안전한 컨테이너 환경 구축/운영을 위한 보안 위협 검증 자동화 플랫폼

Security Threat Verification Automation Platform to Build and Operating Secure Container Environment

최상훈, 박준규, 박기웅[1)]
Sang-Hoon Choi, Jun-Gyu Park, Ki-Woong Park

(05006) 서울특별시 광진구 능동로 209 세종대학교 정보보호학과 시스템보안 연구실, 정보보호학과[1)]
{csh0052, wnsrb3001}@gmail.com, woongbak@sejong.ac.kr

요 약

최근 컨테이너는 기존의 가상 머신보다 낮은 구동 오버헤드와 다양한 컴퓨팅 환경에서도 안정적으로 작동하는 장점이 있어 IT 업계에서 많은 관심을 받고 있다. 하지만 컨테이너 기반의 플랫폼은 다수의 컨테이너가 커널을 공유하기 때문에 사이드 채널 공격이 가능하다는 문제점과 취약한 패키지를 사용하는 컨테이너 이미지가 다수의 사용자와 공유 될 수 있다는 문제점 등 다양한 측면에서의 보안 위협이 존재한다. 본 논문에서는 안전한 컨테이너 환경 구축 및 운영을 위해 CIS에서 발표한 Docker Benchmark 문서의 보안 체크리스트 항목을 자동화하여 검증 할 수 있는 플랫폼을 제안한다. 본 논문에서 제안한 프레임워크는 보안 체크리스트 97가지 항목에 대한 자동화된 검증을 수행하며, 컨테이너 이미지에 대한 취약점 검증과 컨테이너 인스턴스의 행위를 분석할 수 있는 실시간 모니터링 기능을 제공한다.

Abstract

Containers are attracting much attention in the IT industry because they have the advantages of lower operating overhead usage than existing virtual machines and stable operation in various computing environments. However, there are security threats in various aspects such as a problem that a container-based platform can be attacked through a side channel because of a single kernel shared with a plurality of containers, and a container image using a vulnerable package can be shared with a large number of users. In this paper, we propose an automated platform which verifies the security checklist items of Docker Benchmark documents published by CIS for building and operating a secure container environment. The proposed framework verifies 97 security checklists automatically, and provides real-time monitoring functions for analyzing the vulnerability

※ 본 연구는 한국연구재단 지원사업(2017R1C1B2003957) 및 2018년도 과학기술정보통신부의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2018-0-00420, API 호출 단위 자원 할당 및 사용량 계량이 가능한 서버리스 클라우드 컴퓨팅 기술 개발)

1) 교신저자

논문투고일자 : 2018. 10. 08 심사일자 : 2018. 10. 08 게재확정일자 : 2018. 10. 23

of container image and behavior of container instance.

키워드: 클라우드 컴퓨팅, 컨테이너, 보안
Keyword: Cloud Computing, Container, Security

1. 서론

최근 클라우드 플랫폼을 더 효율적으로 사용하기 위한 다양한 프레임워크가 연구되고 있다. 특히, 컨테이너 기술이 각광 받고 있다 [1, 2]. 컨테이너 가상화 기술은 Linux 계열의 운영체제에서 제공하는 LXC(Linux Container)[3]를 기반으로 운영체제 레벨의 가상화 기법을 활용하는 것이다. 컨테이너 기반의 가상화 기술은 에뮬레이션 구조로 하드웨어를 제공하는 호스트 기반의 가상화 방식과 다르게 사용자 cgroups[4]를 통해 프로세스들의 자원의 사용(CPU, 메모리, 디스크 입출력, 네트워크 등)을 제한하고, namespace[4]를 통해 프로세스가 각자의 독립된 공간을 할당받은 것과 같은 격리된 환경을 제공한다. 이와 같은 에뮬레이션 구조는 기존의 호스트 기반의 가상화 방식과 비교하여 상대적으로 낮은 구동 오버헤드와 이미지 재사용과 같은 장점이 부각되며 급부상하였다.

하지만 컨테이너 기술은 커널 영역을 모든 컨테이너가 공유하기 때문에 호스트 운영체제 커널에 취약점이 있을 경우 컨테이너 탈출, 특권권한 탈취와 같은 보안 문제들을 발생시킬 수 있다. 또한 컨테이너의 이미지는 공유되어 재사용되기 때문에 배포될 때에 취약성을 가진 상태의 컨테이너가 유출되거나 혹은 악성코드가 심겨진 컨테이너가 배포될 수 있다. 이외에도 컨테이너 기반의 플랫폼은 다수의 컨테이너가 동시에 실행되기 때문에 병렬적인 모니터링의 어려움과 컨테이너 엔진에 의해 컨테이너가 실행됨에 따라 생기는 가시성 부족 문제 등 다양한 측면에서의 보안 위협이 있다[5, 6].

이러한 보안위협에 따라 컨테이너 인스턴스 보안 위협을 사전에 예방하고자 CIS(Center for Internet Security) [7]에서 안전한 컨테이너 환경을 구축 및 운영하기 위한 체크리스트 문서를 발표하였다 [8]. CIS(Center for Internet Security)는 사이버 위협으로부터 개인 및 공공 조직을 보호하기 위해 전 세계적인 IT 커뮤니티의 힘을 활용하는 비영리 단체이다. CIS에 발표한 CIS Docker Benchmark는 보안성이 확립된 Docker 컨테이너 환경 구성을 위한 다양한 요구사항들을 도출하여 각각의 요구사항에 대한 정의, 해석 및 요구사항 충족 여부에 대한 감사 방법 등이 기술되어 있는 가이드라인이다. 이와 같은 문서에서 제공해주는 체크리스트들을 검증해주는 도구들이 등장했다 [9, 10, 11]. 하지만 기존의 체크리스트 검증 도구들은 실행중인 컨테이너를 검증하는 기능이나 로컬 호스트에 존재하는 안전하지 않은 이미지에 대해 검증하는 기능은 존재하지 않는다. 따라서 보안 체크리스트의 모든 항목을 검증하기 위해서는 수동으로 검증해야 한다는 한계점이 있다.

본 논문에서는 Docker Benchmark 문서에서 제공해주는 모든 보안 체크리스트 항목을 자동화하여 검증 할 수 있는 플랫폼을 제안하였다. 우리가 제안한 보안 위협 검증 자동화 플랫폼은 4가지의 특징이 있다. 첫째, 안전한 컨테이너 환경을 구성하기 위해 97가지에 대한 보안체크리스트를 검증한다. 둘째, 검증 결과 시각화를 통해 보안 위협 범위를 쉽게 파악할 수 있다. 셋째, 컨테이너 실시간 모니터링을 통해 구동중인 컨테이너 인스턴스 내부에서 발생 할 수 있는 보안 항목 위반 행위를 탐지, 마지막으로 컨테이너 이미지에 설치 된 패키지 정보를 추출해 안전한 이미지인지 판단 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 안전한 컨테이너 환경을 구축하고 운영하기 위해 수행되었던 기존 연구 사례들을 조사하고, 3장에서는

기존의 컨테이너 보안 도구 분석을 통해 기존 도구들의 한계점을 도출한다. 4장에서는 3장에서 도출된 한계점을 해결하기 위해 제시한 프레임워크를 설명하고, 5장에서는 결론 기술한다.

2. 관련연구

2.1 컨테이너 보안 요구사항 도출 연구

컨테이너의 보안 요구사항을 정의하여 컨테이너 보안 위협으로 인한 피해를 사전에 방지하는 것이 가능하다. 특정 조직에서는 컨테이너의 보안 요구사항들을 정의한 가이드라인을 배포한다. 이러한 가이드라인을 통해 우리는 컨테이너의 보안성을 확립하기 위해 제작된 기존의 컨테이너 보안 모니터링 도구들의 가이드라인에 정의된 요구사항들에 대한 검증 여부를 점검하는 것이 가능하다. 이를 통해 기존 도구들의 문제점을 파악하고 문제점을 해결하기 위한 방안을 제시한다.

CIS(Center for Internet Security)는 사이버 위협으로부터 개인 및 공공 조직을 보호하기 위해 전 세계적인 IT 커뮤니티의 힘을 활용하는 비영리 단체이다 [7]. 이러한 CIS에서는 다양한 컨테이너 보안 위협으로부터의 공격을 사전에 예방하기 위한 보안 구성 기준을 설정하기 위해 CIS Docker Benchmark를 저술하였다. CIS Docker Benchmark는 보안성이 확립된 Docker 컨테이너 환경 구성을 위한 다양한 요구사항들을 도출하여 각각의 요구사항에 대한 정의, 해석 및 요구사항 충족 여부에 대한 감사 방법 등이 기술되어 있는 가이드라인이다[8]. CIS Docker Benchmark는 6개의 큰 항목(호스트 구성, 도커 데몬 구성, 도커 데몬 구성파일, 컨테이너 이미지 및 빌드 파일, 컨테이너 런타임, 도커 보안 동작)으로 구성되어 있으며, 총 104개의 세부 항목에 대한 기준을 제공한다. 하지만 104가지 항목 중 7가지 항목은 정량적으로 측정 할 수 없는 항목이다. 즉 자동화 하여 검증할 수 있는 항목은 97가지 항목이다.

본 논문에서는 기존의 컨테이너 보안 모니터링 도구들에 실시간으로 발생하는 컨테이너 보안 위협에 대한 대응 및 방지하는 기능이 존재하는지 분석하고 문제점을 도출하기 위해 CIS Docker Benchmark의 모든 보안 요구사항 항목에 대한 기존 도구들의 검증 여부를 분석하였다. 또한, CIS Docker Benchmark의 버전에 따라 세부 항목의 차이가 있기 때문에 무료 Docker 제품인 Docker CE(Community Edition) 뿐만 아니라 상용 Docker 제품인 Docker EE(Enterprise Edition)가 설치된 컨테이너 환경에서의 보안 모니터링 도구 적용가능성도 고려하여 Docker CE를 기준으로 작성된 가이드라인이 아닌 Docker 버전 1.13.0 이상일 경우 적용 가능한 CIS Docker 1.13.0 Benchmark v1.0.0.(이하 'CIS Docker Benchmark'로 표기함)을 채택하였다.

2.2 컨테이너 보안 검증 관련 연구

2017년 IBM에서 발표한 "Usable Declarative Configuration Specification and Validation for Applications, System, and Cloud" 연구에서는 현대 소프트웨어 스택의 잘못된 구성을 진단하는 것이 어려워지고 있는 문제를 해결하기 위해 보안, 성능, 기능에 영향을 줄 수 있는 잘못된 구성에 대한 탐지가 가능한 규칙을 작성하는 유효성 검사 언어인 CVL(Configuration Validation Language)를 제시 하였다 [12]. 그리고 CVL 규칙을 적용하여 활용 할 수 있는 ConfigValidator를 제안하였다. ConfigValidator는 체크리스트를 기반으로 잘못된 구성을 진단하며 기본으로 작성되어 있는 체크리스트는 CIS(Center for Internet Security)에서 보안 요구사항 정의를 위해 저술한 CIS Docker Benchmark를 기준으로 작성되었다. ConfigValidator는 CIS Docker Benchmark의 항목 중 41%에 해당하는 부분에 대해서 점검이 가능하다는 특징을 가지고 있다.

2018년 Matthew Ridley는 시스템의 보안 상태를 파악하고 알려진 사이버 위협으로부터 시스템을 보호하기 위한 구성 방법을 평가하기 위한 기술 감사 수행이 가능하지만, 고객의 요구에 따라 시스템의 보안 상태를 정확하게 나타내는 중요한 메트릭으로 요약하기가 어렵다는 문제를 제기하였다 [13]. 이와 같은 문제를 해결하기 위해 보안 감사를 따르는 과

정에서 중요 보안 강화 요소와 배포된 시스템이 시스템별로 운영자가 설정한 목표를 충족하는지에 대해 고려한 메트릭을 제공하는 포괄적인 접근 방식을 제안하였다. 제안된 접근법은 CIS Docker Benchmark를 사용하여 Docker 컨테이너의 보안에 영향을 미치는 기본 요소를 결정한다. 또한, 메트릭을 계산하는데 필요한 데이터 수집을 위해 CIS Docker Benchmark를 기준으로 제작된 오픈소스 컨테이너 보안 모니터링 도구인 Docker Bench Security를 사용하였다. 하지만, 우리가 Docker Bench Security에 대해서 분석한 결과 CIS Docker Benchmark에서 제공해주는 모든 보안감사 체크리스트를 검증하지 못한다는 한계점이 존재하였다.

3. 컨테이너 보안 모니터링 도구 분석

CIS Docker Benchmark에 정의된 보안 요구사항과 연관성이 높은 도구들을 분석하기 위해 CIS Docker Benchmark를 기준으로 제작된 스크립트 기반 도구인 Docker Bench Security, Batten, Drydock을 선정하여 분석하였다 [9, 10, 11].

위의 4가지 도구들은 안전한 Docker 컨테이너 환경을 구축 및 운영하기 위한 컨테이너 환경 구성을 체크하는 메타스크립트이다. Docker Bench Security는 CIS Docker Community Edition Benchmark v1.1.0.을 기준으로, Batten과 Docker Bench Security는 CIS Docker 1.6 Benchmark 1.0.0.을 기준으로 제작되었다. 각 도구들의 CIS Docker Benchmark 모든 항목에 대한 검증 여부를 분석한 결과는 다음과 같다. 호스트 운영체제의 환경설정 및 도커 데몬에 대한 환경설정에 대해 분석한 결과는 <표 1>에 표기하였고, 도커 데몬과 관련된 파일 및 컨테이너 이미지에 관련된 분석 결과는 <표 2>에 표기하였다. 마지막으로 구동중인 컨테이너 및 도커 운영에 대한 보안 항목 분석 결과는 <표 3>에 표기하였다. 마지막으로 정량적으로 검증을 수행 할 수 없는 보안 체크리스트 항목들은 회색으로 표기하였다.

〈표 1〉 CIS Docker Benchmark 호스트 구성, 도커 데몬 구성 항목에 대한 컨테이너 보안 모니터링 도구들의 검증 여부 분석 결과

항목	세부항목	Docker Bench Security	Batten	Drydock
1.1	Create a separate partition for containers	O	O	O
1.2	Harden the container host			
1.3	Keep Docker up to date		O	O
1.4	Only allow trusted users to control Docker daemon	O	O	O
1.5	Audit docker daemon	O	O	O
1.6	Audit Docker files and directories - /var/lib/docker	O	O	O
1.7	Audit Docker files and directories - /etc/docker	O	O	O
1.8	Audit Docker files and directories - docker.service	O	O	O
1.9	Audit Docker files and directories - docker.socket	O	O	O
1.10	Audit Docker files and directories - /etc/default/docker	O	O	O
1.11	Audit Docker files and directories - /etc/docker/daemon.json	O	O	O
1.12	Audit Docker files and directories - /usr/bin/docker-containerd	O	O	O
1.13	Audit Docker files and directories - /usr/bin/docker-runc	O	O	O
2.1	Restrict network traffic between containers	O	O	
2.2	Set the logging level	O	O	
2.3	Allow Docker to make changes to iptables	O	O	
2.4	Do not use insecure registries	O	O	
2.5	Do not use the aufs storage driver	O	O	
2.6	Configure TLS authentication for Docker daemon	O	O	
2.7	Set default ulimit as appropriate	O	O	
2.8	Enable user namespace support	O		
2.9	Confirm default cgroup usage	O		
2.10	Do not change base device size until needed	O		
2.11	Use authorization plugin	O		
2.12	Configure centralized and remote logging	O		
2.13	Disable operations on legacy registry (v1)			
2.14	Enable live restore	O		
2.15	Do not enable swarm mode, if not needed			
2.16	Control the number of manager nodes in a swarm			
2.17	Bind swarm services to a specific host interface			
2.18	Disable Userland Proxy	O		
2.19	Encrypt data exchanged between containers on different nodes on the overlay network			
2.20	Apply a daemon-wide custom seccomp profile, if needed	O		
2.21	Avoid experimental features in production	O		
2.22	Use Docker's secret management commands for managing secrets in a Swarm cluster			
2.23	Run swarm manager in auto-lock mode			
2.24	Rotate swarm manager auto-lock key periodically			

〈표 2〉 CIS Docker Benchmark 도커 데몬 구성파일, 컨테이너 이미지 및 빌드 파일 항목에 대한 컨테이너 보안 모니터링 도구들의 검증 여부 분석 결과

항목	세부항목	Docker Bench Security	Batten	Drydock
3.1	Verify that docker.service file ownership is set to root:root	O	O	
3.2	Verify that docker.service file permissions are set to 644 or more restrictive	O	O	
3.3	Verify that docker.socket file ownership is set to root:root	O	O	
3.4	Verify that docker.socket file permissions are set to 644 or more restrictive	O	O	
3.5	Verify that /etc/docker directory ownership is set to root:root	O	O	
3.6	Verify that /etc/docker directory permissions are set to 755 or more restrictive	O	O	
3.7	Verify that registry certificate file ownership is set to root:root	O	O	
3.8	Verify that registry certificate file permissions are set to 444 or more restrictive	O	O	
3.9	Verify that TLS CA certificate file ownership is set to root:root	O	O	
3.10	Verify that TLS CA certificate file permissions are set to 444 or more restrictive	O	O	
3.11	Verify that Docker server certificate file ownership is set to root:root	O	O	
3.12	Verify that Docker server certificate file permissions are set to 444 or more restrictive	O	O	
3.13	Verify that Docker server certificate key file ownership is set to root:root	O	O	
3.14	Verify that Docker server certificate key file permissions are set to 400	O	O	
3.15	Verify that Docker socket file ownership is set to root:docker	O	O	
3.16	Verify that Docker socket file permissions are set to 660 or more restrictive	O	O	
3.17	Verify that daemon.json file ownership is set to root:root	O	O	
3.18	Verify that daemon.json file permissions are set to 644 or more restrictive	O	O	
3.19	Verify that /etc/default/docker file ownership is set to root:root	O	O	
3.20	Verify that /etc/default/docker file permissions are set to 644 or more restrictive	O	O	
4.1	Create a user for the container	O	O	O
4.2	Use trusted base images for containers		O	
4.3	Do not install unnecessary packages in the container			
4.4	Scan and rebuild the images to include security patches			
4.5	Enable Content trust for Docker	O		
4.6	Add HEALTHCHECK instruction to the container image	O		
4.7	Do not use update instructions alone in the Dockerfile	O		
4.8	Remove setuid and setgid permissions in the images			
4.9	Use COPY instead of ADD in Dockerfile	O		
4.10	Do not store secrets in Dockerfiles			
4.11	Install verified packages only			

〈표 3〉 CIS Docker Benchmark 컨테이너 런타임, 도커 보안 동작 항목에 대한 컨테이너 보안 모니터링 도구들의 검증 여부 분석 결과

항목	세부항목	Docker Bench Security	Batten	Drydock
5.1	Do not disable AppArmor Profile	O		O
5.2	Verify SELinux security options, if applicable	O	O	O
5.3	Restrict Linux Kernel Capabilities within containers	O	O	O
5.4	Do not use privileged containers	O		O
5.5	Do not mount sensitive host system directories on containers	O		O
5.6	Do not run ssh within containers			
5.7	Do not map privileged ports within containers	O		O
5.8	Open only needed ports on container			O
5.9	Do not share the host's network namespace	O		O
5.10	Limit memory usage for container	O		O
5.11	Set container CPU priority appropriately	O		O
5.12	Mount container's root filesystem as read only	O		O
5.13	Bind incoming container traffic to a specific host interface	O		O
5.14	Set the 'on-failure' container restart policy to 5	O		O
5.15	Do not share the host's process namespace	O		O
5.16	Do not share the host's IPC namespace	O		O
5.17	Do not directly expose host devices to containers	O		O
5.18	Override default ulimit at runtime only if needed	O		
5.19	Do not set mount propagation mode to shared	O		
5.20	Do not share the host's UTS namespace	O		
5.21	Do not disable default seccomp profile	O		
5.22	Do not docker exec commands with privileged option			
5.23	Do not docker exec commands with user option			
5.24	Confirm cgroup usage	O		
5.25	Restrict container from acquiring additional privileges	O		
5.26	Check container health at runtime	O		
5.27	Ensure docker commands always get the latest version of the image			
5.28	Use PIDs cgroup limit	O		
5.29	Do not use Docker's default bridge docker0	O		
5.30	Do not share the host's user namespaces	O		
5.31	Do not mount the Docker socket inside any containers	O		
6.1	Perform regular security audits of your host system and containers			
6.2	Monitor Docker containers usage, performance and metering			
6.3	Backup container data			
6.4	Avoid image sprawl	O	O	
6.5	Avoid container sprawl	O	O	

3.1 기존 컨테이너 보안 모니터링 도구의 한계점

3.1.1 검증하지 않는 항목 존재로 인한 보안 미흡 사항 식별의 한계

분석한 기존의 컨테이너 보안 모니터링 도구들은 CIS Docker Benchmark를 기준으로 제작되었기 때문에 다양한 항목들에 대한 검증을 수행한다. 하지만, <표 1, 2, 3>과 같이 검증하지 않는 보안 요구사항 항목들이 존재함으로써 기존의 도구들을 사용하여도 보안 미흡사항 식별의 한계가 존재한다. 이로 인해 다른 방법을 이용하여 보안 미흡사항에 대하여 식별하지 않은 채 기존의 컨테이너 보안 모니터링 도구에만 의존한다면 검증하지 않는 보안 요구사항 항목에 대한 보안 조치는 사실상 불가능하다.

3.1.2 도구 별 상이한 검증 항목

각 도구 별 검증하는 항목이 상이하고 검증하지 않는 항목들이 존재하기 때문에 한 가지의 도구만으로 모든 항목에 대한 보안 검증은 불가능하다. 결국 컨테이너 보안 모니터링 도구 사용자는 구축한 컨테이너 환경에 대한 보안성 체크 시 다양한 컨테이너 보안 모니터링 도구들을 설치하여 검증하여야만 한다.

3.1.3 보안 위협에 대한 대응의 한계

불시에 발생하는 컨테이너 보안 위협에 대응하기 위해 실시간 이상행위 탐지 모니터링은 중요한 요소이다. 하지만, 기존의 도구들은 사용자 요청에 의해 한 번의 검증만 수행하는 방식으로 작동되기 때문에 컨테이너 보안 요구사항을 위반하는 일이 발생하더라도 즉각적인 탐지가 불가능하다. 이로 인해 보안 위협이 발생하더라도 실시간으로 대응하는데 있어 한계가 존재한다.

4. 컨테이너 보안 위협 검증 자동화 플랫폼

본 장에서는 안전한 컨테이너 플랫폼을 운영하기 위해 컨테이너에서 발생 할 수 있는 다양한 보안 위협들을 일괄적으로 검증 할 수 있는 CIS Docker Benchmark 보안 체크리스트 항목 기반의 컨테이너 보안 위협 검증 자동화 플랫폼을 제안한다.

4.1 디자인 및 구현

컨테이너 플랫폼에서 보안 위협이 발생하는 요소들은 크게 3가지(컨테이너 이미지, 컨테이너 환경, 컨테이너 인스턴스)로 나누어진다. 보안위협이 발생하는 3가지의 영역에 대한 보안 검증을 일괄적으로 수행하기 위해 다음과 같은 3가지의 검증 모듈을 설계하였다.

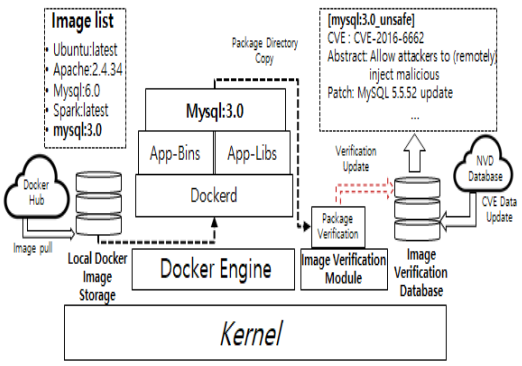
4.1.1 컨테이너 환경 검증 모듈

안전한 컨테이너의 환경을 구성하기 위해서는 첫 번째로 컨테이너들이 구동되는 호스트 운영체제 환경에 대한 검증이 필요하다. CIS Docker Benchmark 문서에서는 안전한 컨테이너 환경을 검증하기 위해 구성되어 있는 도커 환경의 주요 환경설정 파일들에 대한 접근 권한 확인을 요구한다. 추가적으로 로그를 기록하기 위한 환경설정 정보, iptables에 대한 접근권한, 인스턴스의 이미지가 저장되는 스토리지 규격, 도커 데몬에 대한 TLS 인증 등을 확인한다. 컨테이너 환경 검증 모듈은 컨테이너 환경에 대한 33가지 항목에 대해서 일괄적으로 검증을 수행한다. 보안 체크리스트에 기반을 둔 검증 결과는 위협에 노출되어 있다면 Warning으로 표기하고 안전한 경우에는 Pass로 표기하도록 설계하였다.

4.1.2 컨테이너 이미지 검증 모듈

컨테이너 기술은 이미지를 공유하여 다수의 사용자들이 재사용 할 수 있다는 장점이 있다. 하지만 악성코드가 설치 된 이미지 또는 취약한 버전의 패키지가 설치된 이미지가 공유되어 재사용되면 많은 사용자들이 보안 위협에 노출되게 된다 [14, 15, 16, 17]. 하지만 기존 도구들을 분석해본 결과 컨테이너 이미지 내부에 설치 된 패키지들에 대해서 검증하는 과정은 없었다. 안전한 컨테이너를 운영하기 위해 안전한 이미지 사용이라는 체크리스트 항목은 존재하였으나, 기존 도구들은 컨테이너 이미지에 대한 보안 검증이 이뤄지지 않았다. 우리는 컨테이너 이미지에 설치된 패키지의 보안검증을 수행

하기 위해 컨테이너 이미지 검증 모듈을 (그림 1)과 같이 설계 하였다.



(그림 1) 컨테이너 이미지 검증 과정

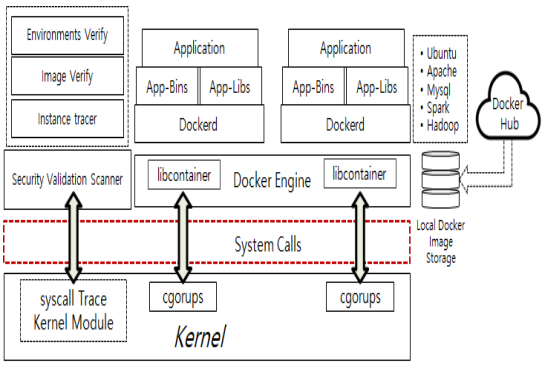
먼저, 이미지에 설치 된 패키지들을 검증하기 위해서는 취약한 패키지 정보와 취약점 리스트들이 필요하다. 우리의 컨테이너 이미지 검증 모듈은 최신 CVE(Common Vulnerabilities and Exposures) 정보들을 최신 데이터로 동기화하기 위해 NIST (National institute of standards and technology)에서 제공하는 NVD(National Vulnerability Database)를 활용한다 [18, 19]. NVD 데이터에는 CVE 번호, CVE 별 취약한 패키지들의 버전, 취약점에 대한 상세 정보 등을 제공해준다. 컨테이너 이미지 검증모듈의 검증과정은 다음과 같다.

첫 번째, 컨테이너 이미지 검증 모듈은 init 과정에서 NVD를 내려 받아 이미지 검증 데이터베이스에 업데이트 한다. 두 번째, 로컬 컨테이너 이미지 저장소에 저장되어 있는 이미지 리스트를 추출한다. 세 번째, 검증되지 않은 이미지를 컨테이너로 구동시킨다. 구동 시킨 후 패키지가 설치되는 디렉터리 전체를 호스트 운영체제로 추출한다. 네 번째, 구동된 컨테이너 인스턴스로부터 추출 한 패키지들을 로컬에 저장된 취약한 패키지 데이터베이스와 비교분석을 수행한다. 최종적으로 취약한 버전의 패키지가 설치 된 이미지들에 대해서 이미지 검증 결과를 데이터베이스에 저장한다. 취약한 이미지가 발견된 경우 CVE 번호, 취약점에 대한 정보, 안전한 이미지로 패치하기 위한 방법 등이 데이터베이스에 저장된다. 사용자가 해당 이미지를 사용하여 컨테이너 구동 시 경고메시지를 출력하기 위하여, 도커 이미지 이름규격에 'unsafe' 라는 태그 추가한다. 사용자는 취약한 이미지를 사용 할 때 마다 'unsafe' 라는 태그를 입력하여 구동 시켜야하기 때문에 취약한 이미지를 사용한다는 것을 각인 시킬 수 있다.

4.1.3 실시간 컨테이너 인스턴스 모니터링 모듈

우리는 안전한 컨테이너 환경을 구축하고 운영하기 위해 기존의 컨테이너 보안 도구들을 분석하였다. 특히, Docker Benchmark의 보안 항목에는 컨테이너 인스턴스에 대한 실시간 모니터링이 수행되어야만 검증 가능한 보안 체크리스트 항목들이 존재한다. 하지만, 3가지(Docker Bench Security, Batten, Drydock) 도구 모두 컨테이너 인스턴스에 대한 실시간 모니터링 기능은 제공하지 않는다. 기존의 컨테이너 보안 도구들은 주기적인 모니터링이 아닌 보안 점검 요청 시 한 번의 보안 검증을 수행하는 One-Shot 형태의 검증 기능만 제공한다. Docker Bench Security와 Drydock은 CIS Docker Benchmark 문서의 "4.11 Install verified packages only", "5.6 Do not run ssh within containers" 등 인스턴스와 관련된 체크리스트 항목들에 대해 검증할 수 있다고 소개되어있다. 하지만, 인스턴스와 관련된 항목들에 대한 보안 검증 결과를 분석해 본 결과 인스턴스 내부 모니터링이 필요한 항목들에 대해서는 검증이 이루어지지 않았다.

우리는 이와 같은 문제점을 해결하기 위해 구동중인 인스턴스를 모니터링하여 보안 체크리스트에 대한 보안 검증을 수행 할 수 있는 모듈을 디자인하였다. 인스턴스의 체크리스트 위반을 검증하기 위해 구현한 실시간 컨테이너 인스턴스 모니터링 모듈은 (그림 1)과 같다.



(그림 2) 컨테이너 보안 검증 자동화

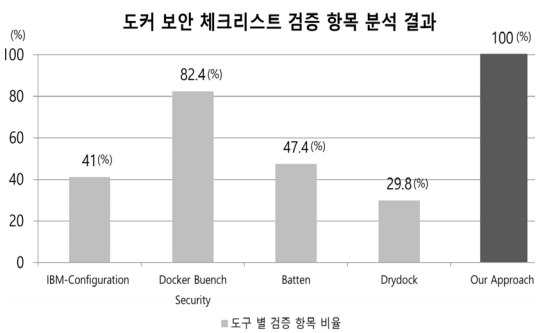
구동중인 컨테이너 인스턴스의 행위 정보를 분석하기 위해서는 컨테이너에서 실시간으로 호출하는 System call 정보를 모니터링 하는 것이 가장 정확하고 효과적이다. 특히, 컨테이너는 호스트 운영체제의 자원을 활용하는 운영체제 기반 가상화 기술의 특징을 가지고 있다. 이와 같은 특징은 컨테이너 인스턴스 내부에 별도의 에이전트 설치 없이, 호스트 운영체제 System call을 모니터링 하는 것만으로도 컨테이너 내부의 행위 정보에 대해 상세히 모니터링 할 수 있다. 예를 들어 구동중인 프로세스 리스트, 파일 쓰기/읽기, 프로세스 실행 요청 등 컨테이너 인스턴스의 모든 정보를 파악 할 수 있다. 컨테이너의 특징을 활용하여 보안 체크리스트를 효율적으로 검증하기 위해 다음과 같은 과정을 통해 보안 체크리스트를 검증하도록 하였다.

먼저 우리는 호스트 운영체제 커널에 도커 엔진에서 호출하는 모든 System call 정보를 기록할 수 있는 커널 모듈을 구현하여 적재시켰다. 해당 커널 모듈은 컨테이너에서 발생하는 모든 System call을 추적하는 역할을 수행한다. System call 추적 모듈에서 추출할 수 있는 시스템 콜 호출정보를 활용하여 보안검증을 수행하기 위해 Security Validation Scanner를 디자인 하였다. Security Validation Scanner는 컨테이너 내부에서 보안 체크리스트를 위반하는 행위가 발생했을 때 이를 감지하고 해당 컨테이너를 일시정지 시키는 일을 수행한다. 가령, CIS Docker Benchmark 보안 항목 중 "4.11 Install verified packages only"와 같이 컨테이너 내부에 SSH가 실행되면 보안이 위반되는 항목의 경우 SSH 데몬이 컨테이너 내부 프로세스 목록에 로드되면 이를 탐지하고 해당 컨테이너를 중지 시키는 과정을 거친다.

4.2 보안 위협 검증 자동화 플랫폼

우리가 제안한 보안 위협 검증 자동화 플랫폼은 크게 다음과 같은 3가지의 특징을 가진다.

첫 번째, CIS Docker Benchmark에서 제공해주는 보안 체크리스트 중 정량화 할 수 있는 모든 보안 체크리스트 항목 97가지에 대한 검증을 자동화 하여 수행할 수 있다. (그림 3)과 같이 IBM에서 2017년 발표한 연구 결과에서는 41%에 해당하는 체크리스트 항목들에 대해 검증 할 수 있었지만, 우리가 제안한 보안 위협 검증 자동화 플랫폼은 정량화 된 97가지 보안체크리스트들에 대해 100% 검증 할 수 있다.



(그림 3) 도커 보안 체크리스트 검증 항목 비교

두 번째, 검증결과에 대한 시각화를 통해 컨테이너 환경에서 보안에 취약한 부분을 쉽게 파악할 수 있다. 우리는 보안 체크리스트 분석을 통해 컨테이너의 보안 위협이 발생할 수 있는 범주가 컨테이너 환경, 컨테이너 이미지, 컨테이너 인스턴스라는 결과를 도출 하였다.

(그림 4) 컨테이너 보안리스트 검증 결과

컨테이너의 보안 위협 범위를 보안 체크리스트 리포트 결과와 일치시키기 위해 (그림 4)와 같이 보안에 취약한 영역을 한눈에 파악 할 수 있도록 시각화하였다. 위와 같은 보고서 형식의 시각화 기능을 통해 현재의 컨테이너 환경의 어느 영역이 보안 위협에 노출되어 있는지 쉽게 파악 할 수 있다.

세 번째, 컨테이너에 대한 실시간 모니터링이 가능하다. 우리는 컨테이너에서 호출되는 모든 System call을 추적하기 위해 커널에 적재할 수 있는 모듈을 설계 하였다. 컨테이너의 행위 정보 모니터링을 통해 기존의 보안 체크리스트 검증 도구들이 검증 할 수 없었던 인스턴스 내부 항목들을 자동화 하여 검증 할 수 있다. 더 나아가, 해당 모듈을 기능을 활용하면 Apparmor [20] 처럼 컨테이너 인스턴스 내부에서의 특정 행위를 제한하기 위한 보안 프로필을 따로 지정하여 컨테이너를 구동하지 않아도, 실행중인 컨테이너 내부에서 특정 명령어 및 System call을 호출 했을 때 이를 감지하고 실시간 대응 할 수 있도록 확장하여 활용 할 수 있다.

마지막으로, 로컬 호스트 저장소에 존재하는 컨테이너 이미지들에 대한 보안성 검증을 수행할 수 있다. 우리게 제안한 시스템은 NVD에서 제공해주는 CVE 데이터베이스를 활용해 컨테이너 이미지에 설치되어 있는 패키지들이 취약점을 가지고 있는지 여부를 자동화하여 검증 할 수 있다.

5. 결론

최근 컨테이너는 기존의 가상 머신보다 낮은 구동 오버헤드와 다양한 컴퓨팅 환경에서도 안정적으로 작동하는 장점이 있어 IT 업계에서 많은 관심을 받고 있다. 하지만, 커널 공유로 인해 발생하는 보안 위협은 호스트 운영체제에 구동중인 모든 컨테이너의 보안위협으로 확장 될 수 있다. 또한 취약한 패키지를 사용하는 컨테이너 이미지가 다수의 사용자와 공유 될 수 있다. 따라서 안전한 컨테이너 환경을 구축 및 운영하기 위해서는 보안 체크리스트에 대한 검증이 필수적이다. 하지만, 기존 컨테이너 보안 체크리스트 검증 도구들은 모든 보안 체크리스트 항목을 검증할 수 없다는 한계 및 실시간 모니터링 기능이 존재하지 않아 불시에 발생하는 보안 위협 대응 관련 한계점을 가지고 있다.

본 논문에서는 CIS에서 저술한 Docker Benchmark 문서에서 제공하는 모든 보안 체크리스트 항목을 자동화하여 검증 할 수 있는 플랫폼을 제안하였다. 우리가 제안한 보안 위협 검증 자동화 플랫폼은 4가지의 특징이 있다. 첫째, 안전한 컨테이너 환경을 구성하기 위해 97가지에 대한 보안체크리스트를 검증한다. 둘째, 검증 결과 시각화를 통해 보안 위협 범위를 쉽게 파악할 수 있다. 셋째, 컨테이너 실시간 모니터링을 통해 구동중인 컨테이너 인스턴스 내부에서 발생하는 보안 항목 위반을 탐지 할 수 있다. 마지막으로, 컨테이너 이미지에 대한 취약점 점검을 통해 취약한 컨테이너 이미지를 판단 할 수 있다. 차후 연구에서는 컨테이너 인스턴스 내부의 행위 정보를 수집하고 이를 분석하여 악의적인 행위를 하는 컨테이너 사용자나 악성코드에 감염된 컨테이너를 탐지하는 연구를 수행할 예정이다.

참고문헌

〔1〕 Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014.239, 2014.

〔2〕 Chung, Minh Thanh, et al. "Using docker in high performance computing applications." Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on. IEEE, 2016.

〔3〕 LXC, https://linuxcontainers.org

〔4〕 Rosen, Rami. "Resource management: Linux kernel namespaces and cgroups." Haifux, May 186, 2013.

〔5〕 김진택, et al. "클라우드컴퓨팅 기술 스택 분석." 한국차세대컴퓨팅학회 논문지 11.6, 79-89, 2015.

〔6〕 김원일, et al. "비정상 권한 흐름 탐지 보안 모듈의 설계와 구현." 한국차세대컴퓨팅학회 논문지 10.2, 66-74, 2014.

〔7〕 CIS(Center for Internet Security), https://www.cisecurity.org

〔8〕 Docker, C. I. S. "1.11.0 Benchmark." Center for Internet Security, 2017.

〔9〕 docker-bench-security, https://github.com/docker/docker-bench-security

〔10〕 batten, https://github.com/dockersecuritytools/batten

〔11〕 drydock, https://github.com/zuBux/drydock

〔12〕 Baset, Salman, et al. "Usable declarative configuration specification and validation for applications, systems, and cloud." Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track. ACM, 2017.

〔13〕 Ridley, Matthew, et al. "Quantifying the Security Posture of Containerized Mission Critical Systems." SoutheastCon 2018. IEEE, 2018.

〔14〕 Bui, Thanh. "Analysis of docker security." arXiv preprint arXiv:1501.02967, 2015.

〔15〕 Combe, Theo, Antony Martin, and Roberto Di Pietro. "To Docker or not to Docker: A security perspective." IEEE Cloud Computing 3.5, 54-62 2016.

〔16〕 Jian, Zhiqiang, and Long Chen. "A defense method against docker escape attack." Proceedings of the 2017 International Conference on Cryptography, Security and Privacy. ACM, 2017.

〔17〕 탁병철. "컨테이너 이미지의 보안 취약성 조사 연구." 한국차세대컴퓨팅학회 논문지 14.3 , 7-15, 2018.

〔18〕 NIST, https://www.nist.gov/

〔19〕 NVD, NIST. "National Vulnerability Database.", 2011.

〔20〕 Bauer, Mick. "Paranoid penguin: an intro duction to Novell AppArmor." Linux Journal 2006.148, 13, 2006.

저자소개

◆ 최상훈

- 2014년 대전대학교 정보보안학과 학사
- 2016년 대전대학교 전산정보보안학과 석사
- 2017년 세종대학교 정보보호학과 박사 과정
- 관심분야: 클라우드 컴퓨팅, 시스템 보안, 디지털 포렌식 등

◆ 박준규

- 2018년 대전대학교 정보보안학과 학사
- 2018년 세종대학교 정보보호학과 석사 과정
- 관심분야: 클라우드 컴퓨팅, Capture The Flag(CTF), 시스템 보안 등

◆ 박기웅

- 2005년 연세대학교 Computer Science 학사
- 2007년 KAIST Electrical Engineering 석사
- 2012년 KAIST Electrical Engineering 박사
- 2008년 Microsoft Research Asia, Wireless and Networking Grourp, Research Intern
- 2009년 Microsoft Research, Network Research Group, Graduate Research Fellow
- 2012년 국가보안기술연구소 연구원
- 2012년 ~ 2016년 대전대학교 정보보안학과 교수
- 2016년 ~ 현재 세종대학교 정보보호학과 교수
- 관심분야: 시스템 보안, 모바일-클라우드 컴퓨팅, 보안 프로토콜, 디지털 포렌식 등