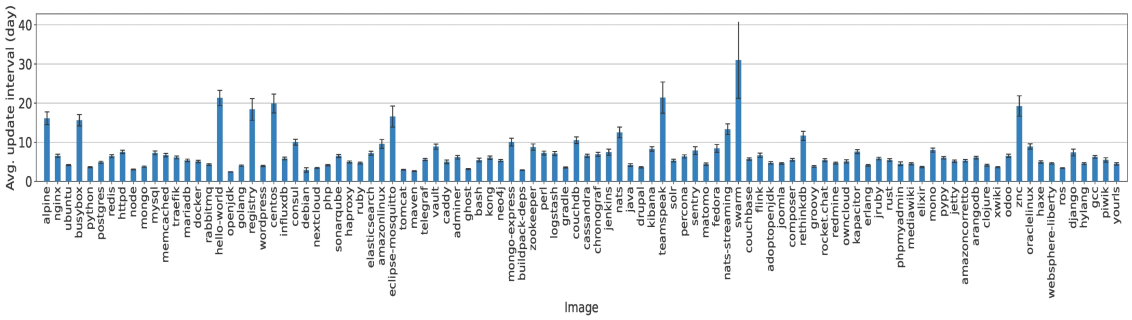



Fig. 7. Update intervals of Top-10 official images sorted by pull (download) count; The error bar denotes the standard errors.

의 업데이트 주기는 해당 컨테이너 내부에 존재하는 응용프로그램의 업데이트 주기도 일부 영향을 받지만, 해당 이미지의 사용량 역시 무시할 수 없는 영향을 준다는 것을 나타낸다.

다음으로 Docker Hub의 공식 이미지에 대한 취약점 노출 기간을 측정했다. 구체적으로 수집한 커밋 로그에서 CVE에 대한 태그를 포함한 커밋 로그를 추출한 뒤 해당 취약점의 공개 시기와 커밋 일시를 비교했다. 본 논문에서는 179개의 취약점에 대한 총 365개 취약점 패치 커밋 로그를 추출했고 이를 분석하여 Fig. 8과 같은 결과를 도출했다. 분석 결과 취약점에 대한 업데이트 속도는 취약점의 심각도와 높은 연관성을 보이는 것으로 드러났다. 공식 이미지들은 심각도가 Critical 등급인 취약점에 대해서는 평균 약 3일의 취약점 노출 기간을 갖지만 Low 등급의 취약점에는 평균 약 17일의 취약점 노출 기간을 보였다. 단편적으로 생각하면 Critical 심각도에 해당하는 취약점은 다른 취약점에 비해 비교적 빠르게 패치가 이루어진다. 하지만 공격자가 공개된 취약점을 이용해 컨테이너 이미지를 공격할 수 있는 기간이 평균적으로 3일이나 된다는 것은 컨테이너 이미지의 관리 측면에서 취약점에 대한 지속적인 모니터링과 대응이 필요하다는 것을 의미한다.

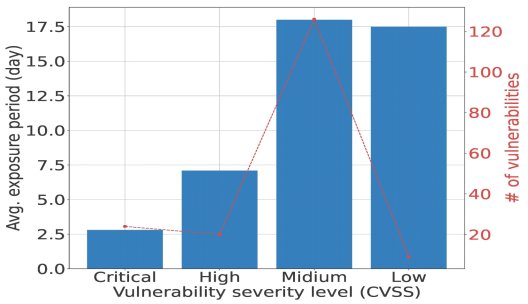



Fig. 8. Period of vulnerability exposure of official images on Docker Hub.

### 4.5 이미지간 취약점 전파 관계

컨테이너 이미지의 계층(layer) 구조는 상위 계층을 구성하는 컨테이너의 취약점이 하위 계층의 컨테이너들로 전파될 수 있는 환경을 형성하였다. 본 논문에서는 컨테이너 이미지 간의 취약점 전파 관계를 분석하기 위해 먼저 수집한 모든 컨테이너를 대상으로 의존성 분석을 진행하였다. 이 결과 가장 상위 계층에 존재하는 이미지, 즉 다른 이미지에서 가장 많이 참조되는 이미지는 Alpine Linux, Ubuntu 등 운영체제와 관련된 이미지였다. 이는 운영체제 이미지가 응용프로그

램 실행을 위한 가장 기본적인 환경을 제공한다는 점을 고려할 때 예상할 수 있는 결과였다. 따라서 본 논문에서는 운영체제 이미지 내부의 취약점이 이를 상속받는 하위 계층 컨테이너에 어떻게 전파되는지 조사했다.

Fig. 9는 각 운영체제 이미지의 버전별로 이를 참조하고 있는 이미지의 수와 내부의 취약점의 수를 보여주는 그래프이다. 본 논문에서 조사한 결과 운영체제 이미지는 평균적으로 466개의 이미지에서 참조되기 때문에 다른 컨테이너에 미치는 영향력이 상당히 높다. 하위 이미지에 가장 많은 영향을 끼치는 이미지는 Alpine Linux[14]로 1,213번 참조되고 있다.

이처럼 하위 이미지에 미치는 영향력이 크기 때문에 대부분의 운영체제 이미지는 공식 이미지로 지정되어 Docker Hub에서 직접 관리하고 있다. 하지만 본 논문에서 진행한 실험 결과 최신 버전 운영체제 이미지에도 평균적으로 56.2개의 보안 취약점이 존재한다. 가장 많은 취약점은 가진 최신 버전 이미지는 CentOS로 211개의 취약점을 가지고 있으며 이 중 25개의 취약점이 심각도 High 이상이다. 취약점의 수는 구 버전에서 최신 버전으로 갈수록 그 수가 감소하는 경향을 보이지만 일부 운영체제 이미지의 경우 구 버전이 최신버전보다 더 많이 참조되고 있기 때문에 운영체제 이미지의 취약점은 심각한 위협으로 다가올 수 있다.

다른 이미지에서 많이 참조되는 운영체제 이미지가 대부분 공식 이미지라는 점과 공식 이미지의 취약점이 모두 커뮤니티 이미지에서 발견된 점을 미루어 볼 때 우리는 상위 이미지에서 발견된 취약점이 이를 참조하는 이미지에서도 그대로 발견된다는 것을 유추할 수 있다. 더욱 심각한 것은 상위 이미지에 취약점에 대한 패치가 이루어진 다음에도, 해당 이미지를 상속받아 생성한 모든 하위 이미지를 새롭게 빌드해야 한다는 것이다. 이러한 특징은 컨테이너 이미지 간의 취약점 전파는 상위 이미지에서 하위 이미지로 쉽게 진행되지만, 해당 취약점을 모두 제거하기 위해서는 큰 비용이 소모된다는 것을 의미한다.

## V. 관련 연구

### 5.1 취약한 컨테이너 이미지 탐지

컨테이너가 다양한 서비스에 활발히 사용되면서 컨테이너 이미지 내에 존재하는 알려진 취약점을 찾는 연구가 다수 등장했다[9,18,4]. 대표적으로 Shu 등은[9] Docker Hub에서 컨테이너 이미지를 자동으로 수집하고 이들 내부에 존재하는 취약점을 분석하는 Docker Image Vulnerability Analysis (DIVA) 프레임워크를 제안했다. 하지만 해당 연구는 취약점의 존재 여부를 판단하는 것에만 초점을 맞추고 있다. 이와 달리 본 논문은 단순한 취약점의 유무뿐 아니라 해당 취약점이 패치되기까지 걸린 시간 등 컨테이너 이미지 생태

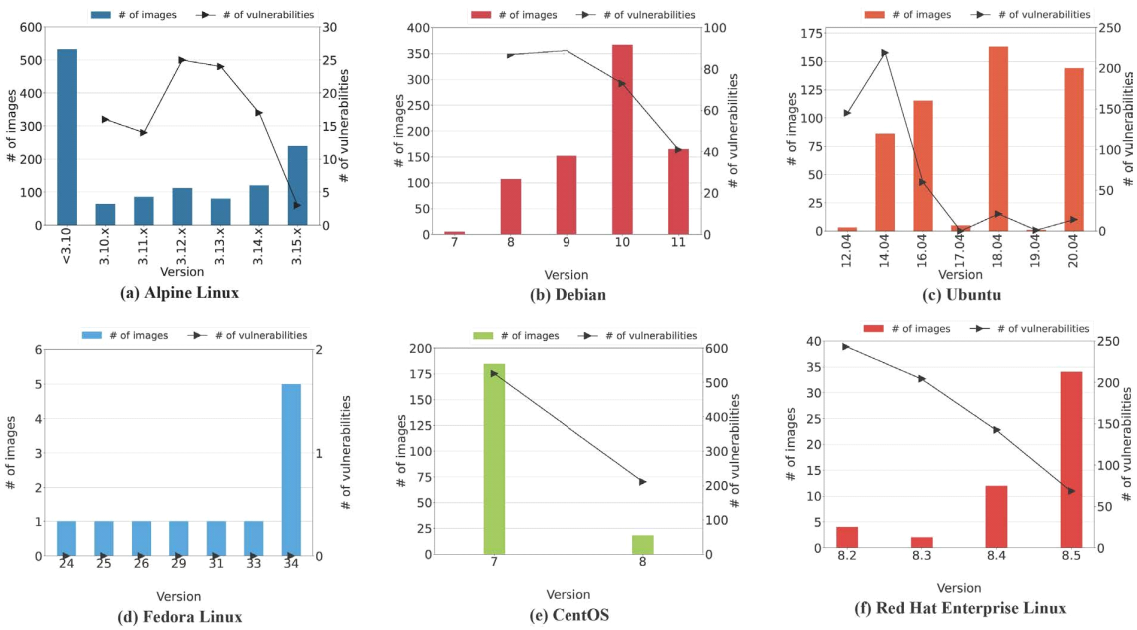



Fig. 9. Reference counts and vulnerabilities distribution of OS container images.

계의 전반적인 보안성을 분석한다.

### 5.2 악성 컨테이너 이미지 탐지

Docker Hub 등의 공개 레퍼지토리를 통해 다양한 공급자 및 사용자가 업로드한 컨테이너 이미지가 활발히 공유되면서, 백도어, 악성코드 등을 포함한 악의적인 컨테이너 이미지를 탐지하는 연구도 활발히 진행되고 있다[19, 20]. Kelly 등은[19] 소프트웨어 개발 주기에 따라 컨테이너 이미지에 대한 전반적인 보안성을 검사하는 프레임워크를 제안했다. 이들은 지속적 통합/지속적 제공의 파이프라인마다 컨테이너 이미지 내부의 알려진 취약점뿐만 아니라, VirusTotal[21] 등 동적 분석 도구를 활용해 백도어 등의 악성 파일도 함께 탐지한다. 하지만 이러한 연구는 소프트웨어 개발 주기에 따른 개별 컨테이너의 보안성을 분석할 뿐 컨테이너 이미지 간의 취약점 전파 등 컨테이너 에코 시스템의 보안성을 파악하지 못했다.

### 5.3 컨테이너 보안성 강화

컨테이너를 대상으로 하는 다양한 보안 위협이 등장하면서 런타임 환경에서 컨테이너의 보안성을 강화하기 위한 보안 솔루션들이 개발되었다[22, 23, 24]. SPEAKER[23]는 동적 분석을 사용해 컨테이너를 샌드박싱하는 대표적인 솔루션이다. 이 시스템은 자동화된 동적 분석으로 컨테이너가 호출하는 시스템 콜을 모니터링한 뒤 이를 바탕으로 시스템 콜 호출을 제한하는 정책 파일을 생성한다. 동적 분석과 달리 컨테이너 이미지 내부의 응용프로그램 소스 코드를 분석하는 정적분석 솔루션도 존재한다. CONFINE[22]는 정적분석을 사용하는 최신 솔루션으로 컨테이너화된 응용프로그램의 바이너리를 분석하여 동작에 필요한 시스템 콜을 추출하고 이를 바탕으로 정책 파일을 생성한다. 생성된 정책 파일은 컨테이너가 허용된 시스템 콜만 호출하도록 샌드박싱해 공격자가 컨테이너 내부에서 악성코드를 실행하는 것을 어렵게 만든다. 이러한 연구는 본 논문과 추구하는 바가 다르며 안전한 컨테이너 시스템 구축을 위해 함께 활용될 수 있다.

## Ⅵ. 결 론

컨테이너 기술이 클라우드 환경에서 활발히 사용되면서 다양한 컨테이너 이미지를 공유하기 위한 레지스트리 서비스인 Docker Hub가 등장했다. Docker Hub의 등장은 컨테이너 기반의 효율적인 서비스 개발 환경뿐 아니라 취약한 컨테이너 이미지의 공유로 인한 심각한 보안 위협도 함께 가져왔다. 본 논문에서는 Docker Hub 내 컨테이너 이미지에 대한 종합적인 취약성 분석을 진행했고 이를 위해 Docker Hub의 컨테이너 이미지를 자동으로 수집하고 분석하는 프레임워크를 설계했다. 본 논문에서는 설계한 프레임워크를 통해 Docker Hub 내 모든 공식 이미지와 가장 많이 다운로드 된 1만 개의 커뮤니티 이미지를 수집했고 이들 내부에 존재하는 보안 취약점 및 취약점 간의 연관 관계를 면밀히 조사했다.

수집한 이미지에 대한 종합적인 분석을 통해 본 논문에서는 공식 이미지를 포함한 대부분의 컨테이너 이미지에 다양한 취약점이 존재함을 밝혔고, 이러한 취약점에 대한 패치가 이미지에 적용될 때까지 상당한 시간이 소요된다는 것을 증명했다. 또한 컨테이너 이미지의 계층 구조에서 상위 계층에 존재하는 소수의 이미지 내부의 취약점이 이들 하위의 수많은 다른 이미지에 영향을 미친다는 것과 이러한 취약점의 연쇄 작용 해결을 위해서는 큰 비용이 소모된다는 것을 밝혔다. 본 논문에서 제공하는 이러한 실험 결과와 시사점은 향후 컨테이너 이미지 보안 향상을 위한 많은 연구의 초석으로 활용될 수 있을 것이다.

## References


[1] A. Randal, “The ideal versus the real: Revisiting the history of virtual machines and containers,” *ACM Comput. Surv.(CSUR)*, vol. 53, no. 1, pp. 1-31, Jan. 2020. (https://doi.org/10.1145/3365199)

[2] *Docker*, Retrieved Apr. 28, 2022, from https://www.docker.com/.

[3] *Docker hub*, Retrieved Apr. 28, 2022, from https://hub.docker.com/.

[4] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, “Docker ecosystem-vulnerability analysis,” *Comput. Commun.*, vol. 122, pp. 30-43, Jun. 2018. (https://doi.org/10.1016/j.comcom.2018.03.011)

[5] Tripwire, *Tripwire State of Container Security Report*(2019), Retrieved Apr. 28, 2022, from https://www.tripwire.com/solutions/devops/tripwire-dimensional-research-state-of-container-security-report-register

[6] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, “A measurement study on linux

container security: Attacks and counter-measures," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, pp. 418-429, San Juan PR, USA, Dec. 2018. (https://doi.org/10.1145/3274694.3274720)

[7] K. Suo, Y. Zhao, W. Chen, and J. Rao, "An analysis and empirical study of container networks," in *IEEE INFOCOM 2018-IEEE Conf. Comput. Commun.*, pp. 189-197, Honolulu, USA, Apr. 2018. (https://doi.org/10.1109/INFOCOM.2018.8485865)

[8] *docker-library/official-images*, Retrieved Apr. 28, 2022, from https://github.com/docker-library/official-images.

[9] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proc. Seventh ACM on Conf. Data and Appl. Secur. and Privacy*, pp. 269-280, New York, USA, Mar. 2017. (https://doi.org/10.1145/3029806.3029832)

[10] *Docker run reference*, Retrieved Apr. 28, 2022, from https://docs.docker.com/engine/reference/run/#docker-run-reference.

[11] *Common vulnerabilities and exposures*, Retrieved Apr. 28, 2022, from https://cve.mitre.org/.

[12] *Snyk*, Retrieved Apr. 28, 2022, from https://snyk.io/.

[13] *National vulnerability database*, Retrieved Apr. 28, 2022, from https://nvd.nist.gov/.

[14] *Alpine linux*, Retrieved Apr. 28, 2022, from https://www.alpinelinux.org/.

[15] *Vulnerability metrics*, Retrieved Apr. 28, 2022, from https://nvd.nist.gov/vuln-metrics/cvss.

[16] *CVE-2005-2541*, Retrieved Apr. 28, 2022, from https://nvd.nist.gov/vuln/detail/CVE-2005-2541.

[17] *Docker library*, Retrieved Apr. 28, 2022, from https://github.com/docker-library/official-images.

[18] K. Wist, M. Helsem, and D. Gligoroski, "Vulnerability analysis of 2500 docker hub images," in *Advances in Secur., Netw., and Internet of Things*, Springer, Cham, pp. 307-327, Mar. 2021. (https://doi.org/10.1007/978-3-030-71017-0_22)

[19] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *2020 10th Annu. CCWC*, pp. 975-980, 2020.

[20] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W.-H. Lee, T. Lu, W. Chen, and R. Beyah, "Understanding the security risks of docker hub," in *Eur. Symp. Res. Comput. Secur.*, vol. 12308, pp. 257-276, Springer, Cham, Sep. 2020. (https://doi.org/10.1007/978-3-030-58951-6_13)

[21] *Virustotal*, Retrieved Apr. 28, 2022, from https://www.virustotal.com/gui/.

[22] S. Ghavamnia, T. Palit, A. Benameur, and M. Polychronakis, "Confine: Automated system call policy generation for container attack surface reduction," in *23rd Int. Symp. RAID*, pp. 443-458, Virtual *Conf.*, Oct. 2020.

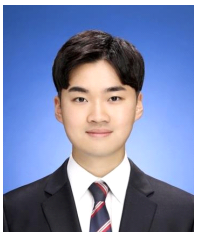
[23] L. Lei, J. Sun, K. Sun, C. Shenefiel, R. Ma, Y. Wang, and Q. Li, "Speaker: Split-phase execution of application containers," in *Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 230-251, Universität Bonn, Germany, Jul. 2017. (https://doi.org/10.1007/978-3-319-60876-1_11)

[24] Z. Wan, D. Lo, X. Xia, L. Cai, and S. Li, "Mining sandboxes for linux containers," in *2017 IEEE ICST, Verification and Validation*, pp. 92-102, Tokyo, Japan, Mar. 2017. (https://doi.org/10.1109/ICST.2017.16)

[25] Y. Park, H. Yang, and Y. Kim, "Performance comparison of container networking technologies," *J. KICS*, vol. 44, no. 1, pp. 158-170, Jan. 2019. (http://doi.org/10.7840/kics.2019.44.1.158)

[26] S. Jeong, J. G. Kim, K. Yeom, and J. Park, "Container orchestration framework deployment technique based on the feature of microservice," *J. KICS*, vol. 46, no. 3, pp. 466-475, Mar. 2021. (http://doi.org/10.7840/kics.2021.46.3.466)

**유 명 성** (Myoungsung You)



2020년 2월 : 충북대학교 컴퓨터 공학과 학사
2022년 2월 : KAIST 정보보호 대학원 석사
2022년 3월~현재 : KAIST 전기및전자공학부 박사과정

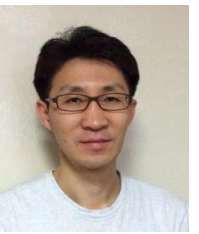
<관심분야> 클라우드 시스템, 정보보안, 네트워크 데이터 평면
[ORCID:0000-0001-5822-5243]

**김 재 한** (Jaehan Kim)

2020년 2월 : KAIST 전기및전자공학부 학사
2022년 2월 : KAIST 전기및전자공학부 석사
2022년 3월~현재 : KAIST 전기및전자공학부 박사과정

<관심분야> 정보보안, CTI, 머신러닝
[ORCID:0000-0001-8048-097X]

**신 승 원** (Seungwon Shin)



2000년 2월 : KAIST 전기및전자공학부 석사
2013년 2월 : Texas A&M University 전산학 박사
2002년~2006년 : ETRI 연구원
2005년~2006년 : MIT 초빙 연구원
2006년~2009년 : TmaxSoft 책임 연구원
2013년~현재 : KAIST 전기및전자공학부 교수
<관심분야> 네트워크 보안, SDN, CTI
[ORCID:0000-0002-7627-5815]