eBPF 기반 도커 컨테이너 이상행위 탐지 도구 구현에 관한 연구

(A Study on the Implementation of an eBPF-based Anomaly Detection Tool for Docker Containers)

이택규*, 박정수**
(Taek Kyu Lee, Jungsoo Park)

요 약

컨테이너에서 프로세스의 동작 모니터링과 네트워크를 통하여 들어오는 패킷을 분석하기 위하여 다양한 연구가 수행되고 있다. Agent 설치 방식, 별도의 instance 생성을 통한 분석 방식 등의 연구가 진행되었으나, 처리 속도가 지연된다는 점에서 단점이 있었다. 또한, 기존 논문들에서는 직접 구현한 system call 모니터링 도구를 이용하기 때문에 효율성이 떨어져 활용이 어려운 점이 있었다. 본 논문에서는 이를 극복하고자 eBPF를 통하여 처리 속도에 영향을 최소화하면서 프로세스의 동작 및 다양한 패킷을 모니터링하는 기법을 제안하였다. eBPF는 host 커널에 설치되어 처리하기 때문에 기존 방식보다 처리 속도 측면에서 우수하다. eBPF 사용을 위하여 Bcc 툴에서 제공되는 기존의 코드를 변경하여 TCP 패킷 전부를 추적하여, 패킷 분석을 수행할 수 있도록 하였다. 또한, eBPF 도구에서 제공되는 execsnoop 툴을 이용하여 컨테이너 system call 추적 도구를 구현하였다. 이를 활용하여, 새롭게 생성되는 프로세스에 대하여 PPID 기반 system call tracing을 수행하고, 새롭게 생성되어 동작하는 프로세스의 명령을 tracing하여 이상 동작을 수행하는 경우, 탐지하는 시스템을 구현하였다.

■ 중심어 : 버클릿패킷필터; 도커; 이상행위탐지

Abstract

To monitor process behavior within containers and analyze incoming network packets, various studies have been conducted. Research has explored methods such as agent-based installations and analysis through separate instances. However, these approaches have been limited by delays in processing speed. Furthermore, existing papers often rely on custom-implemented system call monitoring tools, which are inefficient and difficult to utilize. This paper proposes a method using eBPF to overcome these limitations, minimizing the impact on processing speed while monitoring both process behavior and diverse network packets. Because eBPF is installed and processed within the host kernel, it offers superior processing speed compared to conventional methods. To leverage eBPF, we modified existing code from the Bcc tool to trace all TCP packets for comprehensive packet analysis. We also developed a container system call tracing tool using the execsnoop tool provided by eBPF. This tool performs PPID-based system call tracing for newly created processes and monitors their commands. By doing so, we have implemented a system that can detect processes exhibiting anomalous behavior.

■ keywords : eBPF ; docker ; Anomaly detection

Ⅰ. 서 론

컨테이너 가상화 기술은 실행 환경을 서로 격리하여 독립적으로 운영할 수 있도록 지원하는 기술이다[1]. 이는 독립된 운영체제와 라이브러리, 전용 리소스 및 애플리케이션이 함께 구동되는 가상머신(VM)에 비해 훨씬 경량화되어 있으

* 정회원, 충남대학교 컴퓨터공학과
* 정회원, 강남대학교 컴퓨터공학부

접수일자 : 2025년 08월 31일
게재확정일 : 2025년 09월 17일
교신저자 : 박정수 e-mail : jspark@kangnam.ac.kr

며, 실제 환경과 유사한 성능을 제공함으로써 VM보다 우수한 성능과 효율성으로 인해 널리 활용되고 있다. 이러한 컨테이너 가상화 기술은 리눅스 커널에서 제공하는 cgroup과 namespace 등의 기술이 사용된다[2,3]. 또한 호스트 커널은 컨테이너 간에 공유되며, 모든 컨테이너에서 공유 커널의 direct-physical map을 통해 entire physical memory에 맵핑된다. 따라서 정보 유출은 동일한 컨테이너 및 호스트에 영향을 미칠 수 있고 이로 인해 보안사고 발생 시 컨테이너는 VM보다 더 큰 피해를 입게된다[4]. 또한, 최신 프로세서 아키텍처의 취약점을 악용한 공격이나 컨테이너 기반 서비스에서 발생하는 다양한 공격 역시 컨테이너 보안에 중대한 위협 요소로 작용하고 있다[5,6].

본 논문에서는 eBPF 기반의 실시간 모니터링 방식을 적용하였다. 이를 통해 정상 컨테이너의 네트워크 및 시스템 호출 정보를 수집하고, BCC 툴인 execsnoop을 이용하여 새롭게 생성된 프로세스를 효과적으로 추적하는 시스템 및 BPF를 통하여 실시간으로 들어오는 네트워크 트래픽을 trace 하는 것을 제안한다.

본 논문은 다음과 같이 구성되어 있다. 1장에서론 2장의 본 연구의 배경지식 및 관련연구를 설명한다. 3장에서 본 연구에서 제안된 시스템을 구현 및 결과를 도출하고, 마지막으로 4장에서 결론을 맺는다.

Ⅱ. 본 론

1. Docker

Docker는 리눅스 컨테이너 기반의 오픈 소스 가상화 도구이며 어플리케이션을 구축 및 테스트 배포를 할 수 있는 소프트웨어이다[7-9]. Docker는 소프트웨어를 컨테이너라는 표준화 유닛으로 패키징하며, 이 컨테이너에는 라이브러리, 시스템 도구, 코드, runtime등 소프트웨어를 실행하는 데 필요한 모든 것이 들어 있습니다. Docker를 사용하면 환경에 구애받지 않고 어플리케이션을 신속하게 만들수 있으므로 요즘 개발자들은 Docker를 많이 사용하는 추세이다. Docker는 컨테이너 방식으로 프로세스를 격리해서 실행하고 관리할 수 있도록 도와주며, 계층화된 파일 시스템에 기반해 효율적으로 이미지를 구축할 수 있도록 해줍니다. Docker를 사용하면 이 이미지를 기반으로 컨테이너를 실행할 수 있으며, 다시 특정 컨테이너의 상태를 변경해 이미지로 만들 수 있다. 이렇게 만들어진 이미지는 파일로 보관하거나 원격 저장소를 쉽게 공유할 수 있으며, Docker만 설치되어 있다면 Docker는 모든 컨테이너에게 동일한 라이프 사이클을 관리 할 수 있는 Tool과 플랫폼을 제공한다.

2. eBPF

eBPF(extended Berkeley Packet Filter)는 기존 네트워크 패킷 필터링 프로그램(Classic BPF, cBPF)의 확장된 버전이다[10]. 시스템 모니터링을 하기 위해서는 모니터링을 위한 코드 추가 등의 kernel 수정이 필요한데 리눅스 커널을 수정하는 것은 어떠한 방면에서도 효율적인 선택이 될 수 없기 때문에, 최근에는 커널의 소스 코드 수정 없이 모니터링이 가능한 eBPF를 사용하고 있는 추세이다. 개발자는 필요한 소스 코드를 C 언어로 작성할 수 있고, LLVM/Clang과 같은 컴파일러를 사용하여 eBPF bytecode로 컴파일 할 수 있다. 컴파일된 bytecode는 Verifier의 검사를 거치게 되며, 해당 bytecode에 문제가 없다고 판단되면 eBPF Tool을 통해 kernel의 eBPF로 적재되어 실행된다. eBPF를 사용하면 sandbox 형태의 프로그램을 커널 내부에서 실행시킬 수 있으며 eBPF를 사용하여 시스템 모니터링을 진행하는 것은 발생할 수 있는 시스템 성능 저하를 상대적으로 낮출 수 있다. 이러한 장점 때문에 eBPF는 꾸준히 발전하였고, 현재는 Linux 자체적으로도 kernel에 포함하여 제공하고 있다[11].

[그림 1]은 eBPF Program의 컴파일 과정과 bpf() System Call의 동작을 나타내고 있다.

LLVM/clang 은 Backend로 eBPF를 지원하고 개발자가 작성한 eBPF Source Code는 LLVM/clang을 통해서 eBPF Bytecode로 컴파일 된다. 그 후 eBPF Bytecode 는 tc 나 iproute2 같은 eBPF 관리 App(Tool)을 이용해 Kernel의 eBPF에 적재되고 내부적으로 bpf() System call을 이용하여 eBPF Bytecode를 eBPF에 적재한다.

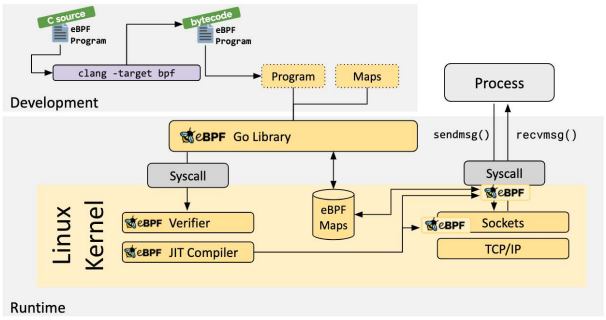

그림 1. eBPF Complie process

bpf()System Call은 eBPF Bytecode 적재 뿐만 아니라 App이 eBPF가 이용하는 Map에 접근할 수 있게 한다. 결구 App과 eBPF는 Map을 이용하여 다양한 통신을 할 수 있게 하고 둘 사이의 통신은 eBPF가 더욱 다양한 기능을 수행 할 수 있도록 한다.

3. cAdvisor

Kubernetes에서 사용하는 기본적인 모니터링 에이전트로, 모든 node에 설치되어 node에 대한 정보와 pod(컨테이너)에 대한 metrics(지표)를 수집한다. cAdvisor은 CPU, 메모리, 파일 시스템, network used등의 리소스 사용량과 성능 정보를 수집하며, 다른 데이터를 수집하고 싶으면 별도의 에이전트를 사용해야 한다[12]. cAdvisor는 장기간 사용하기 위한 metrics를 저장하진 않으므로 해당 기능을 사용하려면 전용 모니터링 도구를 찾아야 한다.

4. 관련연구

클라우드 환경에서 네트워크 침입에 대한 탐지 시스템에 관한 다양한 연구가 진행되고 있다. Chen Xu et al.의 경우, NIDS(Network Intrusion Detection System)를 agent로 구현하여 적용하는 방법을 제안하였다[13]. 기존에 NIDS를 별도의 가상 인스턴스로 구성하여 제안하는 경우, 가상화 오버헤드를 초래하므로 프로비저닝 프로세스에 많은 시간이 초래되는 등 성능이 좋지 못하기 때문에 agent로 구성하여 각 인스턴스에 설치하여 분석하는 형태를 이용하였다. 하지만 이러한 경우에도 컨테이너 기반으로 분석하는 방법보다 속도가 느리다는 단점이 존재한다. Lin Gu et al.의 경우, 컨테이너에서 네트워크 트래픽 제어 프레임워크를 구현하기도 하였다[14]. 속도 향상을 위하여 Linux 트래픽 제어 모듈을 이용하여 적용하는 방식을 고안하였지만, 컨테이너 내부의 네트워크 트래픽 처리 속도가 느리다는 단점이 있다. 본 논문에서는 이러한 단점을 극복하고자 host 커널에 설치되어 모니터링이 가능한 eBPF를 기반으로 네트워크 트래픽을 모니터링하고, 처리속도에 문제가 없도록 구현하였다.

다양한 논문에서 CPU, Memory, Network 등의 정보를 획득한 후, AI를 활용하여 컨테이너의 이상 행위를 탐지하는 기술이 소개되었다[15-17]. AI 기반 컨테이너의 이상행위 탐지를 위하여 컨테이너의 Monitoring, 수집된 정보에 대한 Data processing module, training data 확보를 위한 Fault injection module 등을 구축하는 것은 각기 논문에서 비슷한 동작 원리로 소개되고 있다. 이러한 논문들에서는 CPU, Memory 등에서 발생하는 System call을 수집 및 분석한다. Container ADS(Anomaly Detction System)를 소개하는 논문에서는 각 서비스 및 컨테이너에서 수집되는 CPU 메트릭, Memory 메트릭, 및 Network 메트릭 정보를 수집하였다[18]. [그림 2]과 같은 실시간 성능 데이터를 수집하기 위해 각 VM 에 모니터링 에이전트를 설치하고 대상

시스템의 성능 측정지표를 수집하고 Heapster가 이러한 데이터를 그룹화하고 InfluseDB라는 time-series 데이터베이스에 저장하는 동안 cAdvisor는 모든 컨테이너의 리소스 사용량 및 성능 모니터링 데이터를 수집하는 역할을 모니터링 모듈이 담당한다. Monitoring module에 의해 분류된 모델들은 Data processing module에 의해 DB에서 Data를 가져와 scikit-learn 학습에 포함된 4가지 알고리즘으로 트레이닝하고 DTW 알고리즘을 통해 비정상적인 서비스를 진단한다. 마지막으로 다른 VM에서 bash 스크립트를 실행하여 작동하는 Fault injection module을 통해 컨테이너 이상 징후를 분석하였다.

Metric name	Description
cpu/usage	Cumulative CPU usage on all cores
cpu/request	CPU request (the guaranteed amount of resources) in millicores
cpu/usage-rate	CPU usage on all cores in millicores
cpu/limit	CPU hard limit in millicores
memory/usage	Total memory usage
memory/request	Memory request (the guaranteed amount of resources) in bytes
memory/limit	Memory hard limit in bytes
memory/working-set	Total working set usage Working set is the memory being used and not easily dropped by the kernel
memory/cache	Cache memory usage
memory/rss	RSS memory usage
memory/page-faults	Number of page faults
memory/page-faults-rate	Number of page faults per second
network/rx	Cumulative number of bytes received over the network
network/rx-rate	Number of bytes received over the network per second
network/rx-errors	Cumulative number of errors while receiving over the network
network/rx-errors-rate	Number of errors while receiving over the network per second
network/tx	Cumulative number of bytes sent over the network
network/tx-rate	Number of bytes sent over the network per second
network/tx-errors	Cumulative number of errors while sending over the network
network/tx-errors-rate	Number of errors while sending over the network

그림 2. System call Information from Container

Ⅲ. eBPF 기반 네트워크 및 시스템 모니터링 도구 구현 및 분석

Docker는 기본적으로 docker0 인터페이스를 통해 외부에서 수신한 데이터를 컨테이너에 전달하며, 이 과정을 모니터링하면 컨테이너의 모든 네트워크 트래픽을 확인할 수 있다. 본 연구에서는 기존 http-parse-complete.py 코드가 HTTP GET/POST 요청만 수집한다는 한계를 극복하고자, 이를 수정하여 TCP 기반 트래픽 전체를 수집할 수 있도록 docker_network_trace.py를 구현하였다. 또한, execsnoop 도구를 활용하여 컨테이너 내에서 생성되는 새로운 프로세스를 실시간으로 추적하고, 이를 기반으로 systrace와 default라는 두 가지 이상행위 탐지 모듈을 구현하였다. 실험 결과, 본 시스템은 정상적으로 데이터를 수집하고 추적하며, eBPF 기반 구현은 성능 저하 없이 효율적인 모니터링이 가능함을 확인하였다.

가. eBPF를 활용한 Network 트래픽 수집

기존 eBPF 기반 네트워크 트래픽 수집은 http-parse-complete.py 스크립트를 통해 이루어졌으며, 이는 HTTP 프로토콜의 GET 및 POST 방식에 한정된 모니터링만 수행하는 한계가 있었다. 이러한 문제점을 해결하면서, 다양한 TCP 트래픽에 대하여 분석하기 위하여 기존 코드를 수정하여 추가적인 분석이 가능하도록 하였다.

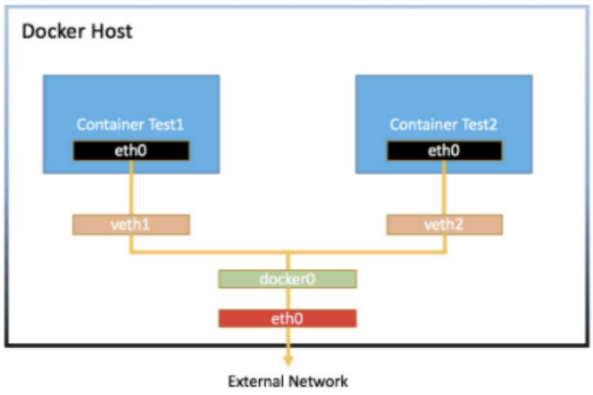

그림 3. Docker network

Docker는 docker proxy가 외부 클라이언트의 연결을 tcp 8080 포트로 받아주고, 내부적으로 컨테이너와의 정해진 통신 방식에 의하여 데이터 컨테이너 80번 포트로 포워딩하게 된다. 외부와 docker-proxy 사이의 통신은 8080/tcp를 사용하고 [그림 3]에서 처럼 docker-proxy와 컨테이너 간의 통신은 docker0라는 내부 bridge를 사용한다.

Docker는 기본적으로 docker0라는 네트워크 인터페이스를 통해 호스트 PC의 외부 인터페이

스(예: eth0)로부터 들어온 데이터를 컨테이너로 전달한다. 이때 -p 옵션을 이용해 호스트에서 수신된 패킷은 docker0를 거쳐 각 컨테이너의 가상 인터페이스(veth)로 전송되므로, docker0 인터페이스를 모니터링함으로써 모든 컨테이너로 전달되는 네트워크 트래픽을 추적할 수 있다.

본 논문에서는 아래 [그림 4]과 같이, bcc tools 중 http protocol로 들어오는 요청을 받아서 처리하는 부분에 패킷이 들어온 것이 있는지 감지하는 http-parse-simple.py를 수정하여 docker_network_trace 시스템을 구현하였다.

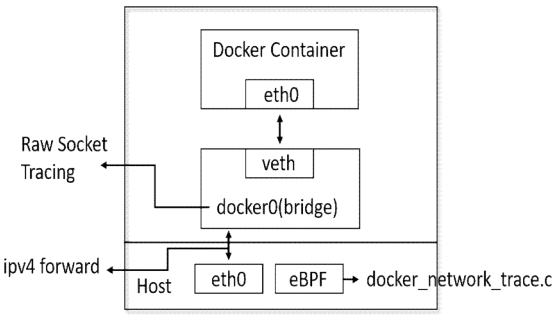

그림 4. Proposed Architecture

Docker는 기본적으로 docker0라는 network interface를 통해서 host pc의 etc로 들어온 데이터를 -p 옵션을 이용하여 호스트에서 forward한 데이터를 docker0에게 전달하여 docker0가 container에게 전달한다. 그러므로 docker0 interface만 볼 수 있다면 container에게 전달되는 모든 데이터를 확인할 수 있다. http-parser-simple을 바탕으로 network interface를 입력받아 해당 interface에 PROG_TYPE_SOCKET_FILTER를 바인드 시키게 되는데, 원본 코드에서는 packet중 http procotcol만 받았지만 해당 부분을 수정하여 tcp의 모든 패킷을 받아 docker_network_trace.py로 전달하여 tracing할 수 있도록 구성하였다. Docker는 기본적으로 docker0라는 network interface를 통해서 host pc의 etc로 들어온 데이터를 -p 옵션을 이용하여 호스트에서 forward한 데이터를 docker0에게 전달하여 docker0가 container에게 전달하기 때문에 docker0 interface만 볼 수 있다면 container에게 전달되는 모든 데이터를 확인할 수 있었다. Bcc에서 제공하는 Http-parser-simple을 바탕으로 network interface를 입력받아 해당 interface에 PROG_TYPE_SOCKET_FILTER를 바인드 시켜 HTTP 트래픽에 대한 정보를 획득할 수 있었다. 이후 기존 http-parse-complete.py의 코드를 수정하여, HTTP 트래픽에 한정되지 않고 TCP 프로토콜의 모든 패킷을 수집할 수 있도록 기능을 확장하였다. 전송된 데이터 추출을 위하여 docker_network_ trace 코드를 활용하였다. Tcp packet을 받아서 해당 packet을 tcp header에 맞게 분리한 다음 전송된 데이터를 추출하는 코드로 네트워크 tracing을 수행할 수 있도록 하였다. 네트워크 tracing 결과는 아래 [그림 5]처럼 정상적으로 tcp의 모든 패킷을 trace 하는 것을 확인할 수 있다. eBPF 적용 여부에 따른 성능 비교 결과, HTTP를 통한 100회 쿼리에서 7.59×10^{-4} sec 정도의 오버헤드가 발생하는 것을 확인할 수 있었으며, 1000회 쿼리에서 8.53×10^{-4}sec 정도의 오버헤드 발생으로 시스템 성능에 미치는 영향은 미미한 것으로 나타났다.

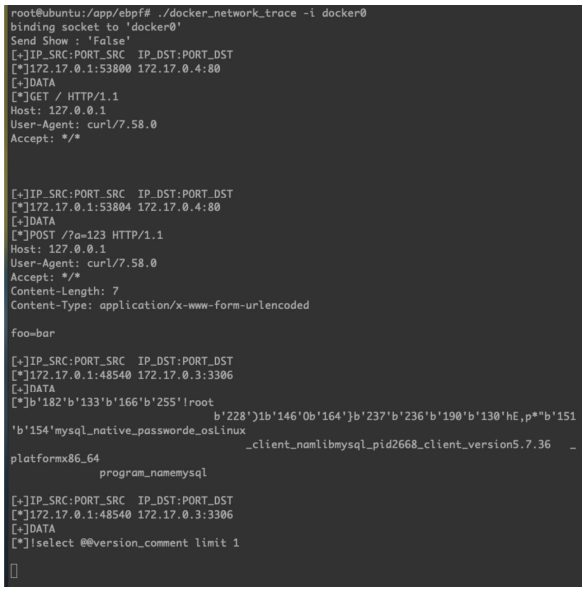

그림 5. Network tracing result

나. eBPF를 이용한 시스템 모니터링

본 시스템은 Ubuntu OS 환경에서 BCC(BPF Compiler Collection)를 기반으로 구현되었으며, 이를 통해 다양한 BCC 도구를 실행하였다.

본 연구에서 개발하는 코드는 BCC에 포함되어 있는 도구이며, [그림 6]과 같이 exec(2)의 변형인 execve(2) 시스템 콜을 트레이싱하며 작동하는 execsnoop라는 tool을 사용하여 진행하였다.

containerd-shim는 컨테이너를 생성하면 runc를 이용하여 container를 생성하고 container가 정상적으로 생성되면 runc는 종료되고 container-shim이 container 내에서 실행되는 프로세스의 부모 프로세스가 된다. 그러므로 container-shim을 pgrep을 통해 pid를 구하고 해당 pid를 기반으로 tracing하면 container에서 발생하는 자식 프로세스를 모두 트레이싱 할 수 있다. 본 연구에서는 BCC 도구인 execsnoop을 기반으로 시스템 호출 중 exec() 함수 사용을 추적하였다. 먼저 pgrep을 통해 실행 중인 컨테이너의 PID를 수집하고, 해당 PID를 기반으로 자식 프로세스를 추적하여 모니터링 대상 PID 리스트를 구성하였다.

execsnoop(8)의 출력 결과는 워크로드 특성화라는 성능 분석 방법을 보조하는 역할을 하며 모든 이벤트 별로 데이터를 출력한다. Containerd-shim은 컨테이너를 생성하면 runc를 이용하여 container를 생성하고 container가 정상적으로 생성되면 runc는 종료되고 container-shim이 container 내에서 실행되는 프로세스의 부모 프로세스가 되기 때문에 러므로 container-shim을 pgrep을 통해 pid를 구하고 해당 pid를 기반으로 tracing하면 container에서 발생하는 자식 프로세스를 모두 트레이싱 할 수 있어 이를 이용하여 구현하였다.

본 논문에서 제안하는 시스템의 실행 예시는 [그림 7]과 같다. exec를 사용하는 부분을 추적하는 bcc_tools의 execsnoop를 기반으로 pgrep를 통해 실행 중인 컨테이너의 pid를 받아오고 해당 pid를 기반으로 하위 프로세스를 추적하여 tracing할 pid list를 만든다. list를 init 단계에서 생성했다면 bpf를 커널에 올려 exec를 이용하여 새롭게 생성되는 child process의 ppid가 pid list에 있다면 해당 프로세스의 cmd line을 출력하고 pid list에 추가한다. 본 연구에서는 execsnoop을 활용하여 exec() 시스템 호출에 의해 생성된 프로세스를 실시간으로 추적하며, 이를 기반으로 두 가지 방식의 모니터링 시스템인 systrace와 default를 개발하였다.

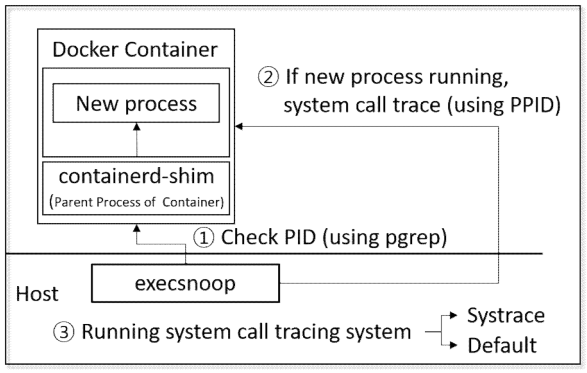

그림 6. Proposed Architecture

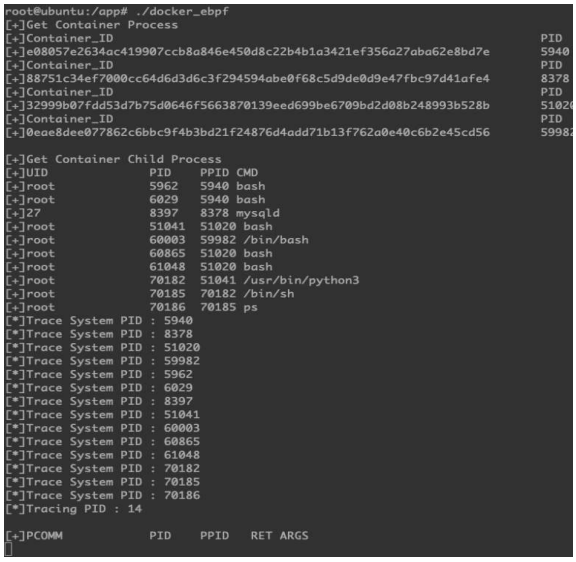

그림 7. execve system call traing result

systrace는 실행하는 순간 현재 모든 프로세스 상태를 받아오고 해당 프로세스들의 부모 자식 리스트를 그려서 어디서 어떤 프로세스가 이상 행위를 하는지, 그리고 해당 프로세스는 어떤

행위를 수행하는 프로세스의 자식인지 확인하는 방법이고 default는 새롭게 생성된 프로세스의 부모 프로세스만 확인하고 해당 프로세스의 Command를 확인하여 이상 탐지를 수행한다. Fault Injection을 통해 실험한 결과, 제안한 시스템은 정상적으로 프로세스를 추적하였으며, eBPF 기반 구현은 시스템 성능에 거의 영향을 주지 않는 것으로 확인되었다.

Ⅳ. 결 론

본 논문은 도커 컨테이너 에서 eBPF를 활용한 bcc tools를 제안하였고 이를 통해 네트워크 패킷 trace를 구현하였다. 실험을 통해 컨테이너에서 네트워크 패킷을 받아오는 타 연구와 비교했을 때 eBPF를 활용하여 패킷을 받아오는 것이 속도적인 측면에서 뛰어난 것을 확인할 수 있었다. 또한, 도커 컨테이너 시스템 환경에서 eBPF를 활용해 system call을 trace 하는 것을 목표로 하고 있다. syscall을 trace하기 위해 BCC tools 중 execsnoop을 수정하여 system과 default 개발해 시스템 호출을 통해서 새롭게 생성된 프로세스를 추적한다. 앞으로 향후 계획은 비지도 학습인 SOM을 이용해 군집화를 이용하여 syscall 을 수집하는 것을 개발할 예정이다.

ACKNOWLEDGEMENT

이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원-대학ICT연구센터(ITRC)의 지원을 받아 수행된 연구임(IITP-2025-RS-2020-II201602)

REFERENCES

[1] 김원용, et al. "리눅스 컨테이너 기반의 운영체제 수준 가상화 연구," *한국정보과학회 학술발표논문집*, 1226-1228쪽, 2015년

[2] Wu, Huichao, et al. "Research on Real-Time Data Acquisition Technology of Dispatching Automation System Based on Container," *2025 International Conference on Electrical Automation and Artificial Intelligence (ICEAAI).* IEEE, 2025.

[3] Volpert, Simon, et al. "Exemplary determination of cgroups-based qos isolation for a database workload," *Companion of the 15th ACM/SPEC International Conference on Performance Engineering.* 2024.

[4] He, Yi, et al. "Cross container attacks: The bewildered {eBPF} on clouds." *32nd USENIX Security Symposium (USENIX Security 23), Mar.,* 2023.

[5] VS, Devi Priya, Sibi Chakkaravarthy Sethuraman, and Muhammad Khurram Khan. "Container security: precaution levels, mitigation strategies, and research perspectives," *Computers & Security* 135 (2023): 103490.

[6] Alyas, Tahir, et al. "Container performance and vulnerability management for container security using docker engine." *Security and Communication Networks* 2022.1 (2022): 6819002.

[7] Acharya, Jigna N., and Anil C. Suthar. "Docker container orchestration management: A review," *International Conference on Intelligent Vision and Computing. Cham: Springer International Publishing,* 2021.

[8] Kithulwatta, W. M. C. J. T., et al. "Adoption of docker containers as an infrastructure for deploying software applications: A review," *Advances on Smart and Soft Computing: Proceedings of ICACIn 2021* (2021): pp.247-259.

[9] Haq, Md Sadun, et al. "SoK: A comprehensive analysis and evaluation of docker container attack and defense mechanisms," *2024 IEEE Symposium on Security and Privacy (SP).* IEEE, 2024.

[10] Liu, Chang, et al. "A protocol-independent container network observability analysis system based on eBPF," *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS).* IEEE, 2020.

[11] Gwak, Songi, Thien-Phuc Doan, and Souhwan Jung. "Container Instrumentation and Enforcement System for Runtime Security of Kubernetes Platform with eBPF," *Intelligent Automation & Soft Computing* vol. 37, no. 2, pp. 1773 - 1786, 2023.

[12] Weng, Tianjun, et al. "Kmon: An in-kernel transparent monitoring system for microservice systems with ebpf," *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence).* IEEE, 2021.

[13] Xu, Chen, et al. "Network intrusion detection system as a service in OpenStack Cloud," *2020*

international conference on computing, networking and communications (ICNC). IEEE, 2020.

[14] Gu, Lin, et al. "Container session level traffic prediction from network interface usage," *IEEE Transactions on Sustainable Computing,* vol 8, no 3, pp. 400-411. 2023

[15] Liu, Chang, et al. "A protocol-independent container network observability analysis system based on eBPF," *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS).* IEEE, 2020.

[16] Lee, Chunghan, Reina Yoshitani, and Toshio Hirotsu. "Enhancing packet tracing of microservices in container overlay networks using EBPF," *Proceedings of the 17th Asian Internet Engineering Conference.* 2022.

[17] Yang, Wanqi, and Pengfei Chen. "eProbe: eBPF-enhanced Accurate Container Status Probing in Cloud-native Systems," *IEEE Transactions on Services Computing* (2025).

[18] Du, Qingfeng, Tiandi Xie, and Yu He. "Anomaly detection and diagnosis for container-based microservices with performance monitoring," *International Conference on Algorithms and Architectures for Parallel Processing. Cham: Springer International Publishing,* 2018.

저 자 소 개

이택규 (정회원)

1999년 아주대학교 전자공학과 학사 졸업
2001년 아주대학교 전자공학과 석사 졸업
2003년 고려대학교 전파공학과 박사 수료
2013년~현재 충남대학교 컴퓨터공학과 박사과정

<주관심분야 : 클라우드 보안, N2SF, 제로트러스트>

박정수 (정회원)

2013년 숭실대학교 정보통신전자공학부 학사 졸업
2015년 숭실대학교 전자공학과 석사 졸업
2021년 숭실대학교 융합소프트웨어학과 박사 졸업
2024년~현재 강남대학교 컴퓨터공학부 조교수

<주관심분야 : 제로트러스트, N2SF, 컨테이너 보안>