컨테이너 간 네트워크 환경에서의 보안 취약점 분석

남재현,○ 이승수
AccuKnox, 인천대학교
jn@accuknox.com, seungsoo@inu.ac.kr

Analysis of Security Vulnerabilities in Inter-Container Networking

Jaehyun Nam,○ Seungsoo Lee
AccuKnox, Incheon National University

요 약

현재, 클라우드 환경에서 다양한 애플리케이션 배포를 위한 컨테이너화(containerization) 기술의 사용이 빠르게 증가하고 있다. 하지만, 배포된 마이크로서비스(microservice) 내 컨테이너들 간의 네트워킹이 필수적임에도 불구하고 컨테이너 자체에 대한 보안 연구들이 주를 이루고 있을 뿐, 컨테이너 네트워킹의 견고성(robustness) 에 대해서는 심도 깊은 연구가 이루어지지 않고 있다. 따라서 본 논문에서는 보안 관점에서 현 컨테이너 네트워크의 구조적 문제점들을 분석하고, 실제 컨테이너 네트워크 환경의 취약점을 악용한 네트워크 공격 사례를 보임으로 컨테이너 네트워크에 대한 보안 연구의 필요성을 강조하고자 한다.

1. 서 론

최근, 가상화 기술의 주된 트렌드 중 하나는 사설(private) 및 공용 (public) 클라우드 환경 전반에 걸친 대규모 컨테이너 애플리케이션의 배포이다. 실제로 Google은 매주 20억 개 이상의 컨테이너를 생성하고 있으며[1], Yelp 또한 하루에 백만 개 이상의 컨테이너를 생성하고 있다[2]. 특히, Netflix의 경우 Titus 컨테이너 관리 플랫폼을 자체 개발, Amazon EC2 내에서 매주 300만 개 이상의 컨테이너를 생성하고 관리하고 있다[3].

하지만, 컨테이너 애플리케이션의 대규모 인스턴스화의 이면에는 컨테이너 내 작은 보안 균열 때문에 컨테이너들뿐만 아니라 컨테이너 전체 시스템 역시 심각한 영향을 받을 수 있다는 문제가 있다. Tripwire의 컨테이너 보안 보고서에 따르면 컨테이너를 사용하고 있는 기관 중 60%가 2018년에 이미 다수의 보안 사고를 경험한 것으로 나타났다[4]. 또한 해당 보고서에 따르면 컨테이너를 사용하는 기관 중 47%가 컨테이너 내 취약점이 존재한다는 점을 알면서도 사용하고 있으며, 또 다른 46%는 서비스 컨테이너 내 취약점이 존재하는지 조차 인지하지 못하고 있다고 밝혔다. 뿐만 아니라, 최근 암호 화폐 채굴을 위한 컨테이너 하이재킹(hijacking)과 같이 사용자 데이터가 목적이 아닌 해당 컴퓨팅 리소스 사용 자체를 목적으로 하는 공격들도 등장하기 시작하였다. 즉, 인터넷 서비스를 제공하는 컨테이너 대부분이 외부로 부터의 공격 대상이 되고 있는 것이 현실이다.

또한, 가상 머신들과 달리, 컨테이너들의 경우 호스트 OS의 커널 리소스(resource)를 공유하는 모델을 가지고 있다. 하지만, 이는 단일 컨테이너가 감염되었을 때 호스트 OS가 해당 컨테이너를 격리시킬 수 있는 능력과 관련하여 중요한 보안 문제를 야기한다. 따라서 이러한 보안 문제를 해결하기 위한 실제로 많은 기법들이 제안되어 왔으며, 예를 들어, AppArmor, SELinux 와 같은 리눅스 보안 모듈을 통해 다양한 시스템 리소스 남용을 방지하고 강력한 컨테이너 격리를 위한 기법들이 제안 되었다.

그러나, 이처럼 컨테이너 자체에 대한 다양한 접근 방식이 계속해서 등장하고 있음에도 불구하고, 컨테이너 네트워크에 대한 보안은 기존 리눅스 시스템에서 널리 사용되던 네트워크 접근 제어 기법 (예를 들어, iptables) 외 추가적인 연구가 많이 이루지지 않고 있는 것이 현실이다. 결국, 각 컨테이너 자체의 네트워크 상호 작용에 대해서 제한 외 컨테이너 네트워크 자체에 대한 보안에는 구조적 허점을 가지고 있다.

따라서, 본 논문에서는 현 컨테이너 네트워크 환경에서의 구조적 보안 문제점들에 대해서 분석하고자 한다. 특히, 현재 널리 사용되고 있는 컨테이너 시스템들에 대해한 5 가지의 보안 문제점을 제시하고, 이러한 문제점들 악용하여 컨테이너 간 가능한 네트워크 하이재킹(hijacking) 공격을 예시로 보여준다. 마지막으로, 본 논문을 통해 현 컨테이너 보안의 실상을 이해하고 컨테이너 네트워크 에 대한 추가적인 보안 연구가 필요함을 보이고자 한다.

2. 컨테이너 네트워크 구조 분석

이 장에서는 현재 가장 널리 사용되는 두 가지 컨테이너 관리 시스템인 도커[5]와 쿠버네티스[6]를 소개하고, 현 컨테이너 네트워크가 어떻게 구성되어 있는지에 대해 간략한 소개하고자 한다.

(1) 도커(Docker) 플랫폼: 도커는 컨테이너를 배포하고 실행하기 위한 가장 대표적인 플랫폼이다. 도커에서 기본적으로 브리지 네트워크(bridge network)를 기본 컨테이너 네트워크로 사용한다. 그리고 각각의 도커 컨테이너는 기본적으로 Docker0 라는 브리지에 연결된다. 그러나 docker-compose와 같이 여러 컨테이너를 마이크로서비스 형태로 생성할 경우에는 해당 컨테이너들에 대한 네트워크 트래픽을 관리하기 위해 별도의 브리지(bridge)

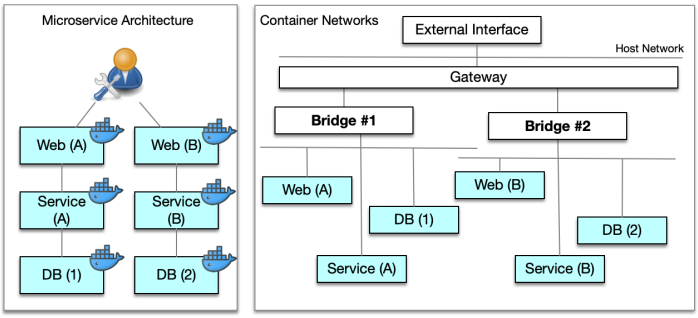

그림 1 마이크로 서비스 및 컨테이너 네트워크

가 자동으로 생성되고, 컨테이너들은 해당 브리지에 연결된다. 예를 들어, 그림 1은 두 개의 마이크로서비스 아키텍처를 보여준다. 일반적으로, 네트워크 서비스를 표현할 때는 그림 1 왼쪽처럼 표현한다. 하지만, 실제 네트워크와 컨테이너들의 구성은 그림 1 오른쪽처럼 각각의 브리지 네트워크에 버스 형태로 연결된 모습을 보인다.

도커의 경우에는 네트워크 흐름 제어를 제공하기 위해 iptables를 사용하여 브리지 네트워크에 네트워크 및 보안 정책을 적용한다. 특히 도커는 iptables에서 네트워킹을 위한 Docker와 보안을 위한 Docker-Isolation의 두 가지 체인을 유지 및 관리한다. 서비스가 외부 네트워크에 노출되면 NAT(Network Address Translation) 테이블의 Docker 체인에 전달 정책이 추가되며 Docker-Isolation 체인에 보안 정책을 추가하여 브리지 간의 통신을 차단한다.

(2) 쿠버네티스(Kubernetes) 오케스트레이션 시스템: 쿠버네티스는 여러 노드 (예를 들어, 물리적 서버) 의 다수의 컨테이너를 통합 관리할 수 있도록 하는 오픈소스 컨테이너 오케스트레이션 시스템이다. 즉, Docker의 경우 동일한 호스트 내 컨테이너들을 관리를 하였다면, 쿠버네티스는 여러 노드에 걸쳐 대규모 컨테이너 환경을 구성할 수 있다는 것을 의미한다.

쿠버네티스는 도커처럼 자체적인 네트워킹 환경을 제공하기 보다는, 제3의 오버레이 네트워크(예를 들어, Flannel, Weave, Calico)를 사용하여 여러 노드의 컨테이너 간 연결을 제공함으로써, 다양한 네트워킹 구성이 가능하도록 하고 있다. 즉, 어떠한 오버레이 네트워크를 사용하는가에 따라서도 컨테이너 네트워크 보안성이 크게 좌지우지할 수 있다는 것을 의미하기도 한다.

(3) 호스트 네트워크 컨테이너: 일반적인 컨테이너와 달리, 호스트 IP 주소를 통해 외부로 서비스를 제공하는 컨테이너를 호스트 네트워크 컨테이너라고 한다. 예를 들어 HAProxy, OpenVPN, MemSQL 등이 있다. 이러한 컨테이너들의 경우 개별적인 컨테이너 네트워크 네임스페이스가 아닌 호스트 네트워크 네임스페이스를 공유, 호스트 네트워크 인터페이스를 통해 직접적인 컨테이너 서비스를 외부에 노출시킬 수 있다. 그리고 이를 통해, 일반적인 컨테이너에 비해 성능적인 이점을 얻게 된다.

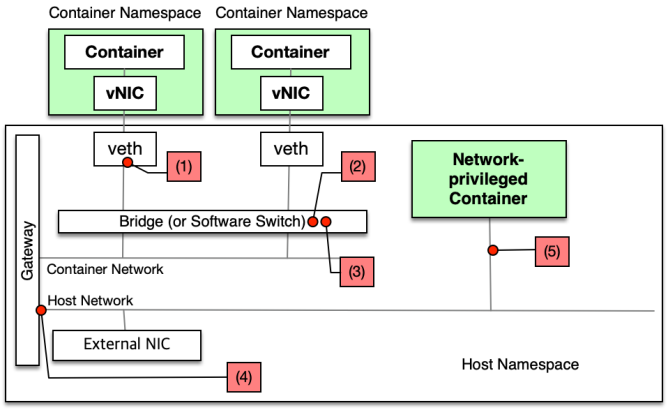

그림 2 현 컨테이너 네트워크 보안 문제점

3. 컨테이너 네트워크 문제점

현 컨테이너 플랫폼들은 IP 기반의 접근 제어 기법을 대부분 채택하여 컨테이너 네트워크 보안 정책을 시행하고 있지만, 이는 오늘날의 컨테이너 토폴로지의 통신 권한을 제한하는데 많은 허점을 야기하고 있다. 다음은 현 컨테이너 네트워크의 구조적인 특성으로 발생하는 5가지 보안 문제점에 관한 분석이다.

(1) 패킷 기반 출발지 검증의 한계: 그림 2와 같이 각 컨테이너에는 가상 인터페이스(vNIC)가 있지만 해당 인터페이스는 컨테이너의 네트워크 네임스페이스에서만 접근이 가능하다. 따라서 컨테이너 플랫폼은 호스트에서 이에 해당하는 트윈(twin) 가상 인터페이스(veth)를 생성하고, 이 가상 인터페이스를 브리지와 연결하여 다른 컨테이너들과 통신할 수 있다.

그러나 이러한 구조에서의 보안 문제점은 컨테이너에서 생성한 각 패킷이 호스트 네트워킹 네임스페이스로 전환되는 순간 모든 컨테이너의 패킷들이 출발지 컨테이너와 무관하게 처리된다는 점이다. 즉, 추가적인 보안 검사나 패킷 전달에 대한 모든 결정은 패킷 헤더의 정보를 기반으로 내려지게 되며, 악의적인 컨테이너는 이 점을 악용하여 다른 컨테이너의 신원으로 패킷을 직접 위조, 다른 컨테이너에 트래픽 포이즈닝(traffic poisoning) 공격을 수행할 수 있게 된다.

(2) IP 기반 액세스 제어의 한계: 현 컨테이너 플랫폼들은 컨테이너 간 네트워크 접근 제어를 위해 IP 기반 접근 제어 기법을 사용한다. 하지만, 컨테이너의 IP 주소는 동적으로 할당되기 때문에, 컨테이너가 생성되고 종료할 때마다 조정이 필요하다. 또한, 이러한 문제점을 보완하기 위해, 네트워크 운영자가 IP 주소 대신 컨테이너에 대한 고수준 레이블로 다양한 보안 정책을 시행하더라도, 내부적으로는 이러한 레이블 역시 IP 주소로 변환되어 적용되기 때문에, 여전히 문제는 발생한다. 뿐만 아니라, Layer 2 공격에 대해서는 여전히 취약하다.

(3) 애플리케이션 프로토콜에 대한 보안 검증의 부재: IP 기반 접근 제어는 컨테이너 간의 악성 네트워크 연결 시도를 제한할 수 있지만, 정상적인 컨테이너 간의 악성 콘텐츠 전송에 관한 제어는 할 수 없어 심각한 보안문제를 야기할 수 있다. 또한 다양한 미들박스 또는 가상 네

트워크 기능을 통해 다양한 보안 기능을 제공할 수 있던 레거시(legacy) 네트워크와 달리, 컨테이너는 동일한 호스트 내에서 OS 수준 네트워킹 기능을 사용하여 논리적으로 생성된 여러 네트워크에서 통신하기 때문에, 호스트 내 각각의 논리 네트워크에 대해 개별적인 보안 기능(심층 패킷 검사)을 적용하는 것에는 한계가 있다.

(4) 무분별한 호스트 접근: 그림 2와 같이 각 컨테이너 네트워크는 외부 액세스를 위한 게이트웨이 인터페이스가 있다. 하지만, 해당 인터페이스가 호스트 네트워크 네임스페이스에 속해 있기 때문에 악의 적인 컨테이너가 게이트웨이를 통해 호스트 내 동작 중인 서비스를 게이트웨이 주소를 통해 접근할 수 있다는 문제점이 존재한다. 더욱이, 쿠버네티스의 경우 여러 노드의 컨테이너들이 하나의 오버레이 네트워크로 묶여 있기 때문에, 다른 노드의 게이트웨이 주소를 통해 해당 노드의 호스트 서비스 역시 접근이 가능하다. 그리고 최악의 경우 악의적인 컨테이너가 호스트 서비스를 악용하여 호스트 자체의 권한을 탈취할 수 있다.

(5) 호스트 네트워크 컨테이너에 대한 제한 부재: 호스트 네트워크 컨테이너의 트래픽은 컨테이너 네트워크 자체를 통과하지 않기 때문에 성능 이점을 얻을 수 있는 반면, 이 때문에, 컨테이너 네트워크 보안 영역에서 배제된다. 즉, 기존 컨테이너 네트워크 접근 제어 기법들을 해당 컨테이너에 적용이 불가능하다. 또한, 호스트 네트워크 컨테이너들의 경우 호스트와 동일한 네트워크 네임스페이스를 공유하므로 호스트 네트워크 인터페이스에 접근할 수 있을 뿐만 아니라 호스트 내 모든 컨테이너에 대한 가상 인터페이스 역시 접근할 수 있기 때문에, 손쉽게 컨테이너들의 모든 네트워크 트래픽을 모니터링 할 수 있으며, 악성 패킷 역시 해당 인터페이스들을 통해 주입할 수 있다.

4. 컨테이너 네트워크 하이재킹 공격

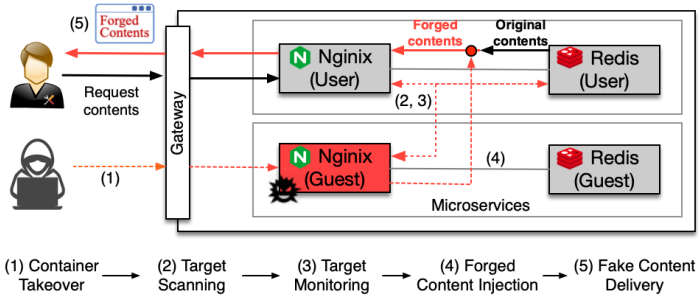

그림 3 컨테이너 네트워크 하이재킹 공격 시나리오

그림 3은 쿠버네티스 환경 내 배포된 두 개의 독립적인 마이크로서비스를 보여주고 있다. 하나는 정상적인 사용자를 위한 서비스이고, 다른 하나는 게스트 사용자를 위한 서비스이다.

본 공격 시나리오는 다음과 같다. (1) 공격자는 웹 애플리케이션 취약점을 악용하여 공개 Nginx 서버에 침투한 후 공격 바이너리를 다운로드 한다. 그리고 공격자는 3개의 네트워크 기반 공격을 활용하여 Nginx-Guest 컨테이너를 손상시키고 메시지 가로채기(man-in-the-middle) 공격을 실행한다. (2) 먼저, ARP 기반 스캐닝을 통해 네트워크 주변의 활성 컨테이너들을 찾는다. 쿠버네티스 환경에서 모든 컨테이너가 오버레이 네트워크에 연결되어 있기 때문에, ARP 패킷을 통해 컨테이너들 및 각각에 대한 네트워크 정보를 쉽게 수집할 수 있다. (3) 다음으로, 공격자는 가짜 ARP 응답을 네트워크에 주입 (ARP spoofing attack) 하여 Nginx-User와 Redis-User 컨테이너 간의 모든 트래픽이 Nginx-Guest를 통과하도록 한다. (4) 마지막으로 공격자는 Redis-User에서 전달된 패킷을 내부적으로 드롭하고 위조된 패킷을 주입하여 정상적인 사용자에 대한 응답을 위조된 내용으로 대체한다. (5) 그러면 Nginx-User는 원본이 아닌 위조된 콘텐츠를 사용자에게 반환한다. 결국 사용자는 공격자가 의도한 대로 위조된 콘텐츠를 받게 된다.

5. 결론 및 향후 연구

현재, 컨테이너 기술은 대규모 엔터프라이즈 및 클라우드 환경에서 다양한 서비스를 배포하는데 널리 사용되는 가상화 기술로 부상했다. 하지만, 컨테이너 보안에 대해서는 여러 연구를 통해 컨테이너 자체 보안은 많이 향상되고 있으나, 네트워크 보안에 대해서는 미흡한 것이 현실이다. 따라서 본 논문에서는 현재 컨테이너 네트워크와 관련된 보안 문제를 분석하고, 실제 공격 시나리오를 통해 실질적인 문제임을 보이고자 하였다. 특히, 대부분의 컨테이너 네트워크 문제점들이 기존에 널리 알려진 네트워크 문제라는 이유로 간과되고 있는 것이 현실이며, 이 때문에 지속적인 보안 사고가 발생하고 있는 가운데, 컨테이너 네트워크 보안 연구의 필요성을 다시금 강조하고자 하였다.

그리고 본 연구를 기반으로, 컨테이너 네트워크 통신을 보호하기 위한 지능형 통신 브리지를 제시하여 현 컨테이너 환경에서의 네트워크 보안 문제를 해결하는 것이 앞으로 연구할 과제이다.

참 고 문 헌

[1] Google, “Everything at Google runs in containers”, https://cloud.google.com/containers.
[2] Yelp, “How Yelp Runs Millions of Tests EveryDay”, https://engineeringblog.yelp.com/2017/04/how-yelp-runs-millions-of-tests-every-day.html.
[3] A. Joshi, A. Leung, C. Dwyer, F. Kung, S. Dhillon, T. Bak, A. Spyker, and T. Bozarth, “Titus, the Netflix container management platform is now open source”, Netflix Technology Blog, 2018.
[4] Tripwire, “State of Container Security Report”, https://www.tripwire.com/state-of-security/devops/organizations-container-security-incident.
[5] Docker, https://www.docker.com.
[6] Kubernetes, https://kubernetes.io.