클라우드 네이티브 분산 처리 환경에서 eBPF 사이드카를 활용한 데이터 흐름 가시성 부여

김영호, 김종원*
광주과학기술원
kyoungho2018@gist.ac.kr, *jongwon@gist.ac.kr

Data Flow Visibility of Distributed Processing in Cloud-Native Environment using eBPF Sidecar

YoungHo Kim, Jong Won Kim*
Gwangju Institute of Science & Technology

요 약

본 논문은 Driver-Executor 구조의 분산 처리 환경에서 Task 단위 데이터 흐름에 대한 가시성을 확보하는 방법을 제안한다. 이를 위해 Executor Pod에 Sidecar 형태로 eBPF 기반 트래픽 수집기를 배치하고, Driver가 제공하는 Task 정보를 참조하여 수집된 패킷을 식별한다. 각 Pod 내부에서 eBPF 프로그램을 격리 실행함으로써 인터페이스당 단일 프로그램 제한과 충돌 문제를 완화하며, 수집된 로그는 중앙 관제 요소로 전송되어 사후 분석에 활용된다. 실험에서 캡처된 모든 이벤트가 저장된 객체 정보와 누락 없이 일치함을 확인하여, 제안 방식이 Task 단위 데이터 전송 흐름에 대한 가시성을 제공함을 확인하였다.

Ⅰ. 서 론

AI 시대의 도래와 함께 데이터의 가치가 급격히 증가하고 있다. 특히 대규모 데이터셋을 활용한 학습 및 분석이 필수 요소가 되면서, 데이터 프라이버시와 데이터 보안에 대한 중요성도 전례 없이 부각되었다. 이에 따라 데이터가 적절하게 사용되는지, 허가된 요소에서만 활용되는지를 관측하고 검증할 필요성이 대두되고 있다.

기존 네트워크 모니터링 도구(tcpdump, Wireshark)는 송수신 주체를 식별할 수 있지만, 애플리케이션 수준의 데이터 페이로드를 추적하기에는 한계가 있다. 분산 추적 도구(OpenTelemetry, Jaeger 등)는 요청 경로를 세밀하게 추적할 수 있으나, 서비스 코드에 직접 API를 삽입해야 하며 페이로드 추적은 기본적으로 지원되지 않는다[1]. 이로 인해 실제 데이터가 어떤 작업에 의해 어디로 전송되었는지를 식별하는 데는 충분하지 않다.

이러한 문제에 대해, 본 논문은 업무를 지시하는 Driver와 실제로 업무를 처리하는 Executor로 구성된 분산 처리기의 데이터 사용 이력과 네트워크 트래픽을 연결하고자 한다. 이를 통해 데이터 흐름에 대한 가시성(Visibility)을 부여하여, 추적성을 얻기 위한 단초를 마련하고자 한다.

Ⅱ. 아키텍처 설계

본 논문은 쿠버네티스 클러스터 상에 배치된 Driver-Executor 구조의 분산 처리기에 대해, eBPF 기반 트래픽 로그 수집기를 Executor의 Sidecar로 붙이고, Executor로부터 작업 정보를 받아 로그에 추가한 후, 중앙수집기를 통해 관제 요소(이하 '초소')에 전달하여 특정 데이터의 이동 경로와 중간지를 추정할 수 있도록 하고자 한다.

이러한 목적을 위해, 본 논문은 [그림 1]과 같은 환경을 구성하여 취약점을 악용한 직접 침투나 가로채기, 도청, 위장 및 위조 등의 공격으로 인한 데이터 유출 상황을 최대한 방지하고자 한다. 이는 Executor 외의 요소로 인하여 데이터가 유출되는 상황을 최소화하기 위함이다.

민감도 및 기밀성을 기준으로 각 데이터에 C(Classified)/S(Sensitive)/O(Open) 보안등급을 부여하고, 각 Executor에 접근 보안등급을 부여하는 다중 계층 보안을 적용한다. 이를 통해 Executor가 높은 등급의 데이터에 접근하거나, 낮은 등급의 스토리지에 저장하는 것을 방지한다. 또한 가로채기 및 도청을 방지하고자 Pod 간 통신 암호화를 적용하고, 구성원으로 위장하는 것을 방지하기 위해 상호 인증을 적용한다. 이는 공격자가 위장하거나 통신을 도청하여 데이터를 획득할 가능성을 차단하기 위함이다.

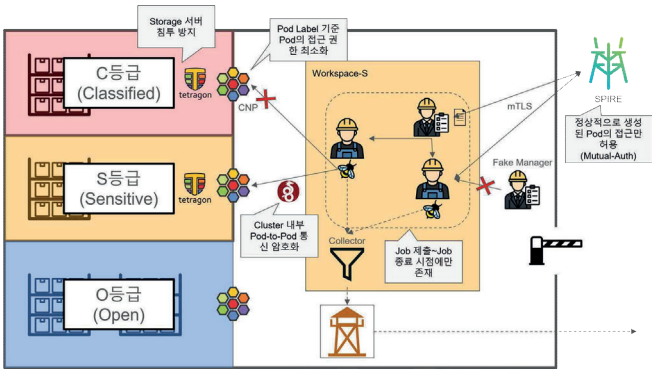

그림 1. 쿠버네티스 클러스터 내 보안 강화를 위한 구성요소 배치 및 주요 보안 기능

이는 Cilium CNI의 'CiliumNetworkPolicy' CRD를 이용하여 Pod Label 및 Namespace 기준 정책 적용을 통해 구현한다. 또한, WireGuard 기반 Transparent Encryption과 SPIFFE/SPIRE 기반 Mutual Authentication으로 통신 암호화 및 상호 인증을 적용한다[2].

마지막으로 내부 침투를 방어하기 위해 각 구성원의 행동을 감시 및 통제할 수 있어야 한다. 이를 위해 eBPF 기반 런타임 보안 도구인 Tetragon을 적용한다. 이는 커널 공간에서 시스템 콜 사용을 제한하거나 Container Escape를 탐지 후 차단하는 등 컨테이너 보안을 강화하는 데에 활용할 수 있다[3]. 이를 활용하여 스토리지 및 분산 처리기에 직접 침투하여 데이터를 유출하는 행위를 방어할 수 있다.

분산 처리기는 다음과 같은 전제조건을 따른다. 이는 상황 시나리오를 단순화하고, 데이터 흐름 추적 가능성 검증에 집중하기 위함이다.

1. 데이터를 저장하는 스토리지는 객체(Object) 스토리지로 한정한다. 이는 REST API로 데이터에 접근하는 특성을 활용하기 위함이다.

2. 데이터 흐름은 Source → Executor → Destination으로 한정한다. 여기서 Source와 Destination은 스토리지이며, Executor는 Source에서 데이터를 가져와 처리 결과를 Destination에 저장한다.

3. Executor는 데이터 처리 과정에서 외부와 통신하지 않는다. 즉, 데이터 수신 이후의 통신은 Destination으로의 데이터 전송으로 취급한다.

4. Source는 항상 Executor와 같은 클러스터에 존재하지만, Destination은 클러스터 외부에 존재할 수 있다고 가정한다. 이에 따라 Destination을 향한 전송을 직접적으로 제어하지 못할 수 있으며, 따라서 사전 통제 실패와 부적절한 Destination 지정으로 데이터 유출이 발생할 수 있다.

Ⅲ. 아키텍처 구현

위에서 제시한 실행 환경과 전제조건을 바탕으로, 데이터 흐름 추적 및 유출 가능성을 식별하는 분산 처리기의 구조를 제시한다.

[그림 2]는 Driver-Executor 구조의 분산 처리기의 구조와 상호작용을 의미한다. Driver는 제출된 작업(이하 Job)을 데이터 범위를 기준으로 여러 소작업(이하 Task)으로 분할하고, 각 Task를 Executor에 할당한다.

Executor는 할당된 Task를 처리하는 주체로, 데이터를 직접 다루고, Source와 Destination과 직접 통신한다. 그렇기에 실질적인 데이터 유출은 Executor에서 발생하며, 이의 Outbound 트래픽을 관측하면 데이터 유출 시 원인 후보군을 파악할 수 있다.

이러한 목적을 위해 Executor를 2개의 구성요소로 나눈다. 하나는 실제 작업을 처리하는 Runner이며, 나머지는 eBPF 프로그램으로 트래픽을 수집하는 Watcher이다. Watcher는 Runner의 Sidecar로 존재하며, 생성 직후 Pod 내부 공유 네트워크 인터페이스에 TCX/Egress 프로그램을 부착한다. 데이터는 Ringbuf Map으로 커널 영역에서 유저 영역으로 전달되며, Watcher는 이를 Task 정보와 결합한 뒤 Collector로 전달한다.

Watcher를 Sidecar로 부착하는 이유는 다른 eBPF 프로그램 간 충돌 문제를 방지하면서 운영 자유도를 확보할 수 있기 때문이다. 일반적인 도구들은 [그림 2] 기준 'lxc' 인터페이스에 프로그램을 부착하는데, TC 프로그램은 'TC_ACT_OK' 등 처리 종료 값을 반환하면 후순위 프로그램이 무시되며, 2개 이상이 즉시 처리를 종료하면 반드시 일부는 동작하지 않는다는 문제가 있다. 하지만 Pod 내 공유 인터페이스에 부착하는 도구는 확인되지 않았으며, Sidecar를 경유하면 eBPF 프로그램 목록을 직접 관리할 수 있기에 그러한 문제는 발생하기 어렵다. 다만 eBPF 프로그램을 Pod 내에 부착하고, RingBuf Map을 사용하려면 Watcher에 CAP_NET_ADMIN, CAP_BPF, CAP_PERFMON 권한을 부여해야 한다.

UDS(Unix Domain Socket)는 Task 정보 교환 및 로그 전송 경로로 쓰인다. 이는 eBPF 프로그램을 우회하여 무가치한 로그 수집으로 인한 오염 및 혼란을 제거하기 위함이다. Collector는 Host에 UDS를 두고, Watcher가 이를 HostPath Volume으로 가져와 이를 통해 Collector에게 로그를 전달한다. 또한 Watcher는 UDS를 Pod 내 공유 EmptyDir Volume에 배치하며, Runner가 이를 통해 Task 정보를 전달한다.

Ⅳ. 실험 구성 및 결과

아키텍처의 유효성을 확인하고자 PostgreSQL을 초소로 사용하고, 동일 클러스터에 MinIO 1개를 배포하여 Source로, AWS EC2 2개에 각각

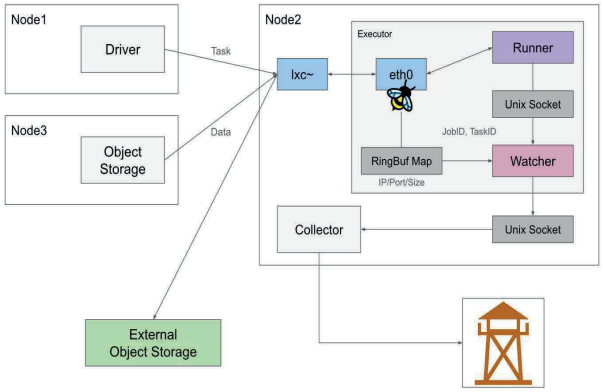

그림 2. 분산처리기의 데이터 흐름 가시성 확보를 위한 Executor 내부 구조 및 eBPF 기반 트래픽 로그 수집 방식

MinIO를 배포하여 Destination으로 활용한다.

Executor는 Source에서 csv 데이터를 읽고, 이를 gzip으로 압축해 임의의 Destination(MinIO)에 저장하였다. 저장 시 파일 이름에는 Task 단위의 작업 식별자와 데이터 범위를 포함하여, 트래픽 로그와의 직접 비교가 가능하게 하였다. 파일 이름과 MinIO의 IP를 수집된 트래픽과 상호 비교하여, 각 Task에서 실제로 전달한 대상과 트래픽으로 식별된 대상의 일치 여부를 파악할 수 있다. 이를 이용하여 Task 단위 데이터 흐름을 식별할 수 있음을 보이고자 한다.

PostgreSQL 기록과 MinIO 저장 객체를 상호 비교한 결과, 식별된 모든 전송에 대응하는 객체를 발견할 수 있었으며, 누락되거나 불일치한 경우는 발견되지 않았다.

Ⅴ. 결론

본 논문은 분산 처리기의 eBPF 기반 트래픽 추출 Sidecar를 도입하여 Task 정보와 트래픽 로그의 결합을 통해 데이터의 이동 경로 추적 가능성을 제시하고 이를 검증하였다.

하지만 트래픽을 단순히 Task에 명시된 데이터 범위와 연결하므로 사용되거나 저장된 데이터 범위는 더 좁을 수 있으며, 전체 데이터 흐름을 제시하지 않고 상호작용 대상만을 알려준다는 한계점이 있다.

ACKNOWLEDGMENT

본 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2019-0-01842, 인공지능대학원지원(광주과학기술원))

참 고 문 헌

[1] N. M. Popa and A. Oprescu, "A data-centric approach to distributed tracing," in Proc. 2019 IEEE Int. Conf. Cloud Comput. Technol. Sci., Sydney, Australia, Dec. 2019, pp. 209 - 216.

[2] A. Gupta, "Zero Trust Security with Cilium," Isovalent, 2024. [Online]. Available: https://isovalent.com/blog/post/zero-trust-security-with-cilium/.

[3] N. R. Ivánkó, "Detecting a Container Escape with Tetragon and eBPF," Isovalent, Jul. 2024. [Online]. Available: https://isovalent.com/blog/post/2021-11-container-escape/.