

Fast Packet Processing with eBPF and XDP:
Concepts, Code, Challenges and Applications
요약 정리

[18반 라민우]

(!) 격주 보고서 작성에 필요한 내용만을 포함하기 위해 eBPF/XDP에 대한 설명을 최대한 간략하게 정리하고 핵심 정보를 중심으로 서술했다.

(!) 이전에 전달했었던 요약본에서 더할 수 있는 부분을 추가했습니다.

▶ eBPF란?

다양한 명령어 체계 , 실행 환경(VM 환경)을 리눅스 커널에서 제공하는 시스템

커널 단에 프로그래머가 원하는 프로그램을 적용할 수 있게 하자는 것이 기본적인 목적이다.

BPF는 cBPF(구버전), eBPF 로 나눌 수 있으며, eBPF 기준 레지스터, 스택, 맵을 구현한 BPF machine 체계를 갖추고 있다.

▶ eBPF 프로그램의 실행 과정

[user space] 에서의 기본적인 과정을 설명하자면, 고수준 프로그래밍 언어를 통해 코드를 구성하고, LLVM/clang 을 통해 프로그램을 커널에서 작동할 코드를 생성한다.(기본적으로 많이들 사용하는 방법이지만, 다양한 개발 툴이 존재한다. ex bcc(파이썬으로 개발)) [user space] 에서의 동작에서 있어서 핵심은 컴파일이라고 보면 된다.

적용할 코드를 실제로 커널 단에 삽입하고 실행하는 과정에서는 ELF eBPF loader 가 관여하는데, 컴파일을 통해 생성된 eBPF ELF 파일에 내재된 일련의 메타 데이터가 제공해주는 정보를 토대로 동작한다.

[kernel space]에서는 위 [user space]로부터 받은 BPF 코드를 실제로 동작을 요구할 커널 요소에 적용하기 위한 일련의 검증 & 컴파일 과정을 실시한다. 커널 공간에 삽입할 코드에 대한 검증은 'Verifier'가 담당하며 관련 알고리즘은 엄격한 기준을 갖고 있다는 점이 특징이다. 이후 'JIT compiler'를 통해 kernel 단에서 작동할 수 있는 기계어로 변환하면, 실제로 eBPF 프로그램을 원하는 커널 요소(우리는 XDP 구현을 위해 Network Device Driver에 연결한다)에 본격적으로 부착할 수 있게 된다.

● 필요하실 때 참고하시면 되겠습니다.

eBPF 프로그램이 작동되기까지의 과정은 이러하다.(논문의 내용에서 살짝 추가한 정도)

eBPF 소스코드 작성(.c) -> 컴파일 -> 오브젝트 파일(.o) -> 커널로 로드(커널로 로드한다는 것이 커널이 이를 수용했다 것은 아니다) -> Verifier(매우 엄격하게 점검) -> JIT(Kernel에서 eBPF 프로그램으로 작동할 수 있도록 컴파일, Just-In-Time 컴파일러 : 프로그램이 동작하는 와중에 컴파일을 해주는 컴파일러) -> Hook에 부착 -> Process

▶ 아래의 그림을 통해 eBPF의 다양한 활용 경로를 시각적으로 확인할 수 있음.

(단, eBPF/XDP의 실질적인 동작 과정에 대한 설명보다는 eBPF의 다양한 활용 가능성을 보여주는 간단한 지표로써 보는 것이 맞아보임)

eBPF 핵심 키워드

eBPF > VM, Instruction set, Verifier, JIT, Map, Helperfunction

eBPF Program Type 중 핵심 등장 요소 : XDP, TC