NIST Special Publication 800-190

애플리케이션 컨테이너 보안 가이드

Application Container Security Guide

권 리

- 저자
  - Murugiah Souppaya, NIST
  - John Morello, Twistlock
  - Karen Scarfone, Scarfone Cybersecurity

- 번역 : Youngeun Moon, Blackfalcon Security

개 요

컨테이너로 알려진 애플리케이션 컨테이너 기술은 애플리케이션 소프트웨어 패키징과 결합된 운영 체제 가상화의 한 형태이다. 컨테이너는 애플리케이션을 패키징하고 실행하기 위해 이식/재사용/자동화하는 방법을 제공한다. 이 문서에서는 컨테이너 사용과 관련된 잠재적인 보안 이슈를 설명하고, 이러한 이슈를 해결하기 위한 권고사항을 제시한다.

독 자

이 문서는 시스템/보안 관리자, 보안 프로그램 운영자(manager), 정보시스템 보안 책임자, 애플리케이션 개발자, 애플리케이션 컨테이너 기술의 보안에 대한 책임이나 관심이 있는 인원을 대상으로 한다. 이 문서는 독자들이 가상화 기술(하이퍼바이저 및 가상 머신), 운영 체제, 네트워크, 보안에 대해 어느 정도 전문 지식을 보유하고 있다고 가정한다. 애플리케이션 컨테이너 기술은 계속 변화화고 있기 때문에, 독자들은 다양한 문서를 통해 최신의 세부적인 정보를 얻기를 권고한다.

목 차

부록 목차

요 약

OS(Operating System, 운영 체제) 가상화는 각 애플리케이션에 별도의 가상화된 OS를 제공한다. 따라서, 각 애플리케이션은 서버의 다른 모든 애플리케이션과 분리된 상태[1]를 유지한다. OS 가상화는 사용 편의성이 향상되고, 주요 이점인 개발 신속성(developer agility)에 더욱 주력하면서 인기가 증가해 왔다. 현재, OS 가상화 기술은 주로 애플리케이션(앱)을 패키징하고 실행하기 위해 이식/재사용/자동화할 수 있는 방법을 제공하는데 초점을 맞추고 있다. 애플리케이션 컨테이너 또는 단순히 컨테이너라는 용어는 이러한 기술을 가리키는데 주로 사용된다. 이 문서의 목적은 컨테이너 기술과 관련된 보안 이슈를 설명하고, 컨테이너를 계획-구현-관리할 때 이러한 이슈를 해결하기 위한 실질적인 권고사항을 제시하는데 있다. 이러한 권고사항은 컨테이너 기술 아키텍처(그림 1) 內 특정 컴포넌트 또는 특정 계층(tier)에만 해당되는 경우가 많다.

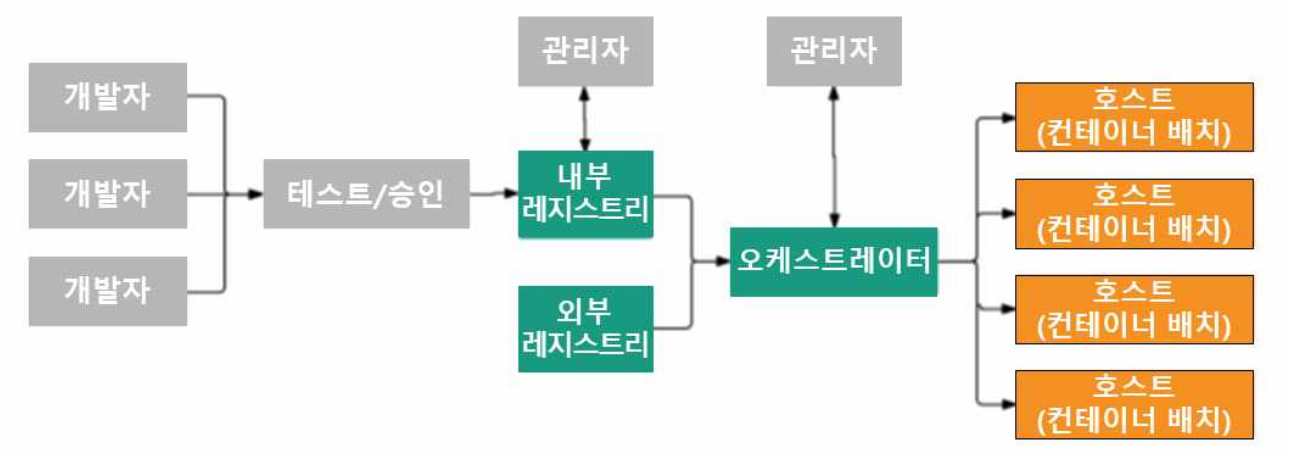


그림 1 : 컨테이너 기술 아키텍처의 계층 및 컴포넌트

조직에서는 다음의 권고사항을 준수해야 한다. 이 권고사항은 컨테이너 기술을 구현하고 사용할 때, 보안을 보장하는데 도움을 줄 것이다.

[1] 가상화된 OS에서 실행되는 애플리케이션은 다른 애플리케이션을 확인할 수 없고, 다른 애플리케이션의 영향을 받지 않는다.

컨테이너는 애플리케이션을 개발/실행/지원하는 새로운 방법을 제공하며, 이를 조직에 적용하기 위해 운영 문화 및 기술 프로세스를 조정해야 한다.

컨테이너 기술을 도입하면, 조직의 기존 문화 및 소프트웨어 개발 방법론이 와해될 수 있다. 기존 개발 실무/패치 기법/시스템 업그레이드 프로세스는 컨테이너 환경에 바로 적용하지 못할 수 있다. 직원들이 새로운 모델에 적응하려는 의지가 중요하다. 직원들은 컨테이너 내부에서 앱을 안전하게 구축하고 운영하기 위해 권장되는 업무 지침을 수용해야 하며, 조직에서는 컨테이너를 활용하기 위해 기존의 절차를 재고할 의지가 있어야 한다. 소프트웨어 개발 생명주기와 관련된 모든 사람에게 기술 및 운영 방법을 교육하고 훈련해야 한다.

공격 표면(attack surface)**을 줄이기 위해, 범용 호스트 OS 대신 컨테이너 전용**(container-specific) **호스트 OS를 사용해야 한다.**

컨테이너 전용 호스트 OS는 최소한의 기능만 보유한 OS로 컨테이너만 실행하도록 명시적으로 설계되었으며, 다른 모든 서비스와 기능이 비활성화되어 있고, 읽기 전용(read-only) 파일 시스템과 시스템 하드닝(hardening)이 적용되어 있다. 컨테이너 전용 호스트 OS를 사용하면 범용 호스트 OS를 사용할 때 보다 공격에 적게 노출되는 것이 일반적이다. 따라서, 컨테이너 전용 호스트 OS가 공격당하거나 침해될 확률이 더 낮다. 따라서, 조직에서는 가급적 컨테이너 전용 호스트 OS를 사용하여 위험을 줄여야 한다. 그러나, 컨테이너 전용 호스트 OS에도 취약점은 존재하며, 업데이트가 필요하다는 사실에 유의해야 한다.

심층 방어(defense in depth)**를 추가하기 위해, 목적/중요도/위협 상태가 동일한 컨테이너만 동일한 호스트 OS 커널을 사용할 수 있도록 그룹화해야 한다.**

대부분의 컨테이너 플랫폼은 컨테이너와 컨테이너를 분리하거나, 컨테이너와 호스트 OS를 분리하는데 효과적이다. 하지만, 동일한 호스트 OS에서 중요도가 다른 앱을 함께 실행시키는 것은 위험을 초래하는 불필요한 행위이다. 목적/중요도/위협 상태에 따라 컨테이너를 세분화하면, 심층 방어를 추가할 수 있다. 이러한 방식으로 컨테이너를 그룹화하면, 공격자가 한 그룹에 침투한 후 다른 그룹에 추가적으로 침투하는 것을 어렵게 한다. 컨테이너 그룹화는 침해를 예방/탐지할 가능성이 높이고, 잔존 데이터(캐시, 임시 파일이 저장된 로컬 볼륨 등)가 노출되지 않게 한다.
수백 개의 호스트와 수천 개의 컨테이너로 구성된 대규모 환경에서는 컨테이너 그룹화를 자동화해야 실용적으로 운영할 수 있다. 다행히, 컨테이너 기술은 일반적으로 앱을 그룹화할 수 있는 개념을 포함한다. 또한, 컨테이너 보안 도구는 컨테이너 이름 또는 라벨과 같은 속성을 사용하여 컨테이너에 보안 정책을 시행할 수 있다.

컨테이너 전용 취약점 관리 도구 및 프로세스를 도입하여 이미지에 대한 침해를 예방해야 한다.

기존 취약점 관리 도구는 호스트의 내구성, 앱의 업데이트 메커니즘/주기와 관련하여 많은 사항을 가정하고 있다. 그러나, 이러한 가정은 기본적으로 컨테이너 모델에 적합하지 않다. 예를 들어, 기존 취약점 관리 도구는 특정 서버에서 동일한 앱이 지속적으로 실행된다고 가정한다. 하지만, 애플리케이션 컨테이너가 실행되는 서버는 실행 당시의 리소스 가용성에 근거하여 달라진다. 또한, 기존 취약점 관리 도구는 컨테이너 내부의 취약점을 탐지할 수 없는 경우가 많다. 따라서, 기존 도구에 의한 결과로 안전성을 판단할 수 없다. 조직에서는 실용성/신뢰성이 더 높은 결과를 얻기 위해, 선언적/단계적 빌드 방식과 컨테이너/이미지의 불변성을 설계에 반영한 도구를 사용해야 한다.
취약점 관리 도구 및 프로세스는 이미지의 소프트웨어 취약점과 설정을 모두 고려해야 한다. 조직에서 도입한 취약점 관리 도구 및 프로세스는 이미지가 안전하게 설정되고, 관련 모범 사례(best practice)를 준수하는지 검증하고 시행해야 한다. 또한, 각 이미지의 준수 상태를 종합하여 레포팅 및 모니터링하고, 위반한 이미지의 실행을 방지하는 기능이 포함되어야 한다.

신뢰 컴퓨팅(trusted computing)에 대한 기준을 설정하기 위해, 하드웨어 기반 보호 대책을 사용하는 것을 검토해야 한다.

컨테이너 기술의 모든 계층에서 보안을 고려해야 한다. 현재 이를 달성하기 위한 방법은 업계 표준 TPM(Trusted Platform Module, 신뢰 플랫폼 모듈)과 같은 하드웨어 ROT(Root of Trust, 신뢰 기반)를 이용하는 것이다. 하드웨어 ROT에는 호스트의 펌웨어/소프트웨어/설정 데이터의 측정값이 저장된다.[1] 호스트를 부팅하기 전에 저장된 측정값으로 현재 측정값을 비교함으로써 호스트의 신뢰 여부를 확인한다. 하드웨어 ROT를 OS 커널과 OS 컴포넌트로 확장하여, 부팅 메커니즘/시스템 이미지/컨테이너 런타임/컨테이너 이미지를 암호학적으로 검증할 수 있다. 신뢰 컴퓨팅을 통해 컨테이너를 안전하게 구축/실행/조정/관리할 수 있다.

컨테이너 인식 런타임 방어 도구를 사용해야 한다.

런타임 동안 컨테이너에 대한 위협을 예방/탐지/대응할 수 있는 컨테이너 전용 보안 솔루션을 배치하고 사용해야 한다. IPS(Intrusion Prevention System, 침입 방지 시스템) 및 WAF(Web Application Firewall, 웹 방화벽)와 같은 기존 보안 솔루션은 컨테이너에 적절하게 보호할 수 없다. 기존 보안 솔루션은 컨테이너의 규모(the scale of containers)에서 운영할 수 없고, 컨테이너 환경에서 변화를 관리할 수 없고, 컨테이너 활동에 대한 가시성을 가질 수 없다. 컨테이너 환경을 모니터링할 수 있고 컨테이너 내부의 비정상적/악의적인 활동을 정밀하게 탐지할 수 있는 컨테이너 네이티브(native) 보안 솔루션을 활용해야 한다.

[1] 이를 RTM(Root of Trust for measurement)이라고 한다.

1. 서론

1.1. 목적 및 범위

이 문서의 목적은 애플리케이션 컨테이너 기술과 관련한 보안 이슈를 설명하고, 컨테이너를 계획-구현-관리할 때 이러한 보안 이슈를 해결하기 위한 실용적인 권고사항을 제시하는 것이다. 컨테이너 기술에 따라 일부 상이한 측면도 있지만, 이 문서의 권고사항은 대부분의 애플리케이션 컨테이너 기술에 적용할 수 있다.

애플리케이션 컨테이너를 제외한 모든 형태 가상화(가상 머신 등)는 이 문서의 범위를 벗어난다.

애플리케이션 컨테이너 기술 이외에도 모바일 기기에서 기업 데이터와 개인 데이터를 분리하기 위한 소프트웨어, 데스크톱 OS에서 일부 애플리케이션을 다른 애플리케이션과 분리하기 위한 소프트웨어와 같은 개념을 지칭하는 용어로 “컨테이너”를 사용한다. 이러한 소프트웨어는 애플리케이션 컨테이너 기술과 일부 속성을 공유할 수 있지만, 이 문서의 범위를 벗어난다.

이 문서는 독자들이 다음과 같은 기술(애플리케이션 컨테이너를 지원/상호 작용하는 기술)의 보안에 대해 이미 알고 있다고 가정한다.

- 애플리케이션 컨테이너 기술의 하위 레이어 : 하드웨어, 하이퍼바이저, OS
- 컨테이너 내부의 애플리케이션을 사용하는 관리자 도구
- 컨테이너 내부의 애플리케이션 및 컨테이너 자체를 관리하기 위한 관리자 엔드포인트

부록 A에는 이러한 기술의 보안과 관련한 자료를 정리해 놓았다. 또한, 3장과 4장에는 컨테이너 전용 OS의 보안 고려사항에 대해 추가적인 정보를 수록하고 있다. 위에서 나열한 기술의 보안과 관련한 추가적인 논의는 모두 이 문서의 범위를 벗어난다.

1.2. 문서 구조

이 문서는 다음과 같이 구성된다.

- 2장 : 컨테이너에 대한 배경지식 - 기술적인 기능, 기술 아키텍처 및 사용
- 3장 : 애플리케이션 컨테이너 기술의 핵심 컴포넌트에 대한 주요 위험 설명
- 4장 : 위험에 대한 보안 대책 권고
- 5장 : 컨테이너에 대한 위협 시나리오 예시 정의
- 6장 : 컨테이너 기술의 계획-구현-운영-관리하기 위한 실용적인 정보 제시
- 7장 : 결론
- 부록 A : 컨테이너 기술에서 핵심적이지 않은 컴포넌트의 보안을 위한 NIST의 리소스
- 부록 B : NIST SP 800-53 보안 통제 및 NIST 사이버 보안 프레임워크 하위 카테고리와 애플리케이션 컨테이너 기술과의 관련성
- 부록 C : 약어표
- 부록 D : 용어 정의
- 부록 E : 참고 문헌

2. 애플리케이션 컨테이너에 대한 배경지식

이 장에서는 서버 애플리케이션(앱)을 위한 컨테이너를 소개한다. 첫 번째, 애플리케이션 가상화와 컨테이너에 대한 기본 개념을 정의한다. 이는 문서를 이해하는데 필요하다. 두 번째, 컨테이너와 컨테이너가 실행되는 호스트 OS가 상호 작용하는 방법에 대해 설명한다. 세 번째, 컨테이너 기술의 전반적인 아키텍처에 대해 설명한다. 컨테이너를 구현할 때 일반적으로 사용하는 주요 컴포넌트를 정의하고, 컴포넌트 사이의 워크플로우를 설명한다.

2.1. 애플리케이션 가상화 및 컨테이너에 대한 기본 개념

NIST SP 800-125에서는 가상화를 "다른 소프트웨어 위에서 실행되는 소프트웨어/하드웨어의 시뮬레이션"으로 정의했다.[1] 가상화는 수년간 사용되어 왔고, 클라우드 컴퓨팅을 가능하게 하는 것으로 가장 많이 알려져 있다. 클라우드 환경에서 하드웨어 가상화는 물리 서버 한 대에서 다수의 OS 인스턴스를 실행하고, 각 인스턴스를 개별적으로 유지하기 위해 사용된다. 하드웨어 가상화는 하드웨어를 더 효율적으로 사용할 수 있고, 멀티테넌시[1]를 지원한다.

하드웨어 가상화에서 각 OS 인스턴스는 가상화된 하드웨어와 상호 작용한다. 다른 형태의 가상화는 OS 가상화로 알려져 있으며, 개념은 유사하다. OS 가상화는 한 개의 실제 OS 커널 위에 다수의 가상화된 OS를 만든다. 이러한 접근 방법을 OS 컨테이너라고 부르기도 한다. OS 컨테이너는 2000년대 초반, Solaris Zone과 FreeBSD jails[2]를 시작으로 다양하게 구현되었다. OS 컨테이너를 최초로 지원한 것은 2008년 리눅스의 LXC(Linux Container) 기술이다. LXC 기술은 대부분의 최신 배포판에 탑재되어 있다. OS 컨테이너는 이 문서의 주제인 애플리케이션 컨테이너와 다르다. OS 컨테이너는 일반적인 OS와 유사하게 동작하도록 설계된다. 즉, OS 컨테이너에는 다수의 앱과 서비스가 같이 존재할 수 있다.

최근, 애플리케이션 가상화는 사용 편의성이 향상되고, 개발 신속성(developer agility)에 더욱 주력하면서 인기가 증가했다. 애플리케이션 가상화에서 개별적인 다수의 앱은 동일한 OS 커널을 공유한다. 즉, 공유된 OS 커널은 다수의 앱에 노출된다. OS 컴포넌트는 각 앱 인스턴스를 서버의 다른 앱 인스턴스와 격리한다. 이 경우, 각 앱은 OS와 자신만 볼 수 있다. 또한, 각 앱은 동일한 OS 커널에서 실행될 수 있는 다른 앱과 분리된다.

---

1 멀티테넌시(Multi-tenancy)는 여러 테넌트(tenant, 사용자)를 가진 아키텍처를 의미한다. 많은 사람이 같은 기능을 사용하는 웹메일 서비스가 대표적인 멀티테넌시 아키텍처 소프트웨어이다.[31]

2 https://www.freebsd.org/doc/handbook/jails.html

OS 가상화와 애플리케이션 가상화의 가장 큰 차이점은 애플리케이션 가상화의 경우 일반적으로 각 가상 인스턴스가 앱 하나만 실행한다는 것이다. 현재의 애플리케이션 가상화 기술은 앱을 패키징/실행하기 위해 이식/재사용/자동화할 수 있는 방법을 제공하는데 초점을 맞추고 있다. 애플리케이션 컨테이너 또는 단순히 컨테이너라는 용어는 이러한 기술을 가리키는데 주로 사용된다. 이 용어는 컨테이너를 운송하는 것과 유사하다는 의미이다. 컨테이너는 다른 컨테이너와 서로 분리하면서, 상이한 콘텐츠를 그룹화하는 표준화된 방법을 제공한다.

기존 앱 아키텍처는 앱을 몇 개의 계층(tier)으로 나누고, 각 계층에 서버 또는 VM을 배치한다. 기존 앱 아키텍처와 다르게, 컨테이너 아키텍처는 앱을 더 많은 컴포넌트로 나눈다. 각 컴포넌트는 잘 정의된 기능 하나를 가지며, 일반적으로 독자적인 컨테이너에서 실행된다. 각 앱 컴포넌트는 서로 다른 컨테이너에서 실행된다. 애플리케이션 컨테이너 기술에서 앱을 구성하기 위해 함께 작동하는 컨테이너의 집합을 마이크로서비스라고 한다. 이러한 접근 방식은 앱의 배치에서 유연성과 확장성을 더 좋게 한다. 또한, 기능의 독립성이 더 강하기 때문에, 개발은 더욱 간단하다. 그러나, 안전하게 관리해야 할 개체가 많기 때문에, 앱 관리와 보안 도구/프로세스에서 문제가 발생할 수 있다.

대부분의 애플리케이션 컨테이너 기술은 불변성을 구현한다. 바꾸어 말하면, 컨테이너는 배치되지만 변경되지 않는 스테이트리스(stateless) 엔티티(entity)로 운영되어야 한다.[1] 실행 중인 컨테이너를 업데이트하거나 콘텐츠를 변경해야 할 때, 기존 컨테이너를 폐기하고 업데이트된 새로운 컨테이너로 대체하면 된다. 이를 통해 개발자와 기술 지원 엔지니어는 더 빠른 속도로 앱을 변경하고 적용할 수 있다. 조직에서 분기마다 새로운 버전의 앱을 배포하였다면, 매주 또는 매일 새로운 컴포넌트를 배포할 수 있다. 불변성은 컨테이너와 하드웨어 가상화 사이의 근본적인 운영적 차이점이다. 기존 VM은 일반적으로 배치-재구성-업그레이드되는 스테이트풀(stateful) 엔티티로 실행된다. 레거시 보안 도구/프로세스는 대부분 정적인 운영을 가정한다. 따라서, 컨테이너 환경의 변경 속도(rate of change)에 맞게 조정이 필요할 수 있다.

컨테이너의 불변성은 데이터의 지속에도 영향을 미친다. 컨테이너는 앱-데이터의 혼합보다 분리라는 개념을 강조한다. 데이터를 지속시키기 위해서는 단순히 컨테이너의 루트 파일 시스템에 데이터를 쓰는 것이 아니라, 외부의 영구적인 데이터 저장소(데이터베이스, 클러스터 인식 영구 볼륨 등)를 사용해야 한다. 컨테이너가 사용하는 데이터는 앱이 기존 버전에서 신규 버전으로 대체되었을 때, 신규 버전에서도 모든 데이터를 사용할 수 있도록 컨테이너 외부에 저장해야 한다.

[1] 컨테이너는 불변성을 현실적이고 실용적이게 한다. 하지만, 컨테이너는 불변성을 요구하지 않는다. 조직에서는 불변성의 이점을 얻기 위해 운영 방식을 조정할 필요가 있다는 것에 주의해야 한다.

컨테이너 기술은 DevOps(Development and Operations)와 함께 크게 부각되었다.[1] DevOps는 개발팀과 운영팀 사이의 긴밀한 조율을 강조하면서, 앱의 구축과 실행 간의 통합이 증대되는 것을 추구한다. 컨테이너의 이식성과 선언적 특성은 DevOps에 특히 적합하다. 조직은 컨테이너를 통해 개발-테스트-운영 환경 사이에서 매운 높은 일관성을 가질 수 있기 때문이다. 조직은 빌드 프로세스에서 앱을 컨테이너에 직접 넣는 형태의 지속적 통합(Continuous Integration) 프로세스를 자주 활용한다. 따라서, 애플리케이션 생명 주기의 최초부터 런타임 환경의 일관성이 보장된다. 컨테이너 이미지(컨테이너를 실행하는데 필요한 파일이 들어있는 패키지)는 일반적으로 기기 및 환경에 관계없이 이동할 수 있도록 설계된다. 따라서, 개발팀에서 생성한 이미지를 테스트팀으로 쉽게 이동하고, 운영 환경으로 복사하여 수정 없이 실행할 수 있다. 반면, 컨테이너를 보호하기 위해 사용되는 보안 도구/프로세스는 특정 클라우드 제공자, 호스트 OS, 네트워크 토폴로지와 같이 자주 변경될 수 있는 컨테이너 런타임 환경에 대한 가정을 세우지 말아야 한다. 더욱 중요한 것은 이러한 모든 환경과 앱 생명 주기 전반(개발-테스트-운영)에 걸쳐 보안이 일관되어야 한다는 것이다.

최근, 도커(Docker)[2]와 rkt[3]와 같은 프로젝트는 OS 컴포넌트 분리 기능을 더 쉽게 사용하고 확장할 수 있도록 설계된 추가 기능을 제공한다. 컨테이너 기술은 윈도우 플랫폼(윈도우 서버 2016 이후)에서도 이용할 수 있다. 모든 컨테이너 기술의 기본적인 아키텍처는 일관성이 있다. 따라서, 이 문서는 컨테이너 기술의 종류에 관계없이 컨테이너에 대해 상세하게 논의할 수 있다.

2.2. 컨테이너 및 호스트 OS

하드웨어 가상화 기술인 VM(Virtual Machine, 가상 머신) 내부에 앱을 배치하는 것과 비교하면, 컨테이너 내부에 앱을 배치하는 것을 쉽게 설명할 수 있다. 그림 2는 왼쪽부터 VM에 배치, 베어메탈에 설치(VM 없이 컨테이너에 배치), VM 내부에 컨테이너 배치를 보인다.

그림 2 : VM 및 컨테이너 배치

[1] 이 문서에서는 DevOps가 수행하는 작업에 대해 언급한다. DevOps에 대해서는 직함/조직 구조가 아닌 수행되는 직무에 초점을 맞추었다.

VM과 컨테이너는 다수의 앱이 동일한 물리적 인프라를 공유할 수 있게 한다. VM과 컨테이너는 분리를 위해 사용하는 방법이 다른다. VM은 하이퍼바이저(hypervisor)를 사용한다. 하이퍼바이저는 하드웨어 레벨에서 VM 간의 리소스를 분리한다. 각 VM은 자체 가상 하드웨어를 갖고, 완전한 게스트 OS/앱/데이터가 포함된다. VM은 상이한 OS(리눅스, 윈도우 등)가 동일한 물리적 하드웨어를 공유할 수 있게 한다.

컨테이너에 배치된 앱은 동일한 OS 커널을 공유하지만, 서로 분리된다. OS 커널은 호스트 OS의 일부분이다. 호스트 OS는 컨테이너 하위에 위치하며, 컨테이너에 OS의 기능을 제공한다. 컨테이너는 OS마다 다르다. 리눅스 호스트는 리눅스용으로 제작된 컨테이너만 실행할 수 있고, 윈도우 호스트는 윈도우 컨테이너만 실행할 수 있다. 또한, 한 가지 OS 제품군으로 제작된 컨테이너는 해당 제품군의 모든 최신 OS에서 실행해야 한다.

컨테이너를 실행하는 데 사용되는 호스트 OS는 두 가지로 분류할 수 있다. 범용 OS(레드햇 리눅스, 우분투, 윈도우 서버 등)는 여러 종류의 앱을 실행하는데 사용할 수 있고, 컨테이너 전용 기능을 추가할 수 있다. 컨테이너 전용 OS(CoreOS 컨테이너 리눅스[4], 프로젝트 아토믹[5], 구글 컨테이너 최적화 OS[6] 등)는 컨테이너만 실행하도록 명시적으로 설계된 최소화된 OS이다. 컨테이너 전용 OS에는 일반적으로 패키지 매니저가 없다. 컨테이너 전용 OS는 관리자 도구의 일부만 가지고 있으며, 컨테이너 외부에서 앱이 실행되는 것을 적극 차단한다. 컨테이너 전용 OS는 공격자가 파일 시스템에 데이터를 저장할 가능성을 줄이기 위해 읽기 전용으로 설계하는 경우가 많다. 컨테이너 전용 OS는 앱 호환성과 관련된 문제가 거의 없기 때문에, 업그레이드 프로세스가 단순하다.

컨테이너 실행에 사용되는 모든 호스트 OS는 각 컨테이너 환경을 설정하고 유지하는 바이너리가 있다. 이 바이너리를 컨테이너 런타임이라고 한다. 컨테이너 런타임은 리소스를 분리하는 다수의 OS 컴포넌트를 조정한다. 컨테이너 런타임은 각 컨테이너마다 OS에 대한 전용 뷰를 갖게 하고, 실행 중인 다른 컨테이너와 분리하기 위해 리소스를 분리하는 다수의 OS 컴포넌트를 조정한다. 컨테이너와 호스트 OS는 컨테이너 런타임을 통해 효과적으로 상호 작용한다. 또한, 컨테이너 런타임은 DevOps 직원 등이 특정 호스트에서 컨테이너가 실행되는 방법을 지정할 수 있도록 관리 도구와 API(Application Programming Interface)를 제공한다. 컨테이너 런타임은 설정이 쉽고, 컨테이너를 시작-중지-작동하는 프로세스를 단순하게 한다. 컨테이너 런타임의 예로 도커[2], rkt[3], OCI(Open Container Initiative) 데몬[7] 등이 있다.

호스트 OS는 컨테이너 런타임에 다음과 같은 기능을 제공해야 한다.

- "네임스페이스 분리"는 컨테이너와 상호 작용하는 리소스를 제한하는 기능이다. 여기에는 파일 시스템, 네트워크 인터페이스, 프로세스 간 통신, 호스트 이름, 사용자 정보 및 프로세스가 포함된다. 네임스페이스 분리는 컨테이너 내부의 앱/프로세스가 컨테이너에 할당된 물리/가상 리소스만 볼 수 있게 한다. 예를 들어, 서로 다른 앱이 실행되고 있는 다수의 컨테이너가 있는 호스트의 경우, 아파치(Apache)가 실행되는 컨테이너 내부에서 "ps –A"를 실행하면, 결과로 httpd만 보일 것이다. 네임스페이스 분리는 각 컨테이너마다 자체 네트워크 스택(고유한 인터페이스와 IP 주소를 포함)을 제공한다. 리눅스의 컨테이너는 네임스페이스 분리를 위해 프로세스 ID 마스킹과 같은 기술을 사용한다. 반면, 윈도우에서는 객체 네임스페이스를 사용한다.

- "리소스 할당"은 특정 컨테이너가 사용할 수 있는 호스트 리소스의 양을 제한하는 기능이다. 예를 들어, 호스트 OS의 전체 메모리가 10GB라면, 9개의 컨테이너 각각에 1GB씩 할당할 수 있다. 어떤 컨테이너도 다른 컨테이너의 운영을 방해할 수 있어서는 안된다. 따라서, 리소스 할당은 각각의 컨테이너가 컨테이너에 지정된 리소스의 양만큼만 사용할 수 있게 한다. 리눅스에서는 리소스 할당을 위해 컨트롤 그룹(cgroups)[1]을 주로 사용한다. 반면, 윈도우에서는 유사한 목적으로 잡 객체(Job Objects)를 사용한다.

- "파일 시스템 가상화"는 다수의 컨테이너가 동일한 물리적 스토리지를 공유하면서, 다른 컨테이너의 스토리지에 접근하거나 변경할 수 없게 한다. 파일 시스템 가상화는 네임스페이스 가상화와 거의 유사하다. 하지만, 파일 시스템 가상화는 COW(Copy-on-Write, 쓰기 시 복사)와 같은 기술을 통해 컨테이너가 호스트 스토리지의 최적화(효율적 사용을 보장)를 수반하는 경우가 많기 때문에 별도로 구분한다. 예를 들어, 호스트 한 대에서 동일한 이미지를 사용하는 다수의 컨테이너가 아파치를 실행하는 경우, 파일 시스템 가상화는 디스크에 httpd 바이너리 파일이 하나만 저장되어 있게 한다. 컨테이너 한 개에서 파일을 변경하는 경우, 변경된 비트(bit)만 디스크에 기록될 것이며, 변경 내용은 변경을 수행한 컨테이너에서만 보일 것이다. 리눅스에서는 파일 시스템 가상화를 위해 AUFS(Advanced Multi-Layered Unification Filesystem)와 같은 기술이 사용된다. 반면, 윈도우에서는 NTFS(NT File System)를 확장하여 사용한다.

컨테이너의 기능은 호스트 OS 종류에 따라 다르다. 컨테이너는 기본적으로 각 앱마다 한 가지 OS의 고유한 뷰를 제공하는 메커니즘이다. 따라서, 이러한 분리를 위한 도구는 대부분 OS 종류에 의존적이다. 예를 들어, 리눅스와 윈도우는 프로세스를 격리하는데 다른 방법을 사용한다. 기본적인 구현에는 차이가 있을 수 있지만, 일반적으로 컨테이너 런타임은 사용자가 이러한 차이와 거의 무관하게 사용할 수 있는 공통적인 인터페이스를 제공한다.

---

[1] 컨트롤 그룹(cgroup)은 독립적으로 관리할 수 있는 프로세스의 모음으로, 커널에 메모리/프로세서 사용량 및 디스크 I/O와 같은 하위 시스템을 측정할 수 있는 소프트웨어 기반의 기능을 부여한다. 관리자는 수동 또는 프로그래밍 방식으로 이러한 하위 시스템을 제어할 수 있다.

컨테이너에 의한 분리는 강력하지만, 보안의 경계가 VM만큼 명확하고 구체적이지 않다. 컨테이너는 동일한 커널을 공유하고, 호스트에서 다른 권한/기능으로 실행될 수 있다. 즉, 컨테이너 사이의 분리는 하이퍼바이저에 의한 VM 사이의 분리보다 훨씬 낮은 수준이다. 따라서, 환경이 부주의하게 구성되면, 컨테이너는 VM보다 호스트 및 다른 컨테이너와 훨씬 더 쉽고 직접적으로 상호 작용할 수 있다.

컨테이너는 하드웨어 가상화를 뛰어넘은, 가상화의 다음 단계로 간주되는 경우가 많다. 하지만, 대부분의 조직에서 컨테이너가 사용되는 사례를 보면, 급격한 변화보다 점진적인 변화가 많다. 컨테이너와 하드웨어 가상화는 서로 잘 공존할 수 있고, 공존하는 사례가 매우 많으며, 실제로 서로의 기능을 강화한다. VM은 강력한 분리 및 OS 자동화와 같은 장점이 있으며, 솔루션의 생태계가 넓고 깊다. 조직은 컨테이너와 VM 사이에서 하나를 선택할 필요가 없다. 대신, 하드웨어를 배치/파티셔닝/관리하는데 VM을 계속 사용하면서, 애플리케이션을 패키징하고 VM을 더욱 효과적으로 사용하는데 컨테이너를 사용할 수 있다.

2.3. 컨테이너 기술 아키텍처

그림 3에서는 컨테이너 기술 아키텍처의 5계층을 보인다.

1. 개발 시스템 : 이미지를 생성하고, 테스트/승인 시스템으로 이미지를 전송한다.
2. 테스트/승인 시스템 : 이미지의 콘텐츠를 확인/검증하고, 이미지를 서명하고, 이미지를 레지스트리에 전송한다.
3. 레지스트리 : 이미지를 저장하고, 요청 시 이미지를 오케스트레이터에 배포한다.
4. 오케스트레이터 : 이미지를 컨테이너로 변환하고, 호스트에 컨테이너를 배치한다.
5. 호스트 : 오케스트레이터의 지시에 따라 컨테이너를 실행 및 정지한다.

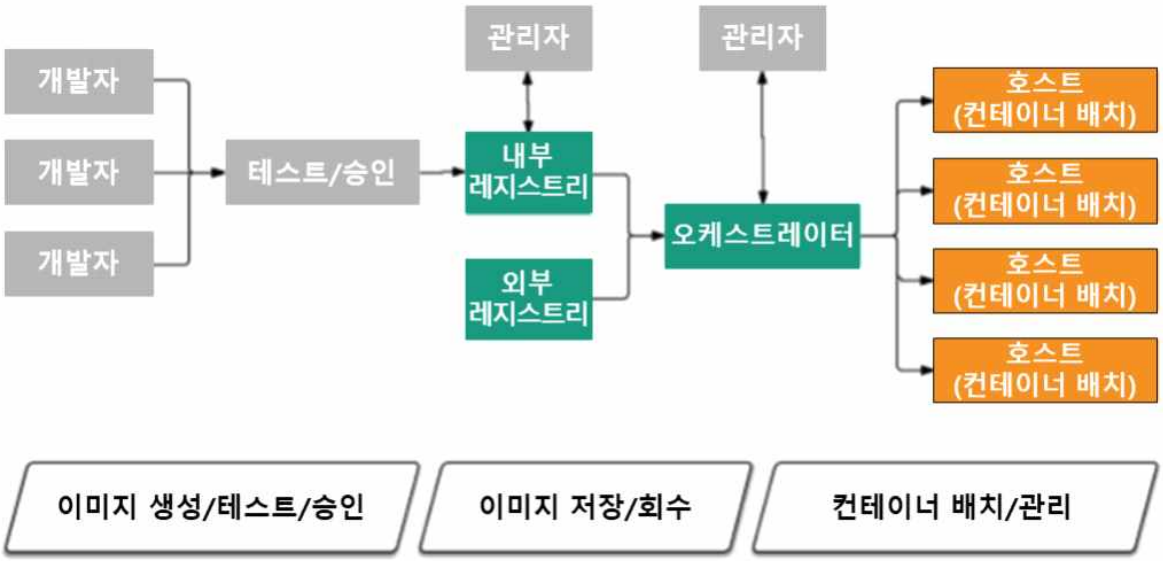


그림 3 : 컨테이너 기술 아키텍처의 계층, 컴포넌트, 생명 주기

전체 프로세스에는 관리 시스템이 더 많이 있다. 하지만, 이 그림에서는 내부 레지스트리 및 오케스트레이터와 관련된 관리 시스템만 표시했다.

회색으로 표시된 시스템(개발 시스템, 테스트/승인 시스템, 관리 시스템)은 컨테이너 기술 아키텍처의 범위 밖에 있지만, 컨테이너 기술 아키텍처와 중요한 상호 작용을 한다. 컨테이너를 사용하는 대부분의 조직에서는 개발 및 테스트 환경도 컨테이너를 활용한다. 이러한 일관성은 컨테이너를 사용하는 주요한 장점이다. 이 문서는 개발 및 테스트 환경의 시스템에 초점을 두고 있지 않다. 개발 및 테스트 환경에서의 보안 권고사항은 운영 환경에서의 권고사항과 대체로 동일하기 때문이다. 녹색으로 표시된 시스템(내부 레지스트리, 외부 레지스트리, 오케스트레이션)은 컨테이너 기술 아키텍처의 핵심 컴포넌트이다.

컨테이너 기술 아키텍처를 이해하기 위해 그림 3의 하단에 설명된 컨테이너 생명 주기를 검토할 수도 있다. 생명 주기의 3단계는 아래에서 더 자세히 설명한다.

조직은 일반적으로 다수의 다양한 앱을 빌드하고 배치하기 때문에, 생명 주기의 각 단계가 동시에 발생하는 경우가 많다. 따라서, 생명 주기의 단계가 올라감에 따라 성숙도가 올라가는 것으로 간주하지 말아야 한다. 대신, 생명 주기의 단계는 지속적으로 작동하는 엔진에서의 사이클로 생각해야 한다. 각 앱은 엔진 내부의 실린더로 비유할 수 있다. 각 앱은 같은 시간에 생명 주기의 다른 단계에 있을 수 있다.

2.3.1. 이미지 생성, 테스트, 승인

컨테이너 생명 주기의 첫 단계에서, 앱의 컴포넌트를 빌드하고 한 개 또는 여러 개의 이미지에 배치한다. 이미지는 컨테이너를 실행하는 데 필요한 모든 파일을 포함하는 패키지이다. 예를 들어, 아파치가 실행되는 이미지에는 관련 라이브러리 및 설정 파일, `httpd` 바이너리가 포함된다. 이미지에는 앱에 필요한 실행 파일과 라이브러리만 포함해야 한다. 다른 모든 OS 기능은 호스트 OS의 커널에서 제공한다. 이미지는 용량을 최소화하고 운영 효율성을 향상시키기 위해 레이어링(layering)과 COW(마스터 이미지는 읽기 전용, 변경 내용은 별도 파일에 기록)와 같은 기술을 주로 사용한다.

이미지는 레이어로 빌드되며, 다른 모든 컴포넌트가 추가된 기본 레이어를 "베이스 레이어"라고 부른다. 베이스 레이어는 일반적으로 OS 커널이 없는 일반적인 OS(우분투, 윈도우 나노 서버 등)의 최소화된 배포판이다. 사용자는 베이스 레이어에 애플리케이션 프레임워크와 커스텀 코드(특정 앱을 배포 가능한 이미지로 개발하기 위한 코드)를 추가함으로써 완전한 이미지를 빌드하기 시작한다. 컨테이너 런타임은 동일한 OS 제품군이라면 OS 버전이 달라도 이미지를 사용할 수 있다. 예를 들어, 도커를 실행하는 레드햇 호스트는 모든 리눅스 베이스 레이어(알파인, 우분투 등)에서 생성한 이미지를 실행할 수 있지만, 윈도우 베이스 레이어로 생성한 이미지를 실행할 수 없다.

이미지 생성 프로세스는 테스트로 전달하기 위해 앱을 패키징하는데 책임이 있는 개발자가 관리한다. 이미지 생성은 일반적으로 젠킨스(Jenkins)[8] 및 팀시티(TeamCity)[9]와 같은 빌드 관리 및 자동화 도구를 사용하며, 이러한 도구는 지속적 통합 프로세스를 지원한다. 빌드 관리 및 자동화 도구는 앱의 다양한 라이브러리/바이너리/컴포넌트를 가져와서 테스트하고, 앱의 이미지를 빌드하는 방법을 설명한 매니페스트(개발자가 생성)에 따라 이미지를 조립한다.

대부분의 컨테이너 기술은 앱의 컴포넌트 및 요구사항을 설명하는 선언적인 방법을 가지고 있다. 예를 들어, 웹 서버 이미지에는 웹 서버의 실행 파일뿐만 아니라 웹 서버가 어떻게 실행되어야 하는지 설명하는 데이터(리스닝 포트, 설정 파라미터 등)가 포함된다.

이미지를 생성한 후 조직에서는 일반적으로 테스트와 승인을 수행한다. 예를 들어, 테스트 담당자 및 테스트 자동화 도구는 애플리케이션 기능을 확인하기 위해 빌드된 이미지를 사용한다. 보안팀은 테스트한 이미지에 대한 승인을 수행한다. 빌드/테스트/승인에서 사용되는 앱의 아티팩트는 완전히 동일하며, 이러한 일관성은 운영/보안에서 컨테이너의 주요한 이점 중 하나이다.

2.3.2. 이미지 저장, 회수

이미지는 일반적으로 호스트에서 쉽게 접근하고, 공유하고, 검색하고, 재사용할 수 있도록 중앙 위치에 저장된다. 레지스트리는 개발자가 이미지를 생성할 때 쉽게 저장할 수 있도록 하고, 식별 및 버전 관리를 위해 이미지에 태그/카탈로그를 지정하여 검색/재사용을 지원하고, 다른 사람이 생성한 이미지를 검색하고 다운로드할 수 있는 서비스이다. 레지스트리는 자체적으로 호스팅하거나, 서비스를 이용할 수 있다. 레지스트리의 예로는 아마존 EC2 컨테이너 레지스트리(Amazon EC2 Container Registry)[10], 도커 허브(Docker Hub)[11], 도커 트러스티드 레지스트리(Docker Trusted Registry)[12], 콰이 컨테이너 레지스트리(Quay Container Registry)[13] 등이 있다.

레지스트리는 이미지와 관련된 일반적인 작업을 자동화할 수 있는 API를 제공한다. 예를 들어, 조직에서는 이미지 생성 단계에서 테스트를 통과하면 이미지를 레지스트리에 자동으로 저장하게 할 수 있다. 또한, 새로운 이미지가 추가되면 자동으로 배치되도록 할 수 있다. 이러한 자동화는 프로젝트에 일관된 결과 및 빠른 반복을 제공한다.

이미지가 레지스트리에 저장되면, 이미지를 쉽게 가져올 수 있고, DevOps 담당자는 컨테이너가 실행되는 모든 환경에서 이미지를 실행할 수 있다. 공용(public) 클라우드에서 이미지를 생성하고, 사설(private) 클라우드에 레지스트리를 호스팅하여 이미지를 저장하고, 제3의 위치에서 앱을 실행하기 위해 이미지를 배포할 수 있다. 이는 컨테이너가 갖는 이식성의 이점을 보여주는 사례이다.

2.3.3. 컨테이너 배치, 관리

오케스트레이터는 DevOps 담당자 또는 자동화된 작업이 레지스트리에서 이미지를 가져오고, 컨테이너에 이미지를 배치하고, 실행 중인 컨테이너를 관리할 수 있게 해주는 도구이다. 이러한 배치 프로세스는 실제 사용 가능한 버전의 앱을 실행하고, 요청에 응답할 준비를 한다. 이미지를 컨테이너를 배치했을 때, 이미지 자체는 변하지 않는다. 대신, 이미지의 복사본을 컨테이너에 두고, 앱 코드의 휴면 상태에서 앱 인스턴스의 실행 상태로 전환한다. 오케스트레이터의 예로는 쿠버네티스(Kubernetes)[14], 메소스(Mesos)[15], 도커 스웜(Docker Swarm)[16]이 있다.

소규모의 간단한 컨테이너를 구현하는 경우, 완전한 형태의 오케스트레이터는 생략할 수 있다. 또한, 오케스트레이터가 불필요하거나, 오케스트레이터의 사용을 제한해야 상황도 있다. 예를 들어, 호스트는 레지스트리에서 이미지를 가져오기 위해 레지스트리에 직접 접근할 수도 있다. 이 문서에서는 논의를 단순화하기 위해 오케스트레이터의 사용을 가정한다.

오케스트레이터가 제공하는 추상화는 DevOps 담당자가 특정 이미지로 실행해야 하는 컨테이너의 수, 각 이미지에 할당해야 하는 자원(메모리, 프로세스, 디스크 등)을 간단하게 명시할 수 있게 해준다. 오케스트레이터는 클러스터 내 각 호스트의 상태(각 호스트에서 사용할 수 있는 리소스 등)를 인지하고 있으며, 컨테이너가 실행될 호스트를 결정한다. 오케스트레이터는 레지스트리에서 이미지를 가져오고, 이미지를 지정된 리소스를 갖는 컨테이너로 실행한다.

오케스트레이터는 호스트에서 실행 중인 컨테이너의 리소스 사용량/작업 실행/기기 상태를 모니터링한다. 컨테이너가 최초로 실행된 호스트에 장애가 발생하면, 자동으로 새로운 호스트에서 컨테이너를 재시작하도록 오케스트레이터를 설정할 수 있다. 많은 오케스트레이터가 다른 호스트에서 실행 중인 컨테이너 사이의 네트워킹(cross-host container networking)과 서비스 검색(service discovery)을 지원한다. 또한, 대부분의 오케스트레이터는 오버레이(overlay) 네트워크로 알려진 SDN(Software-defined Networking, 소프트웨어 정의 네트워킹) 컴포넌트를 포함하고 있다. 오버레이 네트워크는 물리적 네트워크를 공유하는 앱 사이의 통신을 분리할 수 있다.

컨테이너에서 앱의 업데이트가 필요한 경우, 기존 컨테이너를 변경하지 않는다. 기존 컨테이너는 소멸되고, 업데이트된 이미지로 새로운 컨테이너를 생성한다. 이는 컨테이너의 운영에 있어 주요한 차이점이다. 최초에 배치된 베이스라인 소프트웨어는 시간이 지나도 변경되지 않아야 하며, 업데이트는 전체 이미지를 한 번에 교체함으로써 수행된다. 이러한 접근 방법은 보안에 중요한 잠재적 이점을 제공한다. 조직은 완전히 동일한 설정으로 완전히 동일한 소프트웨어를 빌드/테스트/승인/배치할 수 있기 때문이다. 조직에서는 업데이트된 최신 버전의 앱 사용을 보장하기 위해 오케스트레이터를 활용할 수 있다. 앱을 항상 최신 상태로 유지하기 위해, 레지스트리에서 이미지의 최신 버전을 가져오도록 오케스트레이터를 구성한다. 이러한 "지속적 제공(continuous delivery)" 자동화는 개발자가 새로운 버전의 앱 이미지를 쉽게 빌드하고, 이미지를 테스트하고, 레지스트리에 이미지를 저장하고, 자동화 도구가 대상 환경에 이미지를 배치할 수 있게 해준다.

이는 새로운 버전의 이미지를 빌드할 때, 모든 취약점 관리(패치, 설정 등)를 개발자가 처리한다는 것을 의미한다. 컨테이너를 사용하면, 앱과 이미지의 보안은 주로 개발자(운영팀 대신)에게 책임이 있다. 책임과 관련한 이러한 변화는 이전에 필요했던 것보다 담당자 사이에서 더 많은 조정과 협력이 필요하다. 컨테이너를 채택하는 조직은 반드시 각 이해관계자에게 명확한 프로세스 흐름(process flaw)과 책임이 확립되도록 해야 한다.

컨테이너 관리는 보안 관리와 모니터링을 포함한다. 그러나, 非 컨테이너 환경의 보안 통제는 컨테이너 환경에서 사용하기에 적절하지 않는 경우가 많다. 예를 들어, IP 주소를 계정으로 사용하는 보안 통제는 고정 IP 주소(수개월 또는 수년 동안 동일한 IP를 유지)를 사용하는 VM 및 베어메탈 서버에 대해 유효하다. 이와 대조적으로 컨테이너는 일반적으로 오케스트레이터가 IP 주소를 할당한다. 컨테이너의 생성과 소멸은 VM보다 훨씬 더 자주 일어나기 때문에 IP 주소도 자주 변경된다. 따라서, 고정 IP 주소 기반의 보안 기술(IP 주소를 기반으로 트래픽을 필터링하는 방화벽 정책 등)을 사용하여 컨테이너를 보호하는 것은 어렵거나 불가능하다. 추가적으로 컨테이너 네트워크에는 동일한 노드, 다른 노드, 심지어 클라우드에 위치한 컨테이너 사이의 통신까지 포함된다.

2.4. 컨테이너 사용

다른 기술과 마찬가지로 컨테이너가 모든 문제의 해결책이 될 수는 없다. 컨테이너는 많은 상황에서 중요한 도구이지만, 모든 상황에서 최선의 선택은 아니다. 예를 들어, 기성 소프트웨어를 많이 사용하는 조직에서는 벤더의 지원 여부에 따라 컨테이너를 이용하지 못할 수 있다. 그러나, 대부분의 조직에서는 컨테이너를 여러 가지 중요한 용도로 사용할 것이다. 그 사례는 다음과 같다.

- 애자일 개발(agile development) : 앱을 자주 업데이트하고 배치한다. 컨테이너의 이식성 및 선언적 특성은 잦은 업데이트에 대한 테스트를 더 효율적이고 더 쉽게 한다. 이로 인해 조직은 혁신을 가속화하고, 소프트웨어를 더 신속하게 제공할 수 있다. 또한, 앱 코드 內 취약점을 수정하고, 업데이트된 소프트웨어를 더 신속하게 테스트하고 배치할 수 있다.

- 환경의 일관성 및 분리 : 개발자는 앱을 빌드-테스트-실행 환경을 각각 분리하여 가질 수 있다. 컨테이너는 개발자가 운영 중인 앱의 정확한 복사본 전체를 개발 시스템(로컬)에서 실행할 수 있게 한다. 따라서, 컨테이너는 테스트 환경 구성의 비효율성과 번거로움을 없앤다. 또한, 테스트 환경을 공유하거나 조정할 필요도 없다.

- 확장(scale out) : 앱은 특정 시점의 부하에 따라 신속하게 새로운 인스턴스 여러 개를 배치하거나 해제해야 할 필요가 있다. 컨테이너의 불변성은 인스턴스의 신뢰성 있는 확장을 용이하게 한다. 각 인스턴스가 다른 인스턴스와 정확히 일치하기 때문이다.

- 클라우드 네이티브 앱 : 개발자는 처음부터 마이크로서비스 아키텍처를 목적으로 빌드할 수 있다. 앱 운영에 필요한 반복 작업의 효율을 높이고, 배치를 단순하게 하기 때문이다.

컨테이너의 추가적인 이점은 컨테이너 이미지의 불변성 덕분에 배치 파이프라인[1]의 효율성을 증대시킬 수 있다는 것이다. 컨테이너는 설치된 운영 코드를 그대로 옮길 수 있다. 非 컨테이너 환경에서 앱은 운영 서버에 설치된다. 서버에 앱 코드(예 : 프로그래밍 언어 런타임, 서드 파티 라이브러리, 초기화 스크립트, OS 도구)를 설치하기 위해서는 일반적으로 이를 위해 작성된 스크립트를 실행한다. 즉, 모든 테스트[2]에서 사용되는 코드는 실제 운영 코드가 아니라, 빌드 시스템에서 가장 비슷할 것으로 추정되는 코드이다. 이러한 차이는 시간이 지날수록 벌어지며, 특히 운영 시스템과 빌드 시스템이 다른 경우 더욱 깊어진다. 이는 "내 기기에서는 잘 동작합니다(It works on my machine)."라는 전형적인 문제이다.

컨테이너 기술의 경우, 빌드 시스템은 이미지 내부에 앱을 설치한다. 이미지는 앱의 모든 요구사항(예 : 프로그래밍 언어 런타임, 서드 파티 라이브러리, 초기화 스크립트, OS 도구)이 설치된 변하지 않는 스냅샷(snapshot)이다. 운영 환경에서는 빌드 시스템에서 생성된 컨테이너 이미지를 쉽게 다운로드하고 실행한다. 따라서, "내 기기에서는 잘 동작합니다."의 문제를 해결할 수 있다. 개발자/빌드 시스템/운영 환경이 모두 동일한 이미지를 실행하기 때문이다.

또한, 최신 컨테이너 기술은 재사용을 자주 강조한다. 한 개발자가 생성한 컨테이너 이미지를 쉽게 공유할 수 있고, 조직 내/외부의 다른 개발자가 쉽게 재사용할 수 있다. 레지스트리 서비스는 중앙화된 이미지 공유/검색 서비스를 제공하며, 이는 개발자가 이미지를 검색/재사용하는 것을 용이하게 한다. 이러한 편의성은 많은 소프트웨어 벤더/프로젝트가 컨테이너를 사용하도록 이끌고 있다. 즉, 컨테이너를 사용하여, 고객이 소프트웨어를 쉽게 찾아 빠르게 실행할 수 있게 해준다. 예를 들어, 사용자는 MongoDB와 같은 앱을 호스트 OS에 직접 설치하는 것 대신, 간단하게 MongoDB의 컨테이너 이미지를 실행할 수 있다. 컨테이너 런타임은 컨테이너를 다른 컨테이너와 호스트 OS로부터 분리하기 때문에 앱을 더 안전하고 안정적으로 실행할 수 있다. 또한, 사용자는 앱이 호스트 OS를 어지럽히는 것을 걱정할 필요가 없다.

1 원문에서는 빌드 파이프라인(build pipeline)이라는 용어를 사용했으나, 배치 파이프라인(deployment pipeline)이라는 용어로 더 자주 사용된다.

2 개발자의 워크스테이션에서 실시하는 테스트를 포함하여 배치 파이프라인에서 운영 단계 이전에 실시하는 모든 테스트를 의미한다.

3. 컨테이너 기술의 핵심 컴포넌트에 대한 주요 위험

이 장에서는 컨테이너 기술의 핵심 컴포넌트(이미지, 레지스트리, 오케스트레이터, 컨테이너, 호스트 OS)에 대한 주요 위험을 식별하고 분석한다. 핵심 컴포넌트만 분석하기 때문에 컨테이너 기술, 호스트 OS 플랫폼 또는 위치(공개 클라우드, 사설 클라우드 등)에 관계없이 대부분의 컨테이너를 배치할 때 적용된다. 여기서는 두 가지 유형의 위험을 고려한다.

1. **이미지 또는 컨테이너 침해** : 데이터 중심 시스템 위협 모델링 방법론(NIST SP 800-154)[17]을 사용하여 위험을 평가한다. 기본적으로 보호해야 할 데이터는 앱 파일, 데이터 파일 등이 저장된 이미지와 컨테이너이다. 이차적으로 보호해야 할 데이터는 호스트 공유 리소스(메모리, 스토리지, 네트워크 인터페이스 등) 내부의 컨테이너 데이터이다.

2. 다른 컨테이너, 호스트 OS, 다른 호스트 등을 공격할 수 있는 **컨테이너의 오용**

핵심 컴포넌트와 관련된 다른 유형의 위험, 非 핵심 컴포넌트(개발/테스트/승인 시스템, 관리자 시스템, 호스트 하드웨어, VM 매니저)와 관련된 모든 위험은 이 문서의 범위를 벗어난다. 부록 A에는 非 핵심 컴포넌트의 보안에 대해 일반적으로 참조할 수 있는 문서를 수록했다.

3.1. 이미지에 대한 위험

3.1.1. 이미지 취약점

이미지는 특정 앱을 실행하는데 사용되는 모든 컴포넌트가 수록된 사실상의 정적 아카이브 파일이다. 따라서, 이미지 內 컴포넌트는 중요한 보안 업데이트가 누락되었거나 오래된 버전일 수 있다. 완전하게 최신 컴포넌트로 생성한 이미지는 이미지를 생성한 후 수일 또는 수주 동안은 알려진 취약점이 없을 것이다. 하지만, 언젠가는 하나 이상의 컴포넌트에서 취약점이 발생할 것이며, 이미지는 더 이상 최신이 아니게 될 것이다.

소프트웨어 업데이트는 소프트웨어가 실행되는 호스트에서 수행하는 것이 전통적인 운영 패턴이다. 이와 다르게, 컨테이너 환경에서는 이미지에서 업데이트를 수행해야 하며, 그 이후 다시 배포된다. 따라서, 컨테이너 환경에서 흔하게 발생하는 위험은 취약점이 있는 이미지를 사용하여 취약점이 있는 컨테이너가 배치되는 것이다.

3.1.2. 이미지 설정 결함

이미지에는 소프트웨어 결함 이외에 설정 결함도 있을 수 있다. 예를 들어, 이미지에는 다른 계정의 권한을 사용(sudo, runas 등)할 수 있는 계정을 제한하지 않을 수 있다. 이 경우, 권한이 남용될 수 있다. 이미지에 SSH 데몬이 포함되어 있는 경우, 컨테이너를 네트워크 관련 위험에 불필요하게 노출시킬 수 있다. 전통적인 서버 또는 VM의 경우, 설정이 잘못되면 알려진 취약점이 없더라도 최신 시스템을 공격에 노출될 수 있다. 이와 마찬가지로 이미지의 설정이 잘못되면, 이미지에 포함된 컴포넌트가 최신이라도 위험이 증가할 수 있다.

3.1.3. 악성코드 포함

이미지는 단지 파일을 모아 함께 패키징한 것이다. 따라서, 의도적 또는 비의도적으로 악의적인 파일이 이미지에 포함될 수 있다. 이러한 악성코드는 이미지 내 다른 컴포넌트와 같은 능력을 갖는다. 따라서, 컨테이너 환경의 다른 컨테이너나 호스트를 공격하는데 사용될 수 있다. 악성코드는 서드 파티(third party)의 베이스 레이어 및 이미지를 철저히 검증하지 않고 사용하는 경우에 포함될 수 있다.

3.1.4. 평문으로 저장된 기밀(secret)

많은 앱은 다른 컴포넌트와 안전하게 통신하기 위해 기밀이 필요하다. 예를 들어, 웹 앱은 백엔드 데이터베이스에 접속하기 위해 아이디와 패스워드가 필요할 수 있다. 평문으로 저장된 비밀의 다른 사례로는 데이터베이스 연결 문자열, SSH 개인키, X.509 개인키가 있다. 앱을 이미지로 패키징하면서, 이러한 기밀이 이미지의 파일 시스템에 포함될 수 있다. 그러나, 이러한 상황은 보안 위험이 된다. 이미지에 접근할 수 있는 사람은 누구나 이미지를 분석하여 기밀을 획득할 수 있기 때문이다.

3.1.5. 신뢰할 수 없는 이미지 사용

신뢰할 수 없는 소프트웨어를 실행하는 것은 모든 환경에서 발생할 수 있는 가장 일반적이고 위험도가 높은 시나리오 중의 하나이다. 컨테이너는 이식과 재사용이 용이하기 때문에 팀에서는 조직 외부에서 획득한 확인되지 않거나 신뢰할 수 없는 이미지를 실행하려는 유혹에 빠지기 쉽다. 예를 들어, 웹 앱에서 발생한 문제를 해결하기 위해, 사용자는 해당 웹 앱의 다른 버전이 포함된 서드 파티의 이미지를 찾을 수 있다. 이렇게 외부에서 제공된 이미지를 사용하는 것은 외부 소프트웨어 사용과 동일한 유형의 위험(악성코드 유입, 데이터 유출, 취약한 컴포넌트 사용 등)을 초래한다.

3.2. 레지스트리에 대한 위험

3.2.1. 레지스트리에 대한 안전하지 않은 연결

이미지에는 민감한 컴포넌트(조직의 특허 소프트웨어 등)와 기밀이 포함되는 경우가 많다. 만약, 안전하지 않은 채널을 통해 레지스트리와 연결된다면, 평문으로 전송되는 다른 모든 데이터와 마찬가지로 이미지 콘텐츠의 기밀성을 위험하게 한다. 또한, 중간자 공격(man-in-the-middle attack)의 위험이 증가한다. 중간자 공격은 레지스트리로 향하는 네트워크 트래픽을 가로채고, 트래픽에서 개발자 또는 관리자의 크리덴셜을 훔치고, 오케스트레이터로 악의적이거나 노후된 이미지를 전송할 수 있다.

3.2.2. 레지스트리의 노후된 이미지

조직에서는 일반적으로 레지스트리를 통해 모든 이미지를 배포한다. 따라서, 시간이 지나면 레지스트리에 취약하거나 노후된 버전의 이미지가 많이 저장되어 있을 수 있다. 취약한 이미지가 레지스트리에 저장되어 있는 것만으로는 직접적인 위협을 가하지는 않는다. 하지만, 취약한 이미지가 우발적으로 배치될 가능성이 높아진다.

3.2.3. 불충분한 인증 및 인가

레지스트리에는 민감하거나 특허를 보유한 앱을 실행하기 위해 사용되는 이미지, 민감한 데이터에 접근하기 위해 사용되는 이미지가 저장되어 있을 수 있다. 따라서, 불충분한 인증/인가는 지적 재산의 손실을 초래하거나, 앱과 관련한 중요 기술을 공격자에게 노출시킬 수 있다. 일반적으로 레지스트리에는 유효하고 검증된 소프트웨어가 저장되어 있을 것이라고 믿는다. 따라서, 레지스트리가 침해되면, 이미지를 사용하는 컨테이너 및 호스트가 침해될 수 있다.

3.3. 오케스트레이터에 대한 위험

3.3.1. 제한되지 않은 관리자 접근 Unbounded administrative access

많은 오케스트레이터는 “오케스트레이터의 사용자는 모두 관리자이며, 관리자는 전체 환경에 대한 통제권을 가져야 한다”는 가정으로 설계되었다. 그러나, 오케스트레이터는 서로 다른 앱을 다수 실행하는 경우가 많다. 각 앱은 관리하는 팀이 다르며, 중요도도 다르다. 사용자 및 그룹에게 제공된 접근의 필요성을 자세히 조사하지 않으면, 악의적이거나 부주의한 사용자가 오케스트레이터에 의해 관리되는 다른 컨테이너의 운영에 영향을 주거나 침해할 수 있다.

3.3.2. 인가되지 않은 접근

많은 오케스트레이터가 자체적인 인증 디렉터리 서비스를 가지고 있다. 하지만, 조직에서 이미 사용하고 있는 일반적인 디렉터리 서비스와 분리되어 있을 수 있다. 이는 오케스트레이터의 계정 관리를 취약하게 하거나, 오케스트레이터에 소유자가 없는 계정(orphaned account)이 방치될 수 있다. 이러한 시스템은 비교적 덜 엄격하게 관리되기 때문이다. 소유자가 없는 계정의 다수가 오케스트레이터에서 많은 권한을 가지고 있다. 따라서, 이러한 계정이 침해되면 전체 시스템이 침해될 수 있다.

컨테이너는 일반적으로 오케스트레이터가 관리하고, 특정 호스트에 종속되지 않는 별도의 데이터 스토리지를 사용한다. 컨테이너는 클러스터에 속한 모든 노드에서 실행될 수 있기 때문에, 컨테이너 내부의 앱이 요구하는 데이터는 앱이 실행되는 호스트에 관계없이 컨테이너에서 사용할 수 있어야 한다. 동시에 많은 조직에서는 인가되지 않은 접근을 방지하기 위해 반드시 암호화해야 하는 데이터를 관리하고 있다.

3.3.3. 컨테이너 間 네트워크 트래픽 분리 미흡

대부분의 컨테이너 환경에서 개별 컨테이너 사이의 트래픽은 가상 오버레이 네트워크 위에서 라우팅된다. 일반적으로 이 오버레이 네트워크는 오케스트레이터가 관리하는데, 오버레이 네트워크에서는 기존 네트워크 보안/관리 도구를 사용할 수 없다. 예를 들어, 웹 서버 컨테이너에서 다른 호스트의 데이터베이스 컨테이너로 보내는 데이터베이스 쿼리를 기존 네트워크 보안/관리 도구로 보면, 두 호스트 사이에 전달되는 암호화된 패킷만 볼 수 있을 것이다. 즉, 호스트의 어떤 컨테이너가 어떤 트래픽을 송신하는지에 대한 가시성을 가질 수 없다. 암호화된 오버레이 네트워크는 운영 및 보안과 관련한 많은 이점을 제공한다. 하지만, 네트워크의 트래픽을 효과적으로 모니터링할 수 없기 때문에 보안 사각지대(blindness)가 될 수도 있다.

잠재적으로 더욱 심각한 것은 서로 다른 앱이 동일한 가상 네트워크를 공유할 때 발생하는 위험이다. 중요도가 다른 앱(공개 웹, 내부 재무 웹 등)이 동일한 가상 네트워크를 사용하면, 네트워크 공격에 의해 내부의 중요한 앱이 더 큰 위험에 노출될 수 있다. 예를 들어, 공개 웹 사이트가 침해되면, 공격자는 재무 앱을 공격하기 위해 공유된 네트워크를 사용할 수 있다.

3.3.4. 업무 중요도 혼합

오케스트레이터는 일반적으로 업무의 범위와 밀도를 높이는데 초점을 맞춘다. 이는 기본적으로 서로 다른 중요도의 업무를 동일한 호스트에 배치할 수 있음을 의미한다. 예를 들어, 오케스트레이터의 디폴트(default) 설정을 적용하면, 중요한 재무 데이터를 처리하는 호스트에 공개 웹 서버가 실행되는 컨테이너를 함께 배치할 수 있다. 단지, 컨테이너를 배치되는 시점에서 그 호스트가 가장 많은 가용 자원을 가지고 있었기 때문이다. 심각한 취약점이 있는 웹 서버에 중요한 재무 데이터를 처리하는 컨테이너를 배치하면, 침해의 위험은 상당히 높아진다.

3.3.5. 오케스트레이터 신뢰

컨테이너 환경 내에서 노드 간 신뢰를 유지하는 것은 특별한 주의가 필요하다. 오케스트레이터는 가장 기본이 되는 노드이다. 오케스트레이터가 취약하게 설정되면, 오케스트레이터와 다른 모든 컨테이너 컴포넌트를 위험하게 한다. 이러한 사례는 다음과 같다.

- 비인가된 호스트가 클러스터에 가입되고, 컨테이너를 실행한다.
- 클러스터의 한 호스트가 침해되는 것은 전체 클러스터가 침해되는 것을 의미한다. 예를 들어, 인증에 사용되는 비밀키-공개키 쌍(key pair)을 모든 노드에서 공유한다면, 한 호스트의 침해가 전체 클러스터의 침해로 귀결될 것이다.
- 오케스트레이터, DevOps 담당자, 관리자, 호스트 사이의 통신이 암호화되지 않거나, 인증을 수행하지 않는다.

3.4. 컨테이너에 대한 위험

3.4.1. 컨테이너 런타임 內 취약점

비교적 일반적인 상황은 아니지만, “컨테이너 이스케이프(container escape)” 취약점은 컨테이너 런타임에게 특히 위험하다. “컨테이너 이스케이프” 취약점은 다른 컨테이너의 리소스 및 호스트 OS의 리소스를 공격할 수 있는 취약점이다. 공격자는 컨테이너 런타임의 취약점을 공격하여 컨테이너 런타임을 침해/변조한 후 다른 컨테이너에 접근하거나 컨테이너-컨테이너 間 트래픽을 스니핑할 수 있다.

3.4.2. 제한되지 않은 네트워크 접근

대부분의 컨테이너 런타임에서 컨테이너는 기본적으로 네트워크를 통해 다른 컨테이너 및 호스트 OS에 접근할 수 있다. 컨테이너가 침해되어 악의적인 행위를 하는 경우, 다른 컨테이너나 호스트에 대한 네트워크 접근을 허용하는 것은 컨테이너 환경의 다른 리소스를 위험에 노출시킬 수 있다. 예를 들어, 침해된 컨테이너는 네트워크를 스캔하는데 사용될 수 있다. 이를 통해, 공격자는 추가적인 공격이 가능한 다른 취약점을 찾을 수 있다. 이 위험은 3.3.3절에서 언급한 "컨테이너 間 네트워크 트래픽 분리 미흡(가상 네트워크의 분리 미흡)"과 관계가 있다. 그러나, "가상 네트워크의 분리 미흡"은 앱의 "크로스토크(crosstalk, 누화)" 보다 컨테이너로부터의 아웃바운드 트래픽 흐름에 더 초점을 맞추고 있다.

컨테이너 환경에서 네트워크 접근 시도를 관리하는 것은 더 복잡하다. 컨테이너 間 연결에 가상화를 많이 사용하기 때문이다. 따라서, 컨테이너 사이의 트래픽은 네트워크에서 단순히 캡슐화된 패킷으로 드러나며, 최종 출발지/목적지 또는 페이로드가 직접적으로 보이지 않는다. 컨테이너를 인식하지 못하는 도구 및 운영 프로세스는 컨테이너 사이의 트래픽을 검사하거나 위협 여부를 판단할 수 없다.

3.4.3. 안전하지 않은 컨테이너 런타임 설정

관리자는 일반적으로 컨테이너 런타임의 여러 가지 옵션을 설정할 수 있다. 옵션을 적절하게 설정하지 않으면 시스템의 보안이 상대적으로 낮아질 수 있다. 예를 들어, 리눅스 컨테이너 호스트에서 시스템 호출은 기본적으로 제한되며, 컨테이너를 안전하게 운용하기 위해 필요한 것만 허용된다. 허용된 시스템 호출이 많아지면, 침해된 컨테이너에 의해 다른 컨테이너 및 호스트 OS가 위험에 노출될 가능성이 높아진다. 유사하게, 컨테이너가 특권 모드(priviledged mode)로 실행되면, 호스트의 모든 디바이스에 접근할 수 있다. 즉, 컨테이너가 호스트 OS의 한 부분처럼 동작하며, 호스트 OS에서 실행되고 있는 다른 모든 컨테이너에 영향을 줄 수 있다.

안전하지 않은 컨테이너 런타임 설정의 다른 사례는 호스트의 중요한 디렉터리를 컨테이너에서 마운트할 수 있도록 허용하는 것이다. 컨테이너에 의해 호스트 OS의 파일 시스템이 변경되는 경우는 매우 적어야 하며, 컨테이너에 의해 호스트 OS의 기본적인 기능을 제어하는 경로(/boot, /etc, C:\Windows)가 변경되는 경우는 거의 없어야 한다. 침해된 컨테이너가 이러한 경로를 변경할 수 있으면, 권한 상승을 통해 호스트에서 실행되는 다른 컨테이너뿐만 아니라 호스트 자체를 공격할 수 있다.

3.4.4. 앱 취약점

조직이 이 문서에서 권장하는 예방 조치를 취하고 있는 경우에도 컨테이너에서 실행 중인 앱의 결함으로 컨테이너가 침해될 가능성은 여전히 남아있다. 이러한 가능성은 컨테이너 자체의 문제가 아니라, 일반적인 소프트웨어의 결함이 컨테이너 환경에서 발생한 것이다. 예를 들어, 컨테이너에서 실행되는 웹 앱은 XXS(cross-site script) 취약점에 취약할 수 있고, 데이터베이스 컨테이너는 SQL(structured query language) 인젝션의 대상이 될 수 있다. 컨테이너가 침해되면, 컨테이너는 다양한 방법으로 오용(중요한 정보에 비인가된 접근, 다른 컨테이너 또는 호스트 OS에 대한 공격 등)될 수 있다.

3.4.5. 로그(rogue) 컨테이너

로그 컨테이너는 계획되지 않거나 공인되지 않은 컨테이너이다. 특히, 로그 컨테이너는 개발 환경에서 자주 발생할 수 있다. 앱 개발자는 코드를 테스트하기 위한 수단으로 컨테이너를 만든다. 이러한 컨테이너에 대해 취약점 점검과 적절한 설정을 엄격히 하지 않으면, 컨테이너는 침해될 가능성이 높다. 따라서, 로그 컨테이너는 조직을 또 다른 위험에 빠뜨린다. 특히, 개발팀 및 보안 관리자가 로그 컨테이너를 인지하지 못한 채, 계속 유지되면 더 위험하다.

3.5. 호스트 OS에 대한 위험

3.5.1. 넓은 공격 표면(attack surface)

모든 호스트 OS는 공격 표면을 갖는다. 공격 표면이란, 공격자가 취약점에 접근하고 공격할 수 있는 방법을 모두 모은 것이다. 예를 들어, 네트워크를 통해 접근할 수 있는 모든 서비스는 공격자에 의해 침해될 가능성이 있다. 따라서, 이러한 서비스는 모두 공격 표면에 추가될 것이다. 공격 표면이 넓을수록 공격자가 취약점을 찾고 접근할 수 있는 가능성이 높아지며, 호스트 OS와 호스트 OS에서 실행 중인 컨테이너가 침해될 수 있다.

3.5.2. 공유 커널

컨테이너 전용 OS는 범용 OS 보다 공격 표면이 훨씬 좁다. 예를 들어, 컨테이너 전용 OS에는 데이터베이스와 웹 서버 앱을 직접 실행할 수 있는 라이브러리와 패키지 매니저가 없다. 컨테이너는 리소스를 소프트웨어 수준에서 강력하게 분리한다. 그러나, 컨테이너 전용 OS라도 커널은 공유되기 때문에 하이퍼바이저보다 객체(컨테이너 및 호스트 OS) 사이의 공격 표면이 넓다. 즉, 컨테이너 런타임에 의한 분리의 수준은 하이퍼바이저만큼 높지 않다.

3.5.3. 호스트 OS 컴포넌트 취약점

컨테이너 전용 OS를 비롯한 모든 호스트 OS는 기본적인 시스템 컴포넌트를 제공한다. 예를 들어, 원격 인증에 사용하는 암호화 라이브러리와 일반적인 프로세스 호출/관리에 사용하는 커널 프리미티브(kernel primitives) 등이다. 다른 모든 소프트웨어와 마찬가지로, 이러한 컴포넌트에도 취약점이 있을 수 있다. 또한, 이러한 취약점은 컨테이너 아키텍처의 하위에 위치하기 때문에 호스트에서 실행되는 모든 컨테이너와 앱에 영향을 줄 수 있다.

3.5.4. 부적절한 사용자 접근 권한

컨테이너 전용 OS는 대화형 사용자 로그온(interactive user logon)을 지양하기 때문에, 일반적으로 다중 사용자(multiuser)에 대한 지원에 최적화되어 있지 않다. 만약 사용자가 오케스트레이터를 통하지 않고 컨테이너를 관리하기 위해 호스트에 직접 로그온한다면, 조직은 위험에 노출된다. 이는 시스템 및 시스템에서 실행되는 모든 컨테이너의 많은 부분을 변경할 수 있고, 특정 컨테이너만 관리해야 하는 사용자가 많은 컨테이너에 영향을 줄 수 있기 때문이다.

3.5.5. 호스트 OS 파일 시스템 변조

컨테이너의 설정이 안전하지 않으면, 호스트의 볼륨이 파일 변조의 위험에 노출된다. 예를 들어, 컨테이너가 호스트 OS의 중요한 디렉터리를 마운트할 수 있으면, 그 디렉터리의 파일을 변경할 수 있다. 이러한 변경은 호스트 및 호스트에서 실행되는 다른 모든 컨테이너의 안정성과 보안에 영향을 줄 수 있다.

4. 중요 위험에 대한 보호 대책

이 장에서는 3장에서 식별한 중요 위험에 대한 보호 대책을 권고한다.

4.1. 이미지에 대한 보호 대책

4.1.1. 이미지 취약점

컨테이너 전용 취약점 관리 도구/프로세스가 필요하다. 기존 취약점 관리 도구는 호스트의 내구성, 앱 업데이트 메커니즘, 앱 업데이트 빈도에 대해 가정하는 부분이 많다. 그러나, 이러한 가정은 기본적으로 컨테이너 모델에 적합하지 않다. 기존 취약점 관리 도구는 컨테이너 내부의 취약점을 발견할 수 없으며, 잘못된 안전 의식(false sense of safety)을 초래한다.

조직에서는 파이프라인 기반 배치 방법, 컨테이너/이미지의 불변성을 설계에 반영한 도구를 사용해야 한다. 이러한 도구는 더욱 실용적이고 신뢰할 수 있는 결과를 제공한다. 효과적인 도구/프로세스가 가져야 할 중요 요소는 다음과 같다.

1. 이미지의 모든 생명 주기(배치 프로세스 시작-레지스트리-런타임)와 통합되어야 한다.
2. 이미지의 모든 계층(이미지의 기본 계층, 애플리케이션 프레임워크, 커스텀 소프트웨어)에서 취약점에 대한 가시성을 가져야 한다. 조직 전반의 가시성은 통합되어야 하며, 조직의 비즈니스 프로세스에 적합한 그리고 유연한 레포팅/모니터링 화면을 제공해야 한다.
3. 조직은 배치 프로세스의 각 단계마다 품질 게이트(guality gate)를 만들어야 한다. 품질 게이트는 조직의 취약점/설정 정책에 부합하는 이미지만 계속 진행될 수 있게 한다. 예를 들어, 조직은 설정한 임계값을 초과하는 CVSS(Common Vulnerability Scoring System)[18] 등급이 부여된 취약점이 존재하는 이미지가 배치 프로세스에서 계속 진행되는 것을 방지하는 규칙을 설정할 수 있다.

4.1.2. 이미지 설정 결함

조직은 안전한 설정과 관련한 모범 사례(best practice)가 적용된 컴플라이언스를 확인하고 시행하는 도구/프로세스를 채택해야 한다. 예를 들어, 이미지는 일반 사용자 권한으로 실행되도록 설정해야 한다. 이러한 도구/프로세스에는 다음과 같은 기능을 가져야 한다.

1. 이미지 설정(벤더의 권고사항, 서드 파티의 모범 사례 등) 확인
2. 조직의 수준에서 취약점/위험을 식별하기 위해 이미지의 컴플라이언스 상태를 지속적으로 업데이트하고 통합하는 레포팅 및 모니터링
3. 컴플라이언스를 준수하지 않는 이미지의 실행을 선택적으로 차단함으로써 컴플라이언스 요구사항을 시행
4. 신뢰할 수 있는 베이스 레이어만 사용, 주기적인 베이스 레이어 업데이트, 공격 표면을 축소하기 위해 베이스 레이어를 최소화한 기술(알파인 리눅스, 윈도우 나노 서버 등)을 선택

이미지 설정 결함과 관련한 마지막 권고사항은 호스트에 리모트 쉘을 제공할 수 있는 원격 관리 도구(SSH 등)를 컨테이너 내에서 사용할 수 있도록 설정하지 않아야 한다는 것이다. 컨테이너 사용에 따른 이점을 최대한 활용하기 위해서는 컨테이너를 변경되지 않는 방법으로 운영해야 한다. 원격 관리 도구를 통해 컨테이너에 원격으로 접근할 수 있으면, 컨테이너가 변하지 않아야 한다는 원칙을 위반할 수 있다. 또한, 컨테이너를 네트워크 기반 공격의 위험에 노출시킨다. 컨테이너에 대한 원격 관리는 컨테이너 런타임 API(오케스트레이터를 통해 접근 가능)를 통하거나, 컨테이너가 실행 중인 호스트에서 대한 리모트 쉘 세션을 생성하는 방법으로만 수행되어야 한다.

4.1.3. 악성코드 포함

조직은 이미지에 포함된 악성코드를 탐지하기 위해 모든 이미지를 지속적으로 모니터링해야 한다. 모니터링 프로세스는 실제 공격 기반의 악성코드 패턴(malware signature)과 행위 분석(behavioral detection heuristics)을 사용해야 한다.

4.1.4. 평문으로 저장된 기밀(secret)

기밀은 이미지 외부에 저장해야 하며, 실행 중에 동적으로 제공되어야 한다. 대부분의 오케스트레이터(도커 스웜, 쿠버네티스 등)는 기밀 관리 기능을 기본적으로 제공한다. 오케스트레이터는 안전한 기밀 스토리지 및 JIT(Just in time) 인젝션 기능을 제공한다. 또한, 오케스트레이터는 기밀 관리와 배치 프로세스의 통합을 용이하게 한다. 예를 들어, 조직은 오케스트레이터를 사용하여 데이터베이스 연결 문자열을 웹 애플리케이션 컨테이너에 제공한다. 오케스트레이터는 해당 웹 애플리케이션만 이 기밀(데이터베이스 연결 문자열)에 접근할 수 있게 하고, 디스크에 저장되지 않게 하며, 웹 앱이 배치될 때마다 기밀이 웹 앱에 제공되도록 한다.
또한, 조직은 기존 전사 기밀 관리 시스템(非 컨테이너 환경에서 기밀을 저장하기 위해 사용)과 컨테이너 배치 프로세스의 통합을 계획할 수 있다. 이러한 기밀 관리 시스템은 일반적으로 컨테이너가 배치될 때 기밀을 안전하게 검색하기 위한 API 제공한다. 따라서, 이미지 내부에 비밀을 저장할 필요가 없다.

선택된 도구에 관계없이, 조직은 기밀이 필요한 컨테이너[1]에만 기밀을 제공해야 하며, 저장/전송 중에는 항상 검증된 암호화 모듈 및 안전한 암호화 알고리즘[2]을 사용하여 암호화되도록 해야 한다.

4.1.5. 신뢰할 수 없는 이미지 사용

조직은 신뢰할 수 있는 이미지와 레지스트리를 관리해야 하며, 신뢰할 수 있는 이미지만 실행할 수 있도록 해야 한다. 이를 통해 신뢰할 수 없거나 악의적인 컴포넌트가 배치되는 위험을 완화할 수 있다.

이러한 위험을 완화하기 위해, 조직은 다음 사항을 포함하는 다중 레이어 접근법(multilayered approach)을 채택해야 한다.

- 해당 환경의 이미지/레지스트리에 대한 신뢰를 중앙에서 정확하게 통제할 수 있어야 한다.
- 각 이미지는 안전한 암호 알고리즘[3]으로 개별 서명되어야 한다.
- 모든 호스트는 승인된 이미지만 실행할 수 있도록 해야 한다.
- 이미지의 서명을 검증하여, 이미지의 출처 및 변조 여부를 확인한 후 이미지를 실행해야 한다.
- 이미지 저장소를 지속적으로 모니터링하고 유지보수하여, 취약점 및 설정 관련 요구사항이 변경되면 이미지가 업데이트되도록 해야 한다.

4.2. 레지스트리에 대한 보호 대책

4.2.1. 레지스트리에 대한 안전하지 않은 연결

조직은 개발 도구, 오케스트레이터, 컨테이너 런타임이 암호화된 채널을 통해 레지스트리에 연결되도록 설정해야 한다. 구체적인 절차는 도구마다 차이가 있다. 하지만, 중요한 목적은 신뢰할 수 있는 엔드포인트만 레지스트리와 데이터를 주고 받고, 전송 중에 데이터가 암호화되게 하는 것이다.

---

1 기밀이 필요한 컨테이너는 관리자가 사전에 정의한 설정을 기반으로 선정되어야 한다.

2 원문에서는 FIPS(Federal Information Processing Standard) 140에서 승인된 암호화 알고리즘으로 명시되어 있다.

3 원문에서는 NIST에서 검증한 알고리즘(NIST-validated implementation)으로 표현하고 있다.

4.2.2. 레지스트리의 노후된 이미지

노후된 이미지 사용에 따른 위험은 두 가지 중요한 방법을 통해 완화할 수 있다. 첫째, 레지스트리에서 더 이상 사용해서는 안되는 취약한 이미지를 정리하는 것이다. 이 프로세스는 시간 기반 트리거(time trigger) 및 이미지 라벨을 기반으로 자동화할 수 있다. 둘째, 이미지에 접근할 때 사용할 이미지의 각 버전을 명시한 변경되지 않는 이름을 사용하는 것이다. 예를 들어, my-app이라는 이미지를 사용하도록 배치 작업을 설정하기보다, 이미지의 특정 버전(예 : my-app:2.3 또는 my-app:2.4)을 배치하도록 설정해야 한다. 이는 양호한 것으로 알려진 이미지가 배치되도록 한다.

다른 방법은 이미지에 “latest” 태그를 사용하고, 배치 자동화에서 “latest” 태그를 참조하는 것이다. 그러나, “latest” 태그는 단지 이미지에 붙이는 라벨이며, 이미지가 최신이라는 것을 보장하지 않는다. 따라서, “latest” 태그를 지나게 신뢰하지 않도록 유의해야 한다. 조직은 “latest” 태그 또는 “최신”을 의미하는 별도의 이름을 사용하는지에 관계없이, 자동화에서 이러한 태그/이름이 붙은 이미지가 실제로 최신 버전을 가리키게 하는 프로세스를 시행하여야 한다.

4.2.3. 불충분한 인증 및 인가

특허가 있는 이미지 또는 중요한 이미지가 저장된 레지스트리에 대한 모든 접근은 인증을 받아야 한다. 레지스트리에 기록하기 위한 모든 접근은 인증을 받아서, 신뢰할 수 있는 인원만 레지스트리에 이미지를 추가할 수 있도록 해야 한다. 예를 들어, 개발자는 모든 저장소가 아닌 자신이 담당하는 저장소에만 이미지를 저장할 수 있도록 해야 한다.

조직은 기존 계정(조직의 자체 디렉터리 서비스, 클라우드 디렉터리 서비스 등)과 통합을 검토해야 한다. 기존 계정에는 이미 보안 통제가 마련되어 있고, 이를 활용할 수 있기 때문이다. 레지스트리에 기록하기 위한 모든 접근을 검사해야 하며, 이와 유사하게 중요한 이미지를 읽기 위한 모든 접근도 기록해야 한다.

또한, 레지스트리는 상황에 따라 인가를 차등적으로 통제할 수 있게 해준다. 예를 들어, 담당 직원이 서명하고, 취약점 점검과 컴플라이언스 평가를 통과한 이미지만 레지스트리에 저장할 수 있도록 지속적 통합 프로세스를 구성할 수 있다. 조직은 취약점 점검 및 컴플라이언스 평가와 같은 자동화된 점검을 조직의 지속적 통합 프로세스에 통합하여, 취약하거나 잘못 설정된 이미지가 다음 단계로 진행되거나 배치되는 것을 예방해야 한다.

4.3. 오케스트레이터에 대한 보호 대책

4.3.1. 제한되지 않은 관리자 접근

오케스트레이터의 통제 범위는 특히 넓다. 따라서, 오케스트레이터는 최소 권한 접근 모델을 사용하여, 특정 호스트/컨테이너/이미지에서 특정 작업을 수행할 수 있는 권한만 사용자에게 부여해야 한다. 예를 들어, 테스터에게는 테스트에 사용할 이미지와 이미지를 실행하기 위한 호스트에만 접근 권한이 부여되어야 하며, 테스터가 생성한 컨테이너만 조작할 수 있어야 한다. 테스터는 운영 환경에서 사용 중인 컨테이너에 접근할 수 없어야 한다.

4.3.2. 인가되지 않은 접근

전체 클러스터를 관리하는 계정에 대한 접근을 엄격하게 통제해야 한다. 이러한 계정은 클러스터 환경의 모든 리소스에 영향을 미칠 수 있기 때문이다. 조직에서는 비밀번호만 사용하는 대신 다중 인증과 같은 강력한 인증 방법을 사용해야 한다.

조직은 기존의 디렉터리 시스템에 대해 SSO(single sign-on)를 구현해야 한다. SSO는 오케스트레이터 인증을 단순하게 하고, 사용자가 강력한 인증 크리덴셜을 쉽게 사용할 수 있게 해준다. 또한, 접근에 대한 감사를 통합함으로써 비정상적인 접근을 더 효과적으로 탐지할 수 있게 해준다.

저장 데이터를 암호화하는 일반적인 방법은 호스트의 기능을 사용하는 것이다. 하지만, 이러한 방법은 컨테이너와 호환되지 않을 수 있다. 따라서, 조직은 컨테이너에서 사용할 수 있는 데이터 암호화 도구를 사용해야 한다. 이러한 도구는 컨테이너가 실행되는 노드와 관계없이 컨테이너에서 데이터에 적절하게 접근할 수 있게 해준다. 암호화 도구는 NIST SP 800-111에서 정의한 암호화 방법을 사용하여, 인가되지 않은 접근과 변조를 방어해야 한다.

4.3.3. 컨테이너 間 네트워크 트래픽 분리 미흡

중요도에 따라 네트워크 트래픽을 각각의 가상 네트워크로 분리하도록 오케스트레이터를 설정해야 한다. 앱 별로 세분화하는 것도 가능하다. 하지만, 중요도에 따라 네트워크를 정의하는 것만으로도 충분히 복잡(a manageable degree of complexity)하며, 위험을 충분히 완화할 수 있다. 예를 들어, 다수의 공개 앱은 하나의 가상 네트워크를 공유할 수 있다. 내부 앱은 다른 가상 네트워크를 사용할 수 있다. 공개 앱과 내부 앱 사이의 통신은 잘 정의된 소수의 인터페이스를 통해 이루어져야 한다.

4.3.4. 업무 중요도 혼합

중요도에 따라 특정 호스트를 그룹으로 분리하여 배치하도록 오케스트레이터를 설정해야 한다. 이를 구현하기 위한 세부적인 방법은 사용 중인 오케스트레이터에 따라 다르다. 하지만, 일반적인 모델은 중요도가 높은 업무가 중요도가 낮은 업무를 실행하는 호스트에 같이 배치되지 않도록 정책을 정의하는 것이다. 이는 오케스트레이터에서 호스트 피닝(host-pinning)을 사용하거나, 중요도별로 클러스터를 분리 또는 개별 관리함으로써 가능하다.

대부분의 컨테이너 런타임 환경은 호스트 및 다른 컨테이너로부터 컨테이너를 효과적으로 분리한다. 하지만, 다른 중요도의 앱을 같은 호스트 OS에서 함께 실행하는 것은 지양(unnecessary risk)해야 한다. 컨테이너를 목적/중요도/위협에 따라 구분하는 것은 추가적인 심층 방어(defense in depth)를 제공한다. 애플리케이션 계층화(tiering) 및 네트워크/호스트 분리(segmentation)와 같은 개념은 앱 배치를 계획하는 단계부터 고려해야 한다. 예를 들어, 재무 데이터베이스 컨테이너와 공개 블로그 컨테이너를 동일한 호스트에서 실행하고 있다고 가정하자. 컨테이너 런타임은 일반적으로 이러한 환경을 효과적으로 분리할 것이다. 하지만, DevOps팀은 담당하는 앱을 안전하게 운영하고, 불필요한 위험을 제거해야 하는 책임이 있다. 블로그 앱이 공격자에 의해 침해된 경우, 두 가지 앱이 동일한 호스트에서 실행되고 있다면 재무 데이터베이스을 보호할 수 있는 방어 계층은 훨씬 적을 것이다.

따라서, 모범 사례는 상대적인 중요도에 따라 컨테이너를 그룹화하고, 특정 호스트에서는 동일한 중요도의 컨테이너만 실행되도록 하는 것이다. 이렇게 분리하기 위해 물리적 서버 여러 대를 사용할 수 있다. 하지만, 최신 하이퍼바이저를 사용하여 분리하여도 이러한 위험을 완화하기에 충분히 강력하다. 앞의 예에서, 조직의 컨테이너는 두 가지 중요도로 구분할 수 있다. 한 그룹에는 재무 앱과 데이터베이스가 포함된다. 다른 그룹에는 웹 앱과 블로그가 포함된다. 그렇다면, 조직은 두 가지 VM 풀을 가질 것이다. 각 VM 풀은 한 가지 중요도의 컨테이너를 관리한다. 예를 들어, “재무 VM”이라고 불리는 호스트는 재무 데이터베이스와 세금 신고 소프트웨어가 실행되는 컨테이너를 관리할 것이다. 반면, “웹 VM”이라고 불리는 호스트는 블로그와 공개 웹 사이트를 관리할 것이다.

이러한 방법으로 컨테이너를 분리하면, 공격자가 특정 그룹에 침투한 후 다른 그룹으로 침투하는 것이 매우 어려워진다. 공격자가 한 서버에 침투한 후 사전 조사(reconnaissance)를 수행하거나, 유사한 중요도의 다른 컨테이너를 공격하는 것은 제한될 것이다. 그리고, 공격자가 침투한 서버를 넘어 추가적으로 접근할 수 없을 것이다. 또한, 모든 잔존 데이터[1]는 데이터 보안 영역을 벗어날 수 없다. 앞의 예에서 이 방법으로 컨테이너를 분리하면, 컨테이너가 종료된 후 캐시되었던 모든 재무 데이터는 낮은 중요도의 앱을 실행하는 호스트에서 절대 사용할 수 없다.

[1] 캐시(cache) 또는 임시 파일이 마운트된 로컬 볼륨 등

수백 대의 호스트와 수천 대의 컨테이너가 있는 대규모 환경에서는 이러한 분리를 자동화해야만 실제 운영할 수 있다. 다행히 일반적인 오케스트레이터는 애플리케이션을 그룹화할 수 있는 몇 가지 기능을 가지고 있다. 그리고, 컨테이너 보안 도구는 컨테이너 이름/라벨과 같은 속성을 사용하여 보안 정책을 적용할 수 있다. 또한, 이러한 대규모 환경에서 이러한 분리는 단순히 호스트를 분리하는 것을 넘어, 추가적인 심층 방어 계층으로도 활용할 수 있다. 예를 들어, 조직은 호스팅 존(hosting zone)을 분리하거나 네트워크를 분리하여, 하이퍼바이저 내부의 컨테이너를 분리할 수 있고, 특정 중요도를 갖는 앱의 트래픽을 다른 중요도를 갖는 앱과 분리되도록 네트워크 트래픽을 분리할 수도 있다.

4.3.5. 오케스트레이터 신뢰

오케스트레이터가 실행하는 모든 앱을 위해 안전한 환경을 생성할 수 있도록 오케스트레이터를 설정해야 한다. 오케스트레이터는 노드를 클러스터에 안전하게 포함시켜야 하고, 전체 생명 주기(lifecycle)에서 고유한 ID(persistent identity)를 가져야 한다. 또한, 노드 및 노드의 연결 상태에 대한 정확한 목록을 제공할 수 있어야 한다. 조직은 개별 노드가 침해되더라도 클러스터 전체의 보안이 침해되지 않도록 오케스트레이터를 명확하게 설계해야 한다. 침해된 노드는 전체 클러스터의 운영에 지장을 주지 않으면서 클러스터에서 분리하고 제거할 수 있어야만 한다. 마지막으로 오케스트레이터를 선택할 때, 클러스터 멤버 間 상호 인증 및 종단 間 암호화를 제공하는지 확인해야 한다. 컨테이너의 이식성(portability) 때문에 조직이 직접 통제하지 않는 네트워크에 컨테이너가 배치되는 경우가 많이 있다. 따라서, 기본 설정을 가장 안전한 설정으로 하는 것(secure-by-default)이 특히 중요하다.

4.4. 컨테이너에 대한 보호 대책

4.4.1. 런타임 소프트웨어의 취약점

컨테이너 런타임에 취약점이 있는지 주의깊게 모니터링해야 하고, 문제가 탐지되면 빠르게 해결해야 한다. 취약한 컨테이너 런타임은 자신이 지원하는 모든 컨테이너 및 호스트를 잠재적으로 중대한 위험에 빠뜨린다. 조직은 컨테이너 런타임에 대한 CVE(Common Vulnerabilities and Exposures) 취약점을 검색하는 도구를 사용해야 하고, 위험이 있는 모든 인스턴스를 업그레이드해야 한다. 또한, 오케스트레이터는 적절하게 관리되는 컨테이너 런타임에만 배치해야 한다.

4.4.2. 제한되지 않은 네트워크 접근

조직은 컨테이너가 아웃바운드로 전송하는 네트워크 트래픽을 통제해야 한다. 컨테이너가 다른 중요도의 네트워크로 트래픽을 보낼 수 없도록 최소한 네트워크의 경계에서는 통제를 수행해야 한다. 이는 중요한 데이터를 호스팅하는 환경에서 인터넷으로 전송하는 등의 전통적인 아키텍처에서 사용하는 패턴과 유사하다. 그러나, 컨테이너 間 트래픽이 가상화되는 네트워크 모델은 새로운 문제가 발생한다.

다수 호스트에 배치된 컨테이너 사이의 통신은 가상의 암호화된 네트워크를 사용하는 것이 일반적이기 때문에 기존의 네트워크 장비로는 이 트래픽을 볼 수 없다. 또한, 오케스트레이터가 컨테이너를 배치할 때 컨테이너에 IP 주소를 동적으로 자동 할당하는 것이 일반적이다. IP 주소는 앱을 확장하거나 부하를 분산하면서 지속적으로 변경된다. 따라서, 조직에서는 기존의 네트워크 장비와 앱 인식 네트워크 도구를 조합하여 사용해야 한다. 앱 인식 도구는 컨테이너 사이의 트래픽을 볼 수 있어야 한다. 또한, 컨테이너에서 실행되는 앱의 특성에 기반하여 트래픽을 필터링하는데 사용되는 규칙을 동적으로 생성할 수 있어야 한다. 이러한 동적 규칙 관리는 중요하다. 컨테이너 앱은 변화의 규모가 크고, 변화의 속도가 빠르며, 네트워크 토폴로지도 자주 변하기 때문이다.

특히, 앱 인식 도구는 다음과 같은 기능을 보유해야 한다.

- 적절한 컨테이너 네트워크 표면[1]을 자동으로 결정해야 한다.
- 컨테이너와 다른 네트워크 엔티티 사이에서 전송 중(on the wire)이거나 암호화된 트래픽 흐름(traffic flow)을 탐지해야 한다.
- 네트워크 이상 징후[2]를 탐지해야 한다.

4.4.3. 안전하지 않은 컨테이너 런타임 설정

조직은 컨테이너 런타임 설정 표준에 대한 컴플라이언스를 자동화해야 한다. CIS(Center for Internet Security) 도커 벤치마크[20]와 같은 기술 구현 가이드에는 옵션과 권장 설정에 대한 세부사항이 수록되어 있다. 하지만, 이 가이드를 운용하는 것은 자동화에 달려 있다. 조직에서는 특정 시점의 컴플라이언스를 검사하고 평가하기 위해 다양한 도구를 사용할 수 있다. 하지만, 이러한 방식은 확장성이 떨어진다. 조직에서는 전체 환경의 설정을 지속적으로 평가하고, 적극적으로 시행하는 도구 또는 프로세스를 사용해야 한다.

1 인바운드 포트 및 프로세스 포트 바인딩 등

2 예상 밖의 트래픽 흐름, 포트 스캐닝, 잠재적 위험이 있는 목적지에 접근 등

또한, SeLinux[21] 및 AppArmor[22]와 같은 MAC(Mandatory Access Control, 강제적 접근 제어) 기술은 리눅스 OS가 실행되는 컨테이너를 더욱 강하게 통제하고 분리한다. 예를 들어, MAC 기술을 사용하면, 컨테이너가 특정 파일/프로세스/네트워크 소켓에만 접근하게 할 수 있다. 또한, 침해된 컨테이너가 호스트 또는 다른 컨테이너에 영향을 줄 수 없게 한다. MAC 기술은 호스트 OS 레이어의 특정 파일/경로/프로세스만 컨테이너 앱에 접근할 수 있게 해준다. 조직에서 컨테이너를 배치할 때, 호스트 OS의 MAC 기술을 사용할 것을 권장한다.

보안 컴퓨팅(seccomp)[1] 프로필은 컨테이너에 할당되는 시스템 레벨의 기능을 제한하는데 사용할 수 있는 메커니즘이다. 도커와 같은 일반적인 컨테이너 런타임의 디폴트 보안 컴퓨팅 프로필은 일반적으로 불안전/불필요한 시스템 호출을 중단하는 설정이 포함된다. 또한, 사용자 정의 프로필을 생성하고 컨테이너 런타임에 전달하여 기능을 더욱 제한할 수 있다. 최소한, 조직은 컨테이너를 컨테이너 런타임의 디폴트 프로필로 실행해야 한다. 그리고, 고위험 앱에 대해서는 추가적인 프로필을 사용하는 것을 검토해야 한다.

4.4.4. 앱 취약점

기존 호스트 기반의 침입 탐지 프로세스/도구는 컨테이너 내부의 공격을 탐지하고 예방할 수 없다. 앞에서 언급한 것처럼 기술 아키텍처와 운영 실무가 다르기 때문이다. 조직에서는 컨테이너를 인식할 수 있고, 일반적인 컨테이너 환경의 규모/변화를 고려하여 설계한 추가적인 도구를 구현해야 한다. 이러한 도구는 컨테이너 앱의 행위를 학습하여 분석하고, 컨테이너 앱에 대한 보안 프로필을 자동으로 작성함으로써 사람의 개입을 최소화해야 한다. 이러한 보안 프로필은 다음과 같은 이상 행위를 탐지하고 예방할 수 있어야 한다.

- 불명확하거나 예상 밖의 프로세스 실행
- 불명확하거나 예상 밖의 시스템 호출
- 보호된 설정 파일 및 바이너리의 변경
- 예상 밖의 경로 및 파일 형태로 저장
- 예상 밖의 네트워크 리스너(listener)의 생성
- 예상 밖의 목적지로 전송되는 네트워크 트래픽
- 악성코드 저장 또는 실행

또한, 컨테이너의 루트 파일 시스템을 읽기 전용으로 실행되어야 한다. 이 방법은 구체적으로 정의된 디렉터리에만 기록할 수 있게 한다. 따라서, 앞서 언급한 도구로 더 쉽게 모니터링할 수 있다. 또한, 읽기 전용 파일 시스템을 사용하면, 컨테이너가 침해되더라도 더 쉽게 복구할 수 있다. 특정 경로만 변조할 수 있으며, 앱의 다른 부분과 쉽게 분리할 수 있기 때문이다.

[1] https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt.

4.4.5. 로그(rogue) 컨테이너

조직에서는 개발-테스트-운영 등의 환경을 분리하여 구축해야 한다. 각 환경은 컨테이너를 배치 및 관리할 때, 역할을 기반으로 접근 통제(Role-Based Access Control, RBAC)하기 위해 특정한 통제를 갖추어야 한다. 컨테이너의 생성은 개별 사용자 ID와 빠짐없이 연결되고 기록되어야 한다. 이는 컨테이너 생성에 대한 명확한 감사 로그를 남기기 위해서이다. 또한, 조직에서 이미지의 실행을 허용하기에 앞서, 취약점을 관리하고 컴플라이언스를 준수하기 위한 기본적인 요구사항을 시행할 수 있는 보안 도구를 사용할 것을 권장한다.

4.5. 호스트 OS에 대한 보호 대책

4.5.1. 넓은 공격 표면

컨테이너 전용 OS를 사용하면, 위협이 더 적어지는 것이 일반적이다. 컨테이너 전용 OS는 컨테이너를 호스팅하고, 다른 서비스 및 기능이 비활성화되도록 특별하게 설계되었기 때문이다. 또한, 이러한 최적화된 OS는 일반적으로 읽기 전용 파일 시스템을 특징으로 하고, 다른 하드닝(hardening) 작업을 기본으로 한다. 조직에서는 가능하다면 이러한 최소화된 OS를 사용해야 한다. 이는 공격 표면을 최소화하고, 일반적인 위험을 완화하며, 범용 OS을 사용했을 때 수행해야 하는 하드닝 작업을 줄일 수 있기 때문이다.

컨테이너 전용 OS를 사용할 수 없다면, 호스트의 공격 표면을 가능한 한 줄이기 위해 범용 서버 보안 가이드[1]를 준수해야 한다. 예를 들어, 컨테이너를 실행하는 호스트는 컨테이너만 실행해야 하며, 컨테이너 외에 다른 앱(웹 서버 또는 데이터베이스 등)을 실행해서는 안된다. 호스트 OS는 공격 표면을 증가시키는 불필요한 시스템 서비스(프린트 스풀러 등)를 실행해서는 안된다. 마지막으로 컨테이너 런타임 뿐만 아니라 호스트의 로우 레벨(lower-level) 컴포넌트[2]에 대해서도 취약점 및 중요 업데이트를 지속적으로 검사해야 한다.

4.5.2. 공유 커널

조직에서는 중요도에 따라 컨테이너 작업을 호스트로 그룹화해야 한다. 또한, 컨테이너화된 작업과 非 컨테이너화된 작업을 동일한 호스트에서 혼합하여 실행해서는 안된다. 예를 들어, 호스트에서 웹 서버 컨테이너를 실행중이라면, 호스트 OS에 설치된 컴포넌트로 웹 서버 등의 앱을 실행해서는 안된다. 컨테이너화된 작업을 컨테이너 전용 호스트로 분리하면, 컨테이너 보호에 최적화된 보호 대책 및 방어를 더욱 간단하고 안전하게 적용할 수 있다.

---

1 NIST 800-123, Guide to General Server Security [23]

2 다수의 컨테이너를 안전하게 분리하기 위해 사용하는 커널 등

4.5.3. 호스트 OS 컴포넌트 취약점

조직에서는 베이스 OS에 포함된 관리/기능 컴포넌트의 버전을 확인하기 위한 관리 활동 및 도구를 구현해야 한다. 컨테이너 전용 OS는 범용 OS보다 훨씬 더 적은 컴포넌트를 가지고 있다. 하지만, 취약점은 여전히 존재하며, 업데이트도 필요하다. 조직에서는 OS 제조사 또는 다른 신뢰할 수 있는 기관에서 제공하는 도구를 사용하여, OS의 모든 소프트웨어 컴포넌트를 정기적으로 업데이트해야 한다. OS는 보안 업데이트 및 제조사에서 권고하는 최신 컴포넌트 업데이트를 통해 최신 상태로 유지해야 한다. 커널 및 컨테이너 런타임 컴포넌트에 대한 최신 업데이트는 특히 중요하다. 이러한 컴포넌트의 새로운 릴리즈는 단순히 취약점을 수정하는 것 이외에 보안과 관련된 추가적인 방어 및 기능을 추가하는 경우가 많기 때문이다. 일부 조직에서는 기존 시스템을 업데이트하지 않고, 필요한 업데이트를 적용한 새로운 OS 인스턴스를 재배포할 수 있다. 이러한 방법 또한 효과적이지만, 더 정교하게 운영해야 한다.

호스트 OS는 변경되지 않는 방법으로 운영해야 한다. 즉, 호스트에는 데이터 및 상태(state)가 단독/영구적으로 저장되지 않아야 한다. 또한, 호스트의 애플리케이션에 대한 의존성이 없어야 한다. 대신 모든 앱 컴포넌트와 의존성이 있는 애플리케이션을 컨테이너로 패키징하여 배치해야 한다. 이렇게 하면, 상태를 거의 유지하지 않는 방법(stateless)으로 호스트를 운영할 수 있다. 또한, 비정상 및 구성 편차(configuration drift)를 더욱 확실하게 식별할 수 있다.

4.5.4. 부적절한 사용자 접근 권한

오케스트레이터는 호스트에 작업을 분배하며, 오케스트레이터가 호스트에 컨테이너를 배치하는 경우가 대부분이다. 하지만, 조직에서는 호스트 OS에 대한 모든 인증을 감사하고, 비정상적인 로그인을 모니터링하며, 권한이 상승되는 모든 작업을 기록해야 한다. 이렇게 감사/모니터링/기록하면, 비정상적인 접근 패턴[1]을 식별할 수 있다.

4.5.5. 호스트 OS 파일 시스템 변조

컨테이너는 파일 시스템 권한을 최소한으로 실행해야 한다. 컨테이너가 호스트의 로컬 파일 시스템을 마운트(mount)하는 경우는 거의 없어야 한다. 컨테이너가 파일을 디스크에 보존해야 한다면, 이를 위해 특별히 할당된 스토리지 볼륨에서만 변경할 수 있어야 한다. 어떠한 경우에도 컨테이너는 호스트 파일 시스템의 중요한 디렉터리(특히, OS 설정 파일이 저장된 디렉터리)를 마운트할 수 없어야 한다.
조직에서는 컨테이너가 마운트한 디렉토리를 모니터링하고, 앞서 언급한 호스트 파일 시스템과 관련된 정책을 위반하는 컨테이너가 배치되는 것을 방지할 수 있는 도구를 사용해야 한다.

[1] 호스트에 대한 직접 로그인, 컨테이너를 조작할 수 있는 권한을 가진 명령의 실행 등

4.6. 하드웨어에 대한 보호 대책

NIST SP 800-164[24]에서 인정한 것처럼 소프트웨어 기반 보안은 계속 침해된다. NIST는 NIST SP 800-147[25], 800-155[26], 800-164에서 신뢰(trusted) 컴퓨팅 요구사항을 정의했다. 신뢰란, 플랫폼이 예상대로 동작하는 것을 의미한다. 즉, 소프트웨어 인벤토리(inventory)가 정확하고, 설정과 보안 통제가 적재적소에서 정상적으로 운영되는 등을 의미한다. 또한, 신뢰는 권한이 없는 사람에 의해 호스트의 소프트웨어와 설정이 변조되지 않았다는 것을 의미한다. 하드웨어 ROT는 컨테이너에만 존재하는 개념이 아니다. 그러나, 컨테이너 관리/보안 도구는 컨테이너 기술 아키텍처 전반에서 증명서를 활용하여, 컨테이너를 안전한 환경에서 실행할 수 있다.

신뢰 컴퓨팅은 다음과 같은 방법으로 구축할 수 있다.

1. 펌웨어/소프트웨어/설정 데이터를 측정한 후, RTM을 사용하여 실행해야 한다.
2. TPM과 같은 하드웨어 ROT에 측정값을 저장해야 한다.
3. 현재 측정값이 저장된 측정값과 동일한지 검증해야 한다. 동일하다면, 플랫폼이 예상대로 동작한다는 것을 증명할 수 있다.

TPM을 지원하는 장치는 보호/탐지 메커니즘을 하드웨어에서 구동할 수 있고, 부팅을 하는 과정에서 그 장치의 무결성을 검사할 수 있다. 이와 같은 신뢰성 및 무결성은 OS/부트로더를 넘어 컨테이너 런타임/앱으로 확장될 수 있다. 클라우드 서비스의 사용자가 하드웨어의 신뢰를 검증하기 위한 표준을 개발하고 있지만, 모든 클라우드 서비스가 고객에게 이 기능을 제공하지는 않을 것이다. 만약 기술적으로 검증할 수 없다면, 조직은 클라우드 제공자와 서비스 계약에 하드웨어 신뢰에 대한 요구사항을 반영해야 한다.

시스템의 복잡도는 증가하고 있고, 최근 위협은 깊숙한 곳에 포함되고 있다. 따라서, 보안은 하드웨어 및 펌웨어를 시작으로 모든 컨테이너 기술 컴포넌트로 확장되어야 한다. 이는 신뢰 컴퓨팅 모델을 균등하게 분산함으로써, 가장 신뢰할 수 있고 안전한 방법으로 컨테이너를 빌드-실행-오케스트레이션-관리할 수 있다.

신뢰 컴퓨팅 모델은 측정된/안전한 부팅으로 시작해야 한다. 측정된/안전한 부팅은 시스템을 검증할 수 있다. 신뢰 컴퓨팅 모델은 하드웨어를 기반으로 부트로더/OS 커널/OS 컴포넌트로 확장된 신뢰 체인을 구축하여, 부팅 메커니즘/시스템 이미지/컨테이너 런타임/컨테이너 이미지를 암호학적으로 검증할 수 있다. 컨테이너 기술의 경우, 신뢰 컴퓨팅을 하드웨어/하이퍼바이저/호스트 OS에 적용할 수 있으며, 컨테이너 전용 컴포넌트에 적용하기 위한 초기 작업이 진행 중이다.

이 문서를 작성하는 시점에서 NIST는 업계와 협력하여, 상용 제품을 기반으로 컨테이너 환경에 대한 신뢰 컴퓨팅을 보여줄 수 있는 참조 아키텍처를 구축하고 있다. [1]

[1] NIST IR 7904, Trusted Geolocation in the Cloud: Proof of Concept Implementation [27]

5. 컨테이너 위협 시나리오

4장에서 권고한 보호 대책의 효과를 확인하기 위해, 다음과 같은 컨테이너 위협 시나리오를 검토한다.

5.1. 이미지 내부 취약점에 대한 공격

컨테이너 환경에 대한 가장 일반적인 위협은 컨테이너 내부 소프트웨어에서 발생하는 애플리케이션 레벨의 취약점이다. 예를 들어, 조직은 일반적인 웹 앱 이미지를 만들 수 있다. 앱에 취약점이 있다면, 공격자는 그 취약점을 사용하여 컨테이너 내부의 앱을 침해할 수 있다. 침해된 경우, 공격자는 침해된 환경 내의 다른 시스템을 스캔하거나, 침해한 컨테이너 내에서 권한 상승을 시도하거나, 침해한 컨테이너를 다른 시스템을 공격하기 위해 남용(파일 드로퍼, C&C 엔드포인트 등)할 수 있다.

권고한 보호 대책을 적용한 조직은 다음과 같은 위협을 심층적으로 방어할 것이다.

1. 배치 프로세스 초기에 취약한 이미지를 탐지하고, 취약한 이미지가 배치되지 않도록 통제하면, 취약점이 운영 환경으로 유입되는 것을 예방할 것이다.

2. 컨테이너 인식 네트워크 모니터링/필터링을 적용하면, 다른 시스템에 대한 스캔을 시도하는 과정에서 다른 컨테이너에 대한 비정상적인 연결을 탐지할 것이다.

3. 컨테이너 인식 프로세스 모니터링 및 악성코드 탐지를 적용하면, 불명확하고 예상되지 않은 악의적인 프로세스가 실행되는 것을 탐지하고, 악의적인 프로세스가 컨테이너 환경에 생성한 데이터를 탐지할 것이다.

5.2. 컨테이너 런타임에 대한 공격

일반적인 상황은 아니지만, 컨테이너 런타임이 침해되면, 공격자는 컨테이너 런타임에 접근하여 호스트와 호스트의 모든 컨테이너를 공격할 수 있다.

이러한 위협 시나리오를 완화할 수 있는 방법은 다음과 같다.

1. MAC(Mandatory Access Control)은 프로세스와 파일 시스템을 정의된 경계에서 지속적으로 분리시키는 추가적인 분리대로서 동작할 수 있다.

2. 작업을 분리하면, 침해 범위가 호스트를 공유하는 동일한 중요도의 앱으로 제한될 것이다. 예를 들어, 웹 앱만 실행하는 호스트의 컨테이너 런타임이 침해되더라도, 금융 앱을 실행하는 다른 호스트의 컨테이너 런타임에는 영향을 미치지 않을 것이다.

3. 컨테이너 런타임의 취약점 상태를 보고하고, 취약한 컨테이너 런타임에 이미지 배치를 차단할 수 있는 보안 도구는 취약한 컨테이너 런타임에서 작업이 실행되는 것을 방지할 수 있다.

5.3. 감염된 이미지의 실행

공개된 위치에서 인증서를 확인할 수 없는 이미지를 쉽게 획득할 수 있다. 따라서, 공격자는 공격 대상이 사용한다고 알려진 이미지에 악의적인 소프트웨어를 포함시킬 수 있다. 예를 들어, 공격 대상이 특정 프로젝트에 대한 활성화된 게시판을 가지고 있고, 그 프로젝트의 웹 사이트에서 제공하는 이미지를 사용하고 있다는 것을 공격자가 알아냈다면, 공격자는 악의적인 버전의 이미지를 제작하여 공격에 사용하려 할 것이다.

이와 관련된 완화 방법은 다음과 같다.

1. 심사(vetted)되고, 테스트되고, 확인(validated)되고, 전자 서명된 이미지만 조직의 레지스트리에 업로드되도록 해야 한다.

2. 신뢰할 수 있는 이미지만 실행되도록 해야 한다. 이렇게 하면 심사되지 않은 외부의 이미지가 사용되는 것을 방지할 수 있다.

3. 이미지의 취약점과 악성코드를 자동으로 스캔해야 한다. 이렇게 하면 이미지에 내장된 루트킷과 같은 악의적인 코드를 탐지할 수 있다.

4. 컨테이너가 리소스를 남용하거나, 권한을 상승시키거나, 실행 파일을 실행할 수 없도록 컨테이너 런타임에 대한 통제를 구현해야 한다.

5. 컨테이너 수준에서 네트워크를 분리하여, 감염된 이미지가 영향을 미치는 반경을 최소한으로 제한해야 한다.

6. 컨테이너 런타임이 최소 권한 및 최소 접근 원칙에 따라 운영되도록 확인해야 한다.

7. 컨테이너 런타임에 대한 위협 프로필을 구축해야 한다. 위협 프로필에는 프로세스, 네트워크 접속, 파일 시스템 변경을 기본적으로 포함되어야 한다.

8. 이미지를 실행하기 전에 해시 및 전자 서명을 활용하여 이미지의 무결성을 확인해야 한다.

9. 허용 가능한 취약점의 심각도에 기반하여 이미지의 실행을 제한해야 한다. 예를 들어, 중간 이상의 CVSS 점수가 매겨진 취약점을 가진 이미지가 실행되는 것을 차단해야 한다.

6. 컨테이너 기술 생명 주기에서 보안 고려사항

컨테이너 기술을 설치/설정/배치하기 전에 상세하게 계획하는 것이 매우 중요하다. 이렇게 상세하게 계획하면, 컨테이너 환경을 가능한 한 안전하게 하고, 관련된 조직의 정책/외부의 규제/다른 요구사항을 모두 준수하게 도와준다.

컨테이너 기술과 가상화 솔루션의 계획/구현에 대한 권고사항은 상당히 유사하다. NIST SP 800-125 문서의 5장에는 가상화 솔루션에 대한 모든 권고사항이 수록되어 있다. 따라서, 이 문서에서는 그런 모든 권고사항을 반복하지 않을 것이다. 조직은 컨테이너 기술의 맥락에서 NIST SP 800-125 5장의 모든 권고사항(아래에서 나열한 예외사항은 제외)을 적용해야 한다. 컨테이너 기술의 맥락이란, 예를 들어 가상화 보안 정책을 대신하여 컨테이너 기술 보안 정책을 생성하는 등을 의미한다.

이 장에서는 NIST SP 800-125 5장의 권고사항에 대한 추가사항과 예외사항을 나열하였다.

6.1. 착수 단계

조직은 컨테이너가 다른 보안 정책에 어떤 영향을 미칠 수 있는지 검토해야 하며, 필요에 따라 영향을 받는 다른 보안 정책을 조정해야 한다. 예를 들어, 사고 대응(특히 포렌식) 및 취약점 관리 정책은 컨테이너의 특수한 요구사항을 고려하여 조정해야 할 필요가 있다.

컨테이너 기술을 도입하면, 조직의 기존 사고방식이나 소프트웨어 개발 방법론을 사용할 수 없을 수 있다. 컨테이너의 이점을 최대한으로 활용하기 위해서는 조직의 프로세스를 조정하여 앱을 개발/실행/운영할 때 새로운 방법을 지원해야 한다. 기존의 개발 실무, 패치 기법, 시스템 업그레이드 프로세스는 컨테이너 환경에 바로 적용할 수 없을 수 있다. 임직원이 새로운 모델에 적응하기 위해 노력하는 것도 중요하다. 기술 전환에 의해 발생하는 모든 잠재적인 문화 충격은 새로운 프로세스를 통해 해결할 수 있다. 소프트웨어 개발 생명 주기에 관련된 모든 인원을 교육/훈련하여, 앱을 빌드/저장/실행하는 새로운 방법에 익숙하게 할 수 있다.

6.2. 계획 및 설계 단계

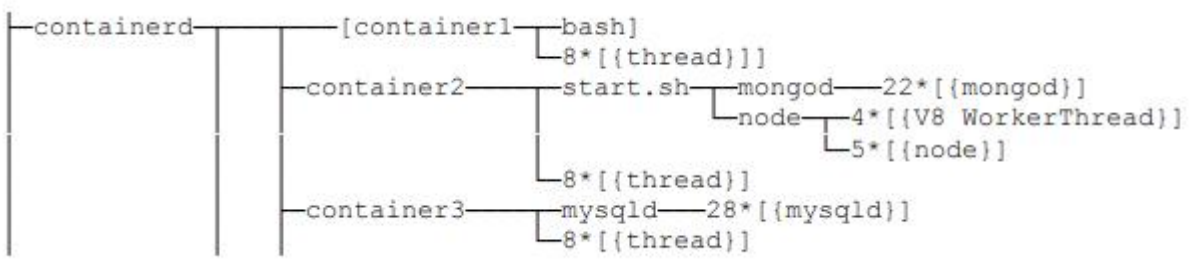
컨테이너 계획 및 설계 단계의 주요 고려사항은 포렌식이다. 컨테이너는 대부분 OS에 이미 존재하는 컴포넌트를 기반으로 빌드된다. 따라서, 컨테이너 환경에서 포렌식 수행을 위한 도구와 기술은 대부분 기존의 도구와 기술을 발전시킨 것이다. 컨테이너 및 이미지의 불변성은 포렌식 능력을 실제로 향상시킬 수 있다. 이미지가 수행해야 하는 것과 사고 中 실제 발생한 것의 구분이 더 명확하기 때문이다. 예를 들어, 웹 서버를 실행하기 위한 컨테이너가 메일 릴레이를 시작하면, 새로운 프로세스(메일 릴레이)가 컨테이너 생성에 사용된 원본 이미지의 일부가 아니라는 것이 명확하다. 기존 플랫폼에서는 OS와 앱의 분리가 더 적기 때문에, 이렇게 구별하는 것이 훨씬 어려울 수 있다.

프로세스/메모리/디스크 사고 대응 활동에 익숙한 조직은 컨테이너에서도 대체로 유사하다는 것을 알게 될 것이다. 하지만, 몇 가지 차이점에 주의해야 한다.

컨테이너는 일반적으로 호스트 OS에서 가상화된 계층형 파일 시스템을 사용한다. 호스트에서 경로를 검사하면, 일반적으로 이러한 계층의 외부 경계만 표시되며, 계층 내부의 파일과 데이터는 표시되지 않는다. 따라서, 컨테이너 환경에서 사고에 대응할 때, 사용 중인 특정 스토리지 제공자를 식별해야 한다. 또한, 오프라인에서 특정 스토리지의 콘텐츠를 적절하게 검사하는 방법을 이해해야 한다.

컨테이너는 일반적으로 가상화 오버레이 네트워크를 사용하여 서로 연결된다. 오버레이 네트워크는 캡슐화 및 암호화를 사용하여, 기존 네트워크를 통해 트래픽을 안전하게 라우팅할 수 있다. 그러나, 컨테이너 네트워크에서 발생한 사고를 조사할 때, 특히 라이브 패킷을 분석할 때, 사용되는 도구는 이러한 가상화 네트워크를 인식해야 한다. 또한, 기존 도구로 트래픽을 분석하기 위해 가상화 네트워크에서 포함된 IP 프레임을 추출하는 방법을 이해해야 한다.

컨테이너 내부의 프로세스 및 메모리 동작은 기존 앱에서 관찰되는 것과 대체로 유사하지만, 상위 프로세스는 다르다. 예를 들어, 컨테이너 런타임은 컨테이너 내부의 모든 프로세스를 중첩된 방식으로 생성할 것이다. 컨테이너 런타임은 최상위 프로세스이며, 각 컨테이너는 1단계 하위 프로세스를 갖고, 컨테이너 내부의 각 프로세스는 2단계 하위 프로세스를 갖는다. 아래에서 예를 보인다.



6.3. 구현 단계

컨테이너 기술을 설계한 이후, 다음 단계는 솔루션을 운영 환경에 배치하기에 앞서 설계의 프로토타입을 구현하고 테스트하는 것이다. 컨테이너 기술은 VM 기술 수준의 실행 상태 모니터링(introspection) 기능을 제공하지 않는다는 것에 유의해야 한다.

NIST SP 800-125[1]에는 가상화 기술을 운영 환경에 배치하기 앞서 평가해야 하는 가상화 기술의 다양한 측면을 언급하였다. 여기에는 인증, 연결/네트워크, 앱의 기능, 관리, 성능, 기술 자체의 보안이 포함된다. 이에 추가적으로 컨테이너 기술의 분리 능력을 평가하는 것도 중요하다. 컨테이너 내부 프로세스는 허가된 모든 리소스에 접근할 수 있어야 하지만, 다른 리소스에는 접근할 수 없어야 한다.

구현에는 컨테이너와 클라우드 네이티브 앱에 초점이 맞춰진 새로운 보안 도구가 필요할 수 있다. 마지막으로 배치에는 기존 보안 통제/기술(보안 이벤트 로그, 네트워크 관리, 코드 저장소, 인증 서버 등)의 설정을 변경함으로써 기존 보안 통제/기술을 컨테이너와 직접 연결/동작하게 하고, 새로운 컨테이너 보안 도구와 통합할 수 있다.

프로토타입에 대한 평가가 완료되고, 컨테이너 기술을 운영 환경에서 사용하기 위해 준비되면, 초기에는 소수의 앱을 대상으로 컨테이너를 사용해야 한다. 소수의 앱에서 발생한 문제는 다수의 앱에도 영향을 줄 수 있다. 이러한 문제를 조기에 식별하면, 추가적인 구축 이전에 문제를 해결하는데 도움이 된다. 단계적 배치는 개발자와 IT 직원(예 : 시스템 관리자, 헬프 데스크)이 컨테이너의 사용 및 지원에 대해 교육받을 수 있는 시간도 벌어준다.

6.4. 운영 및 유지 단계

운영 프로세스는 컨테이너 기술의 보안을 유지하기 위해 특히 중요하다. 따라서 정기적으로 수행되어야 한다. 운영 프로세스에는 모든 이미지를 업데이트하고, 오래된 이미지를 대체하기 위해 업데이트된 이미지를 컨테이너에 배포하는 것이 포함된다. 취약점 관리, 호스트/오케스트레이터 등 지원 레이어에 대한 업데이트와 같은 보안 모범 사례도 운영과 관련된 중요 작업이다. 컨테이너 보안/모니터링 도구는 기존의 SIEM(Security Information and Event Management)과 통합하여, 모든 보안 도구/프로세스에 컨테이너와 관련된 이벤트가 전달되도록 해야 한다.

컨테이너 환경에서 보안 사고가 발생하는 경우, 조직은 컨테이너에 최적화된 프로세스/도구로 대응할 준비가 되어있어야 한다. 컴퓨터 보안 사고 처리 가이드라인[1]은 컨테이너 환경에서도 적용할 수 있다. 그러나, 컨테이너를 채택한 조직에서는 컨테이너 보안의 특수한 측면에 대한 대응을 강화해야 한다.

[1] NIST SP 800-61, Computer Security Incident Handling Guide [28]

- 컨테이너 앱은 기존의 운영팀이 아닌 다른 팀에 의해서 실행될 수 있다. 따라서, 조직은 컨테이너 운영에 책임이 있는 모든 팀이 사고 대응 계획 작성에 참여하여, 역할을 이해하도록 해야 한다.

- 컨테이너는 영구적이지 않고, 컨테이너의 관리는 자동화되는 것이 특징이다. 따라서, 조직에서 기존에 사용해 왔던 자산 관리 정책 및 도구에 적합하지 않을 수 있다. 사고 대응팀은 반드시 컨테이너의 역할/소유자/중요도를 확인할 수 있어야 하고, 이 데이터(컨테이너의 역할/소유자/중요도)를 사고 대응 프로세스에서 사용할 수 있어야 한다.

- 컨테이너 관련 사고에 대응하기 위한 절차를 명확하게 정의해야 한다. 예를 들어, 수백 개의 컨테이너에서 사용 중인 이미지가 침해된다면, 사고 대응팀은 공격을 중지시키기 위해 침해된 이미지를 사용 중인 모든 컨테이너를 종료할 필요가 있다. 한 개의 취약점은 오랫동안 많은 시스템에 문제를 일으킬 수 있었다. 컨테이너의 경우, 기존 시스템에 패치를 설치하는 대신, 이미지를 리빌드/재배치해야 할 수 있다. 이러한 대응의 변화는 팀 및 승인의 변화를 수반한다. 따라서, 사전에 변화를 이해하고 준비해야 한다.

- 앞서 언급한 것처럼, 컨테이너 환경에서는 로그 및 기타 포렌식 데이터가 다르게 저장될 수 있다. 사고 대응팀은 데이터를 수집하는데 필요한 다른 도구/기법을 숙련해야 하며, 프로세스를 문서화해야 한다.

6.5. 폐기 단계

앱의 필요에 따라 컨테이너를 자동으로 배치/폐기하는 기능은 시스템의 효율성을 높여준다. 그러나, 기록 보존, 포렌식, 이벤트 데이터에 대해서는 해결해야 할 몇 가지 문제도 발생한다. 조직에서는 데이터 보존 정책을 충족시키기 위한 적절한 메커니즘을 마련해야 한다. 해결해야 문제로는 컨테이너/이미지를 폐기하는 방법, 폐기에 앞서 컨테이너에서 추출해야 하는 데이터의 종류, 데이터를 추출하는 방법, 컨테이너에서 사용한 암호키를 폐기/삭제하는 방법 등이 있다.

조직의 폐기 계획(disposal plan)에는 컨테이너 환경을 지원하는 데이터 저장소 및 저장 매체를 포함시켜야 한다.

7. 결론

컨테이너는 앱을 빌드하고 실행하는 방법을 혁신적으로 변화시킨다. 하지만, 컨테이너에 새로운 보안 모범 사례가 반드시 필요한 것은 아니다. 컨테이너 보안에서 가장 중요한 측면은 기존에 확립된 기술과 원칙을 개선하는 것이다. 이 문서에서는 컨테이너 기술에 대한 고유한 위험에 대응하고자, 일반적인 보안 권고사항을 업데이트하고 확장하였다.

이 문서에서는 컨테이너 및 VM 환경에서 보안의 차이점에 대해 이미 논의했다. 이에 대해서는 그 부분을 요약하면, 도움이 될 것이다.

컨테이너 환경에서는 더욱 많은 엔티티가 존재한다. 따라서, 보안 프로세스/도구를 확장할 수 있어야 한다. 확장이란 데이터베이스에서 지원하는 객체의 총수량, 정책의 효율적/자율적 관리 수준을 의미한다. 많은 조직이 수백 대의 VM 전체의 보안을 관리하는 부담에 느끼고 있다. 조직에서 컨테이너 중심의 아키텍처가 기본이 되어, 수만 개의 컨테이너를 책임져야 한다면, 보안과 관련한 업무는 자동화 및 효율성을 강조해야 한다.

컨테이너의 경우, 변화의 정도가 크다. 앱의 업데이트 주기는 1년에 몇 차례에서, 1주일 심지어 하루에 수 차례로 변경된다. 수동으로 할 수 있었던 것이 더 이상은 불가능하다. 자동화가 중요한 이유는 엔티티의 수가 많을 뿐만 아니라, 엔티티가 변하는 빈도가 잦기 때문이다. 정책은 한 곳에서 작성되어야 하고, 소프트웨어에 의해 전체 환경에 적용되어야 한다. 컨테이너를 도입하는 조직은 이러한 변화의 빈도를 관리할 수 있도록 준비해야 한다. 이는 근본적으로 새로운 운영 업무 및 조직의 발전이 필요하다.

컨테이너를 사용하면, 보안에 대한 책임이 상당 부분 개발자에게 넘어간다. 따라서, 조직은 개발자들이 타당한 결정을 내리는데 필요한 모든 정보/기술/도구를 이용할 수 있도록 해야 한다. 또한, 보안팀은 개발 주기 전반에서 적극적으로 품질을 통제해야 한다. 이러한 전환에 성공하면, 조직은 이전보다 신속한 취약점 대응 및 운영 부담 감소라는 보안상 이점을 얻는다.

보안은 컨테이너만큼 이식이 용이해야 한다. 따라서, 조직은 플랫폼/환경에 독립적이고 공개된 기법/도구를 채택해야 한다. 개발자들은 한 환경에서 빌드하고, 다른 환경에서 테스트하고, 또 다른 환경에서 배치를 수행할 것이다. 따라서, 이러한 모든 환경에서 평가/시행에 일관성을 갖는 것이 핵심이다. 이식성에는 환경에 대한 이식뿐만 아니라 시간에 대한 이식도 포함된다. 지속적 통합/배치는 개발/배치 주기에서 각 단계 사이에 존재하는 기존의 경계를 허문다. 조직은 이미지를 생성하고, 레지스트리에 이미지를 저장하고, 컨테이너에서 이미지를 실행하는 전반적인 과정에서 일관되고 자동화된 업무를 보장해야 한다.

이러한 변화를 처리하면, 컨테이너는 조직의 전반적인 보안을 향상시키는데 영향을 줄 것이다. 컨테이너의 불변성/선언적 특성은 수동적인 개입을 최소화하고, 앱이 변화함에 따라 스스로 업데이트되는 “자동화된 앱 중심의 보안”이라는 조직의 비전을 실현할 수 있다.

부록 A – 非 핵심 컴포넌트의 보안을 위한 NIST 문서

이 부록에서는 컨테이너 기술의 非 핵심 컴포넌트의 보안을 위한 NIST 문서를 나열하였다. 非 핵심 컴포넌트에는 개발 시스템, 테스트/승인 시스템, 관리자 시스템, 호스트 하드웨어, 가상 머신 관리자가 포함된다. 이 외에도 다른 조직에서 작성한 많은 문서를 이용할 수 있다.

문서명 및 URI	적용범위
SP 800-40 Revision 3, Guide to Enterprise Patch Management Technologies (전사 패치 관리 기술 가이드라인) https://doi.org/10.6028/NIST.SP.800-40r3	모든 IT 제품/시스템
SP 800-46 Revision 2, Guide to Enterprise Telework, Remote Access, and Bring Your Own Device Security (전사 재택 근무, 원격 접속, BYOD 보안 가이드라인) https://doi.org/10.6028/NIST.SP.800-46r2	클라이언트 OS/앱
SP 800-53 Revision 4, Security and Privacy Controls for Federal Information Systems and Organizations (정보 시스템 및 조직을 위한 보안 및 개인 정보 통제) https://doi.org/10.6028/NIST.SP.800-53r4	모든 IT 제품/시스템
SP 800-70 Revision 3, National Checklist Program for IT Products: Guidelines for Checklist Users and Developers (IT 제품에 대한 사용자 및 개발자 체크 리스트 가이드라인) https://doi.org/10.6028/NIST.SP.800-70r3	서버/클라이언트 OS, 서버/클라이언트 앱
SP 800-83 Revision 1, Guide to Malware Incident Prevention and Handling for Desktops and Laptops (데스크톱 및 랩톱에 대한 악성코드 예방 및 처리 가이드라인) https://doi.org/10.6028/NIST.SP.800-83r1	클라이언트 OS/앱
SP 800-123, Guide to General Server Security (일반 서버 보안 가이드라인) https://doi.org/10.6028/NIST.SP.800-123	서버
SP 800-124 Revision 1, Guidelines for Managing the Security of Mobile Devices in the Enterprise (전사 모바일 기기 보안 관리 가이드라인) https://doi.org/10.6028/NIST.SP.800-124r1	모바일 기기
SP 800-125, Guide to Security for Full Virtualization Technologies (全 가상 기술 보안 가이드라인) https://doi.org/10.6028/NIST.SP.800-125	하이퍼바이저 및 가상 머신

문서명 및 URI	적용범위
SP 800-125A, Security Recommendations for Hypervisor Deployment / Second Draft (하이퍼바이저 배치 관련 보안 권고사항 초안) https://csrc.nist.gov/publications/detail/sp/800-125A/draft	하이퍼바이저 및 가상 머신
SP 800-125B, Secure Virtual Network Configuration for Virtual Machine Protection (가상 머신 보호를 위한 안전한 가상 네트워크 설정) https://doi.org/10.6028/NIST.SP.800-125B	하이퍼바이저 및 가상 머신
SP 800-147, BIOS Protection Guidelines (BIOS 보호 가이드라인) https://doi.org/10.6028/NIST.SP.800-147	클라이언트 하드웨어
SP 800-155, BIOS Integrity Measurement Guidelines (BIOS 무결성 측정 가이드라인) https://csrc.nist.gov/publications/detail/sp/800-155/draft	클라이언트 하드웨어
SP 800-164, Guidelines on Hardware-Rooted Security in Mobile Devices (모바일 기기에서 하드웨어 기반 보안 가이드라인) https://csrc.nist.gov/publications/detail/sp/800-164/draft	모바일 기기

표 1 : 非 핵심 컴포넌트의 보안을 위한 NIST 문서

부록 B – NIST SP 800-53 內 컨테이너 기술 관련 보안 통제

"NIST SP 800-53 리비젼 4"에 기술되어 있는 컨테이너 기술 관련 중요한 보안 통제를 표 2에 나열하였다.

NIST SP 800-53 통제	관련된 통제	관련문헌
AC-2, 계정 관리	AC-3, AC-4, AC-5, AC-6, AC-10, AC-17, AC-19, AC-20, AU-9, IA-2, IA-4, IA-5, IA-8, CM-5, CM-6, CM-11, MA-3, MA-4, MA-5, PL-4, SC-13	
AC-3, 접근 시행	AC-2, AC-4, AC-5, AC-6, AC-16, AC-17, AC-18, AC-19, AC-20, AC-21, AC- 22, AU-9, CM-5, CM-6, CM-11, MA-3, MA-4, MA-5, PE-3	
AC-4, 정보 흐름 시행	AC-3, AC-17, AC-19, AC-21, CM-6, CM-7, SA-8, SC-2, SC-5, SC-7, SC-18	
AC-6, 최소 권한	AC-2, AC-3, AC-5, CM-6, CM-7, PL-2	
AC-17, 원격 접근	AC-2, AC-3, AC-18, AC-19, AC-20, CA-3, CA-7, CM-8, IA-2, IA-3, IA-8, MA-4, PE-17, PL-4, SC-10, SI-4	- NIST SP 800-46, 800-77, 800-113, 800-114, 800-121
AT-3, 역할 기반 보안 훈련	AT-2, AT-4, PL-4, PS-7, SA-3, SA-12, SA-16	- C.F.R. 5장 C절(5C.F.R.930.301) - NIST SP 800-16, 800-50
AU-2, 감사 이벤트	AC-6, AC-17, AU-3, AU-12, MA-4, MP-2, MP-4, SI-4	- NIST SP 800-92 - https://idmanagement.gov/
AU-5, 감사 실패 대응	AU-4, SI-12	
AU-6, 감사 리뷰, 분석, 레포팅	AC-2, AC-3, AC-6, AC-17, AT-3, AU-7, AU-16, CA-7, CM5, CM-10, CM-11, IA-3, IA-5, IR-5, IR-6, MA-4, MP-4, PE3, PE-6, PE-14, PE-16, RA-5, SC-7, SC-18, SC-19, SI-3, SI-4, SI-7	
AU-8, 타임 스탬프	AU-3, AU-12	
AU-9, 감사 정보 보호	AC-3, AC-6, MP-2, MP-4, PE-2, PE-3, PE-6	
AU-12, 감사 생성	AC-3, AU-2, AU-3, AU-6, AU-7	

NIST SP 800-53 통제	관련된 통제	관련문헌
CA-9, 내부 시스템 연결	AC-3, AC-4, AC-18, AC-19, AU-2, AU-12, CA- 7, CM-2, IA-3, SC-7, SI-4	
CM-2, 설정 베이스라인	CM-3, CM-6, CM-8, CM-9, SA-10, PM-5, PM-7	- NIST SP 800-128
CM-3, 설정 변경 통제	CA-7, CM-2, CM-4, CM-5, CM-6, CM-9, SA-10, SI- 2, SI12	- NIST SP 800-128
CM-4, 보안 영향 분석	CA-2, CA-7, CM-3, CM-9, SA-4, SA-5, SA-10, SI-2	- NIST SP 800-128
CM-5, 변경 관련 접근 제한	AC-3, AC-6, PE-3	
CM-6, 설정	AC-19, CM-2, CM-3, CM-7, SI-4	- OMB Memoranda 07-11, 07-18, 08-22 - NIST SP 800-70, 800-128 - https://nvd.nist.gov - https://checklists.nist.gov - https://www.nsa.gov
CM-7, 최소 기능	AC-6, CM-2, RA-5, SA-5, SC-7	- DoD Instruction 8551.01
CM-9, 설정 관리 계획	CM-2, CM-3, CM-4, CM-5, CM-8, SA-10	- NIST SP 800-128
CP-2, 비상 계획	AC-14, CP-6, CP-7, CP-8, CP-9, CP-10, IR-4, IR-8, MP2, MP-4, MP-5, PM-8, PM-11	- Federal Continuity Directive 1 - NIST SP 800-34
CP-9, 정보 시스템 백업	CP-2, CP-6, MP-4, MP-5, SC-13	- NIST SP 800-128
CP-10, 정보 시스템 복구	CA-2, CA-6, CA-7, CP-2, CP-6, CP-7, CP-9, SC-24	- Federal Continuity Directive 1 - NIST SP 800-34
IA-2, 사용자 식별 및 인증	AC-2, AC-3, AC-14, AC-17, AC-18, IA-4, IA-5, IA-8	- HSPD-12 - OMB Memoranda 04-04, 06-16, 11-11 - FIPS 201 - NIST SP 800-63, 800-73, 800-76, 800-78 - FICAM Roadmap and Implementation Guidance - https://idmanagement.gov/
IA-4, ID 관리	AC-2, IA-2, IA-3, IA-5, IA-8, SC-37	- FIPS 201 - NIST SP 800-73, 800-76, 800-78

NIST SP 800-53 통제	관련된 통제	관련문헌
IA-5, 인증 관리	AC-2, AC-3, AC-6, CM-6, IA-2, IA-4, IA-8, PL-4, PS-5, PS-6, SC-12, SC-13, SC-17, SC-28	- OMB Memoranda 04-04, 11-11 - FIPS 201 - NIST SP 800-63, 800-73, 800-76, 800-78 - FICAM Roadmap and Implementation Guidance - https://idmanagement.gov/
IR-1, 사고 대응 정책 및 절차	PM-9	- NIST SP 800-12, 800-61, 800-83, 800-100
IR-4, 사고 처리	AU-6, CM-6, CP-2, CP-4, IR-2, IR-3, IR-8, PE-6, SC-5, SC-7, SI-3, SI-4, SI-7	- EO 13587 - NIST SP 800-61
MA-2, 통제된 유지 보수	CM-3, CM-4, MA-4, MP-6, PE-16, SA-12, SI-2	
MA-4, 원격 유지 보수(nonlocal maintenance)	AC- 2, AC-3, AC-6, AC-17, AU-2, AU-3, IA-2, IA-4, IA-5, IA-8, MA-2, MA-5, MP-6, PL-2, SC-7, SC-10, SC-17	- FIPS 140-2, 197, 201 - NIST SP 800-63, 800-88 - CNSS Policy 15
PL-2, 시스템 보안 계획	AC-2, AC-6, AC-14, AC-17, AC-20, CA-2, CA-3, CA-7, CM-9, CP-2, IR-8, MA-4, MA-5, MP-2, MP-4, MP-5, PL-7, PM-1, PM-7, PM-8, PM-9, PM-11, SA-5, SA-17	- NIST SP 800-18
PL-4, 행동 규칙	AC-2, AC-6, AC-8, AC-9, AC-17, AC-18, AC-19, AC-20, AT-2, AT-3, CM-11, IA-2, IA-4, IA-5, MP-7, PS-6, PS-8, SA-5	- NIST SP 800-18
RA-2, 보안 분류	CM-8, MP-4, RA-3, SC-7	- FIPS 199 - NIST SP 800-30, 800-39, 800-60
RA-3, 위험 평가	RA-2, PM-9	- OMB Memorandum 04-04 - NIST SP 800-30, 800-39 - https://idmanagement.gov/
SA-10, 개발자 설정 관리	CM-3, CM-4, CM-9, SA-12, SI-2	- NIST SP 800-18
SA-11, 개발자 보안 테스트 및 평가	CA-2, CM-4, SA-3, SA-4, SA-5, SI-2	- ISO/IEC 15408 - NIST SP 800-53A - https://nvd.nist.gov - http://cwe.mitre.org - http://cve.mitre.org - http://capec.mitre.org

NIST SP 800-53 통제	관련된 통제	관련문헌
SA-15, 개발 프로세스, 표준, 도구	SA-3, SA-8	
SA-19, 컴포넌트 무결성(Authenticity)	PE-3, SA-12, SI-7	
SC-2, 애플리케이션 파티셔닝	SA-4, SA-8, SC-3	
SC-4, 공유 리소스의 정보	AC-3, AC-4, MP-6	
SC-6, 리소스 가용성		
SC-8, 전송 정보의 기밀성 및 무결성	AC-17, PE-4	
SI-2, 결함 수정	CA-2, CA-7, CM-3, CM-5, CM-8, MA-2, IR-4, RA-5, SA10, SA-11, SI-11	- NIST SP 800-40, 800-128
SI-4, 정보 시스템 모니터링	AC-3, AC-4, AC-8, AC-17, AU-2, AU-6, AU-7, AU-9, AU12, CA-7, IR-4, PE-3, RA-5, SC-7, SC-26, SC-35, SI-3, SI-7	- NIST SP 800-61, 800-83, 800-92, 800-137
SI-7, 소프트웨어/펌웨어/정보의 무결성	SA-12, SC-8, SC-13, SI-3	- NIST SP 800-147, 800-155

표 2 : 컨테이너 기술 보안에 대한 NIST SP 800-53의 보안 통제

NIST 사이버 보안 프레임워크의 하위 카테고리 중에서 컨테이너 기술 보안과 관련하여 가장 중요한 것들을 아래 목록에서 보인다.

- 식별(Identify) : 자산 관리(Asset Management)
  - ID.AM-3 : 조직의 소통과 데이터 흐름을 매핑한다.
  - ID.AM-5 : 리소스의 분류/중요도/가치를 기반으로 리소스(예 : 하드웨어, 기기, 데이터, 소프트웨어)의 우선 순위를 정한다.
- 식별(Identify) : 위험 평가(Risk Assessment)
  - ID.RA-1 : 자산의 취약점을 식별하고 문서화한다.
  - ID.RA-3 : 내/외부 외협을 식별하고 문서화한다.
  - ID.RA-4 : 잠재적인 사업 영향과 가능성을 식별한다.
  - ID.RA-5 : 위협/취약점/가능성/영향을 사용하여 위험을 결정한다.
  - ID.RA-6 : 위험에 대한 대응 방법을 식별하고 우선 순위를 매긴다.
- 예방(Protect) : 접근 통제(Acccess Control)
  - PR.AC-1 : 승인된 기기와 사용자에 대한 ID 및 크리덴셜을 관리한다.
  - PR.AC-2 : 자산에 대한 물리적인 접근을 관리하고 보호한다.
  - PR.AC-3 : 원격 접근을 관리한다.
  - PR.AC-4 : 최소 권한의 법칙과 직무 분리를 통해 접근 허가를 관리한다.

- 예방(Protect) : 인식 및 훈련(Awareness and Training)
  - PR.AT-2 : 특권이 있는 사용자가 역할과 책임을 이해한다.
  - PR.AT-5 : 물리 보안 담당자 및 정보 보안 담당자가 역할과 책임을 이해한다.
- 예방(Protect) : 데이터 보안(Data Security)
  - PR.DS-2 : 전송 데이터를 보호한다.
  - PR.DS-4 : 가용성을 보장하기 위해 적절한 용량을 보장한다.
  - PR.DS-5 : 데이터 유출에 대한 보호를 구현한다.
  - PR.DS-6 : 무결성을 검사하는 메커니즘을 사용하여, 소프트웨어/펌웨어/정보의 무결성을 검증한다.
- 예방(Protect) : 정보 보호 프로세스 및 절차(Information Protection Processes and Procedures)
  - PR.IP-1 : IT(Information Technology)/ICS(Industrial Control System)에 대한 설정의 베이스라인을 생성하고 관리한다.
  - PR.IP-3 : 설정 변경을 통제하는 프로세스를 적절하다.
  - PR.IP-6 : 정책에 따라 데이터를 폐기한다.
  - PR.IP-9 : 대응 계획(사고 대응 및 업무 연속) 및 복구 계획(사고 복구 및 재해 복구)이 적절하고 관리된다.
  - PR.IP-12 : 취약점 관리 계획을 개발하고 구현한다.
- 예방(Protect) : 유지 보수(Maintenance)
  - PR.MA-1 : 조직의 자산을 승인/통제되는 도구를 이용하여 시기 적절하게 유지 보수 및 정비하고, 기록한다.
  - PR.MA-2 : 조직의 자산에 대한 원격 유지 보수는 승인/기록되고, 승인되지 않은 접근을 방지하는 방법으로 수행된다.
- 예방(Protect) : 보호 기술(Protective Technology)
  - PR.PT-1 : 정책에 따라 감사/로그 기록을 정의/문서화/구현/검토한다.
  - PR.PT-3 : 시스템/자산에 대한 접근을 최소 기능의 원칙으로 통제한다.
- 탐지(Detect) : 이상 징후 및 이벤트(Anomalies and Events)
  - DE.AE-2 : 탐지된 이벤트를 분석하여, 공격의 목표와 방법을 확인한다.
- 탐지(Detect) : 지속적인 보안 모니터링(Security Continuous Monitoring)
  - DE.CM-1 : 네트워크를 모니터링 하여, 잠재적인 사이버 보안 이벤트를 탐지한다.
  - DE.CM-7 : 승인되지 않은 인원/연결/기기/소프트웨어를 모니터링한다.
- 대응(Respond) : 대응 계획(Response Planning)
  - RS.RP-1 : 이벤트 발생 중/후에 대응 계획을 실행한다.

- 대응(Respond) : 분석(Analysis)
  - RS.AN-1 : 탐지 시스템의 알림을 분석한다.
  - RS.AN-3 : 포렌식을 수행한다.
- 대응(Respond) : 완화(Mitigation)
  - RS.MI-1 : 사고를 억제한다.
  - RS.MI-2 : 사고를 완화한다.
  - RS.MI-3 : 새롭게 식별된 취약점을 완화하거나, 문서화(위험 수용)한다.
- 복구(Recover) : 복구 계획(Recovery Planning)
  - RC.RP-1 : 이벤트 중/후에 복구 계획을 실행한다.

표 3은 컨테이너 기술을 통해 충족되는 NIST SP 800-53 리비젼 4의 보안 통제 목록이다. 가장 오른쪽 컬럼은 NISP SP 800-53의 보안 통제에 대응되는 본 문서의 관련 항목이다.

NIST SP 800-53 통제	컨테이너 기술 적합성	본 문서의 관련 항목
CM-3, 설정 변경 통제	이미지를 사용하면, 앱에 대한 변경 통제를 관리하기 쉽다.	2.1, 2.2, 2.3, 2.4, 4.1
SC-2, 애플리케이션 파티셔닝	컨테이너 또는 다른 가상화 기술을 사용하여 사용자 기능과 관리자 기능을 부분적으로 분리할 수 있다. 따라서, 다른 컨테이너에서 기능이 수행된다.	2(소개), 2.3, 4.5.2
SC-3, 보안 기능 분리	컨테이너 또는 다른 가상화 기술을 사용하여 보안 기능과 非 보안 기능을 부분적으로 분리할 수 있다. 따라서, 다른 컨테이너에서 기능이 수행된다.	2(소개), 2.3, 4.5.2
SC-4, 공유 리소스의 정보	컨테이너 기술은 각 컨테이너가 공유 리소스에 접근하는 것을 제한하도록 설계된다. 따라서, 실수로 정보가 한 컨테이너에서 다른 컨테이너로 유출될 수 없다.	2(소개), 2.2, 2.3, 4.4
SC-6, 리소스 가용성	각 컨테이너에서 사용 가능한 최대 리소스를 지정한다. 어떤 컨테이너도 과도하게 리소스를 사용할 수 없다. 따라서, 리소스의 가용성이 보호된다.	2.2, 2.3
SC-7, 경계 보호	컨테이너 사이의 통신을 제한하기 위해 컨테이너 사이의 경계를 설정하고 시행할 수 있다.	2(소개), 2.2, 2.3, 4.4

NIST SP 800-53 통제	컨테이너 기술 적합성	본 문서의 관련 항목
SC-39, 프로세스 분리	다수의 컨테이너가 동일한 호스트에서 프로세스를 동시에 실행할 수 있다. 하지만, 프로세스는 서로 분리된다.	2(소개), 2.1, 2.2, 2.3, 4.4
SI-7, 소프트웨어/펌웨어/정보의 무결성	이미지의 콘텐츠가 무단으로 변경된 것을 쉽게 탐지할 수 있고, 이상이 없는 이미지의 복사본으로 대체된다.	2.3, 4.1, 4.2
SI-14, 非 영속성	컨테이너에서 실행 중인 이미지는 필요에 따라 새로운 버전의 이미지로 대체된다. 실행 중인 이미지의 데이터, 파일, 실행파일, 다른 정보는 영속되지 않는다.	2.1, 2.3, 4.1

표 3 : 컨테이너 기술을 통해 충족되는 NIST SP 800-53 통제

표 4는 컨테이너 기술을 통해 충족되는 NIST 사이버 보안 프레임워크[30]의 하위 카테고리이다. 가장 오른쪽 컬럼은 사이버 보안 프레임워크 하위 카테고리에 대응되는 본 문서의 관련 항목이다.

사이버 보안 프레임워크 하위 카테고리	컨테이너 기술 적합성	본 문서의 관련 항목
PR.DS-4 : 가용성을 보장하기 위해 적절한 용량을 보장한다.	각 컨테이너에서 사용 가능한 최대 리소스를 지정한다. 어떤 컨테이너도 과도하게 리소스를 사용할 수 없다. 따라서, 리소스의 가용성이 보호된다.	2.2, 2.3
PR.DS-5 : 데이터 유출에 대한 보호를 구현한다.	컨테이너 기술은 각 컨테이너가 공유 리소스에 접근하는 것을 제한하도록 설계된다. 따라서, 실수로 정보가 한 컨테이너에서 다른 컨테이너로 유출될 수 없다.	2(소개), 2.2, 2.3, 4.4
PR.DS-6 : 무결성을 검사하는 메커니즘을 사용하여, 소프트웨어/펌웨어/정보의 무결성을 검증한다.	이미지의 콘텐츠가 무단으로 변경된 것을 쉽게 탐지할 수 있고, 이상이 없는 이미지의 복사본으로 대체된다.	2.3, 4.1, 4.2
PR.DS-7 : 개발/테스트 환경과 운영 환경을 분리한다.	컨테이너를 사용하면, 개발/테스트/운영 환경을 분리하기 쉽다. 동일한 이미지는 수정없이 모든 환경에서 사용할 수 있다.	2.1, 2.3
PR.IP-3 : 설정 변경을 통제하는 프로세스를 적절하다.	이미지를 사용하면, 앱에 대한 변경 통제를 관리하기 쉽다.	2.1, 2.2, 2.3, 2.4, 4.1

표 4 : 컨테이너 기술을 통해 충족되는 NIST 사이버 보안 프레임워크의 하위 카테고리

이러한 통제 및 가이드라인을 구현하는 방법은 아래의 NIST 문서에서 수록되어 있다.

- FIPS 140-2, Security Requirements for Cryptographic Modules
- FIPS 197, Advanced Encryption Standard (AES)
- FIPS 199, Standards for Security Categorization of Federal Information and Information Systems
- FIPS 201-2, Personal Identity Verification (PIV) of Federal Employees and Contractors
- SP 800-12 Rev. 1, An Introduction to Information Security
- Draft SP 800-16 Rev. 1, A Role-Based Model for Federal Information Technology/Cybersecurity Training
- SP 800-18 Rev. 1, Guide for Developing Security Plans for Federal Information Systems
- SP 800-30 Rev. 1, Guide for Conducting Risk Assessments
- SP 800-34 Rev. 1, Contingency Planning Guide for Federal Information Systems
- SP 800-39, Managing Information Security Risk: Organization, Mission, and Information System View
- SP 800-40 Rev. 3, Guide to Enterprise Patch Management Technologies
- SP 800-46 Rev. 2, Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security
- SP 800-50, Building an Information Technology Security Awareness and Training Program
- SP 800-52 Rev. 1, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
- SP 800-53 Rev. 4, Security and Privacy Controls for Federal Information Systems and Organizations
- SP 800-53A Rev. 4, Assessing Security and Privacy Controls in Federal Information Systems and Organizations: Building Effective Assessment Plans
- SP 800-60 Rev. 1 Vol. 1, Guide for Mapping Types of Information and Information Systems to Security Categories
- SP 800-61 Rev. 2, Computer Security Incident Handling Guide
- SP 800-63 Rev. 3, Digital Identity Guidelines
- SP 800-70 Rev. 3, National Checklist Program for IT Products: Guidelines for Checklist Users and Developers
- SP 800-73-4, Interfaces for Personal Identity Verification
- SP 800-76-2, Biometric Specifications for Personal Identity Verification
- SP 800-77, Guide to IPsec VPNs
- SP 800-78-4, Cryptographic Algorithms and Key Sizes for Personal Identification Verification (PIV)

- SP 800-81-2, Secure Domain Name System (DNS) Deployment Guide
- SP 800-83 Rev. 1, Guide to Malware Incident Prevention and Handling for Desktops and Laptops
- SP 800-88 Rev. 1, Guidelines for Media Sanitization
- SP 800-92, Guide to Computer Security Log Management
- SP 800-100, Information Security Handbook: A Guide for Managers
- SP 800-113, Guide to SSL VPNs
- SP 800-114 Rev. 1, User's Guide to Telework and Bring Your Own Device (BYOD) Security
- SP 800-121 Rev. 2, Guide to Bluetooth Security
- SP 800-128, Guide for Security-Focused Configuration Management of Information Systems
- SP 800-137, Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations
- SP 800-147, BIOS Protection Guidelines
- Draft SP 800-155, BIOS Integrity Measurement Guidelines

부록 C – 약어

AES	Advanced Encryption Standard
API	Application Programming Interface
AUFS	Advanced Multi-Layered Unification Filesystem
BIOS	Basic Input/Output System
BYOD	Bring Your Own Device
cgroup	Control Group
CIS	Center for Internet Security
CMVP	Cryptographic Module Validation Program
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DevOps	Development and Operations
DNS	Domain Name System
FIPS	Federal Information Processing Standards
FIRST	Forum for Incident Response and Security Teams
FISMA	Federal Information Security Modernization Act
FOIA	Freedom of Information Act
GB	Gigabyte
I/O	Input/Output
IP	Internet Protocol
IPS	Intrusion Prevention System
IT	Information Technology
ITL	Information Technology Laboratory
LXC	Linux Container
MAC	Mandatory Access Control
NIST	National Institute of Standards and Technology
NTFS	NT File System
OMB	Office of Management and Budget

OS	Operating System
PIV	Personal Identity Verification
RTM	Root of Trust for Measurement
SDN	Software-Defined Networking
seccomp	Secure Computing
SIEM	Security Information and Event Management
SP	Special Publication
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TPM	Trusted Platform Module
URI	Uniform Resource Identifier
US	United States
USCIS	United States Citizenship and Immigration Services
VM	Virtual Machine
VPN	Virtual Private Network
WAF	Web Application Firewall

부록 D – 용어

애플리케이션 가상화 (Application virtualization)	다수의 분리된 애플리케이션 인스턴스가 하나의 OS를 공유하는 가상화의 형태이다. 각 애플리케이션 인스턴스는 호스트의 다른 애플리케이션 인스턴스와 분리된다.
베이스 레이어 (Base layer)	다른 모든 컴포넌트가 추가된 이미지의 기본이 되는 레이어이다.
컨테이너 (Container)	애플리케이션 가상화 환경에서 애플리케이션을 패키징하고 안전하게 실행하는 방법이다. 애플리케이션 컨테이너 또는 서버 애플리케이션 컨테이너로도 알려져 있다.
컨테이너 런타임 (Container runtime)	각 컨테이너를 위한 환경이다. OS의 여러 컴포넌트(실행 중인 컨테이너의 리소스 및 리소스 사용을 분리)를 통합(coordinate)하는 바이너리 파일로 구성된다.
컨테이너 전용 운영 체제 (Container-specific operating system)	명확하게 컨테이너만 실행하도록 설계된 최소화된 호스트 OS이다.
파일시스템 가상화 (Filesystem virtualization)	다수의 컨테이너가 동일한 물리적 스토리지를 공유하지만, 다른 컨테이너의 스토리지에 접근하거나 변경할 수 없는 가상화의 한 형태이다.
범용 운영 체제 (General-purpose operating system)	컨테이너의 애플리케이션를 포함하여 많은 종류의 애플리케이션에서 사용할 수 있는 호스트 OS이다.
호스트 운영 체제 (Host operating system)	애플리케이션 가상화 아키텍처에서 다수의 애플리케이션이 공유하는 OS 커널이다.
이미지 (Image)	컨테이너를 실행하는데 필요한 모든 파일이 포함된 패키지이다.
분리 (Isolation)	소프트웨어의 인스턴스를 여러 개로 분리하여, 각 인스턴스는 다른 인스턴스를 보지 못하고 다른 인스턴스에 영향을 미칠 수 없도록 하는 기능이다.

마이크로서비스 (Microservice)	애플리케이션을 구성하기 위해 함께 작업하는 컨테이너의 집합이다.
네임스페이스 분리 (Namespace isolation)	컨테이너가 상호 작용할 수 있는 리소스를 제한하는 분리의 형태이다.
운영 체제 가상화 (Operating system virtualization)	특정 OS 用 애플리케이션을 실행하는데 사용할 수 있도록 OS 인터페이스를 가상으로 구현한 것이다.
오케스트레이터 (Orchestrator)	DevOps 직원(또는 자동화)이 레지스트리에서 이미지를 가져오고, 컨테이너에 이미지를 배치하고, 실행중인 컨테이너를 관리할 수 있게 하는 도구이다. 오케스트레이터는 또한 전체 호스트에서 컨테이너의 리소스 소비, 작업 실행, 기기 상태를 모니터링해야 한다.
오버레이 네트워크 (Overlay network)	대부분의 오케스트레이터에 포함된 SDN(Software-Defined Network) 컴포넌트로 동일한 물리적인 네트워크를 사용하는 애플레케이션 사이의 통신을 분리하기 위해 사용할 수 있다.
레지스트리 (Registry)	개발자들이 생성한 이미지를 쉽게 저장하고, 검색과 재사용을 목적으로 식별/버전 통제를 위해 태그/카탈로그를 생성하고, 타인이 생성한 이미지를 검색/다운로드할 수 있는 서비스이다.
리소스 할당 (Resource allocation)	특정 컨테이너가 사용할 수 있는 호스트의 리소스 양을 제한하는 메커니즘이다.
가상 머신 (Virtual machine)	가상으로 생성한 시뮬레이션 환경이다.
가상화 (Virtualization)	다른 소프트웨어가 실행되는 소프트웨어/하드웨어의 시뮬레이션이다.

부록 E - 참고 문헌

[1] NIST Special Publication (SP) 800-125, Guide to Security for Full Virtualization Technologies, National Institute of Standards and Technology, Gaithersburg, Maryland, January 2011, 35pp. https://doi.org/10.6028/NIST.SP.800-125.

[2] Docker, https://www.docker.com/

[3] rkt, https://coreos.com/rkt/

[4] CoreOS Container Linux, https://coreos.com/os/docs/latest

[5] Project Atomic, http://www.projectatomic.io

[6] Google Container-Optimized OS, https://cloud.google.com/container-optimizedos/docs/

[7] Open Container Initiative Daemon (OCID), https://github.com/kubernetesincubator/cri-o

[8] Jenkins, https://jenkins.io

[9] TeamCity, https://www.jetbrains.com/teamcity/

[10] Amazon EC2 Container Registry (ECR), https://aws.amazon.com/ecr/

[11] Docker Hub, https://hub.docker.com/

[12] Docker Trusted Registry, https://hub.docker.com/r/docker/dtr/

[13] Quay Container Registry, https://quay.io

[14] Kubernetes, https://kubernetes.io/

[15] Apache Mesos, http://mesos.apache.org/

[16] Docker Swarm, https://github.com/docker/swarm

[17] NIST Special Publication (SP) 800-154, Guide to Data-Centric System Threat Modeling (Draft), National Institute of Standards and Technology, Gaithersburg, Maryland, March 2016, 25pp. https://csrc.nist.gov/publications/detail/sp/800-154/draft.

[18] Common Vulnerability Scoring System v3.0: Specification Document, Forum for Incident Response and Security Teams (FIRST). https://www.first.org/cvss/specification-document

[19] NIST Special Publication (SP) 800-111, Guide to Storage Encryption Technologies for End User Devices, National Institute of Standards and Technology, Gaithersburg, Maryland, November 2007, 40pp. https://doi.org/10.6028/NIST.SP.800-111.

[20] CIS Docker Benchmark, Center for Internet Security (CIS). https://www.cisecurity.org/benchmark/docker/.

[21] Security Enhanced Linux (SELinux), https://selinuxproject.org/page/Main_Page

[22] AppArmor, http://wiki.apparmor.net/index.php/Main_Page

[23] NIST Special Publication (SP) 800-123, Guide to General Server Security, National Institute of Standards and Technology, Gaithersburg, Maryland, July 2008, 53pp. https://doi.org/10.6028/NIST.SP.800-123

[24] NIST Special Publication (SP) 800-164, Guidelines on Hardware-Rooted Security in Mobile Devices (Draft), National Institute of Standards and Technology, Gaithersburg, Maryland, October 2012, 33pp. https://csrc.nist.gov/publications/detail/sp/800-164/draft.

[25] NIST Special Publication (SP) 800-147, BIOS Protection Guidelines, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2011, 26pp. https://doi.org/10.6028/NIST.SP.800-147.

[26] NIST Special Publication (SP) 800-155, BIOS Integrity Measurement Guidelines(Draft), National Institute of Standards and Technology, Gaithersburg, Maryland, December 2011, 47pp. https://csrc.nist.gov/publications/detail/sp/800-155/draft.

[27] NIST Internal Report (IR) 7904, Trusted Geolocation in the Cloud: Proof of Concept Implementation, National Institute of Standards and Technology, Gaithersburg, Maryland, December 2015, 59 pp. https://doi.org/10.6028/NIST.IR.7904.

[28] NIST Special Publication (SP) 800-61 Revision 2, Computer Security Incident Handling Guide, National Institute of Standards and Technology, Gaithersburg, Maryland, August 2012, 79 pp. https://doi.org/10.6028/NIST.SP.800-61r2.

[29] NIST Special Publication (SP) 800-53 Revision 4, Security and Privacy Controls for Federal Information Systems and Organizations, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2013 (including updates as of January 15, 2014), 460pp. https://doi.org/10.6028/NIST.SP.800-53r4.

[30] Framework for Improving Critical Infrastructure Cybersecurity Version 1.0, National Institute of Standards and Technology, Gaithersburg, Maryland, February 12, 2014. https://www.nist.gov/document-3766.

[31] http://www.itworld.co.kr/news/101255