K-ONE 기술 문서 #41

데이터 중심 보안 기술을 위한 eBPF/XDP 기반 동적 패킷 모니터링 및 필터링

Document No. K-ONE #41
Version 1.0
Date 2019-11-24
Author(s) 최영은, 신준식

K-ONE

ONK
OpenNetworkingKorea

■ 문서의 연혁

버전	날짜	작성자	내용
0.1	19.11.04	최영은	Draft 작성
0.9	19.11.23	최영은	Draft 작성 완료
1.0	19.11.24	최영은	오탈자 검증

본 문서는 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신 기술진흥센터의 지원을 받아 수행된 연구임 (No. 2015-0-00575, 글로벌 SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. 2015-0-00575, Global SDN/NFV OpenSource Software Core Module/Function Development)

기술문서 요약

DoS 공격은 볼륨 공격과 에플리케이션 공격으로 분류할 수 있다. 본 기술문서는 DoS 에플리케이션 공격의 한 종류인 HTTP flooding 공격을 다루며 해당 공격의 방어를 위하여 eBPF(extended Berkeley Packet Filter)/XDP(eXpress Data Path) 프로그램을 이용한다. eBPF/XDP 프로그램을 이용하여 물리 서버로 수신되는 네트워크 패킷들을 모니터링하며 eBPF/XDP 프로그램의 모드 중 하나인 hardware offload 모드를 이용, 수신되는 패킷들을 패킷 처리 시 최초 수신 단인 NIC(Network Interface Card)에서 직접 수행한다. eBPF/XDP hardware offload 모드를 이용, 패킷 필터링 시 패킷 필터링은 NIC으로 offload 되며, 호스트 CPU를 사용하지 않는다. eBPF/XDP 프로그램에서 수신한 패킷에 대한 정보들은 Kafka를 이용하여 Onion-Ring 가시화 프로그램으로 전송된다.

Onion-Ring 가시화 프로그램은 Python에 기반한 Plotly의 Sunburst Char를 이용한 물리 서버의 네트워크 인터페이스들과 해당 인터페이스들의 네트워크 트래픽 수신 상태를 확인 가능한 가시화 프로그램이다. Onion-Ring 가시화 프로그램은 특정 네트워크 인터페이스에서 수신 중인 패킷의 개수를 기준으로 하여 네트워크 인터페이스의 네트워크 트래픽 상태를 가시화한다. 또한, Plotly의 상호작용 기능을 이용하여 해당 네트워크 인터페이스에서 수신 중인 네트워크 트래픽의 송신 IP 주소를 출력한다.

eBPF/XDP 프로그램과 Onion-Ring 가시화 프로그램의 작동을 검증하기 위하여 Kubernetes 서비스에 Goldeneye DoS 공격툴을 이용, DoS 공격 트래픽을 전송하고 이를 eBPF/XDP 프로그램으로 필터링한다. 이때 호스트의 CPU를 사용하지 않으며 DoS 공격 트래픽이 필터링 되고 있음을 검증한다. 또한, Onion-Ring 가시화 프로그램을 이용하여 DoS 공격을 받는 네트워크 인터페이스와 해당 네트워크 인터페이스를 사용 중인 물리 서버를 가시화하는 것을 확인한다.

Contents

K-ONE #41. 데이터 중심 보안 기술을 위한 eBPF/XDP 기반 동적 패킷 모니터링 및 필터링

그림 목차

표 목차

K-ONE #41. 데이터 중심 보안 기술을 위한 eBPF/XDP 기반 동적 패킷 모니터링 및 필터링

1. 서론

DoS (Denial of Service) 공격은 서버 및 컴퓨터의 리소스를 고갈시킴으로서 서비스를 이용하는 사용자들이 더 이상 정상적인 서비스를 제공받지 못하도록 만든다. 본 서론에서는, DoS 공격 방식들 중 DoS 볼륨 공격과 DoS 에플리케이션 레이어 공격에 대해 살펴본다.

1.1. DoS 볼륨 공격

DoS 볼륨 공격은 거대한 양의 네트워크 트래픽을 서비스 제공자의 서버로 전송하며 이뤄진다. 보통은 여러 대의 컴퓨터를 동원한 DDoS (Distributed Denial of Service) 공격의 형태로 진행되며 이때 전송되는 네트워크 트래픽의 종류에 따라 DoS 볼륨 공격의 종류가 구분된다. DoS 볼륨 공격은 ICMP (Internet Control Message Protocol) Flood, IP/ICMP Fragmentation, UDP (User Datagram Protocol) Flood, 그리고 DNS Amplification 등으로 구분할 수 있다 [6].

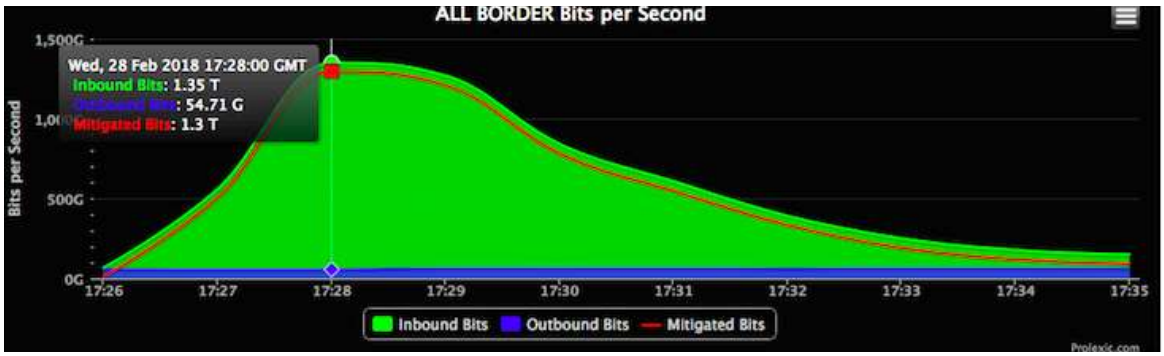

그림 1 2018년 2월 28일 17:21-17:30, Github에 진행되었던 DDoS 공격 트래픽

ICMP flood 공격, 혹은 Ping flood 공격은 공격자가 공격 대상에게 막대한 양의 ping (ICMP echo-request)를 전송하는 방식이다. ICMP flood 공격의 공격 대상은 특정한 컴퓨터, 로컬 네트워크 또는 라우터이다. ICMP echo-request를 받은 공격 대상은 각각의 요청에 대한 응답을 전송하게 된다. 공격 대상의 네트워크 대역폭은 공격자의 ICMP echo-request와 공격 대상자 자신의 응답 신호에 잠식되며 다른 서비스 이용자는 공격 대상자의 서비스 접근이 어려워진다 [7].

IP/ICMP fragmentation 공격은 datagram 의 IP datagram 이 fragment로 나뉘어 전송된 후 수신지에서 fragment들이 다시 재조립되는 메커니즘을 이용한다. 송신자가 fragment 전송 시 fragment에 대한 모든 정보를 포함하는 패킷이 첫 패킷으로 전송되며 나머지 패킷들은 IP 헤더와 payload만을 갖는다. 공격자는 공격 대상에게 재조합이 불가능한 fragment 들을 전송한다. 공격 대상자는 임시 저장소에 해당

fragment 들을 저장하며 다량의 공격 fragment들이 전송될 경우 공격 대상자는 가용한 모든 저장소 자원을 사용하게 된다 [8].

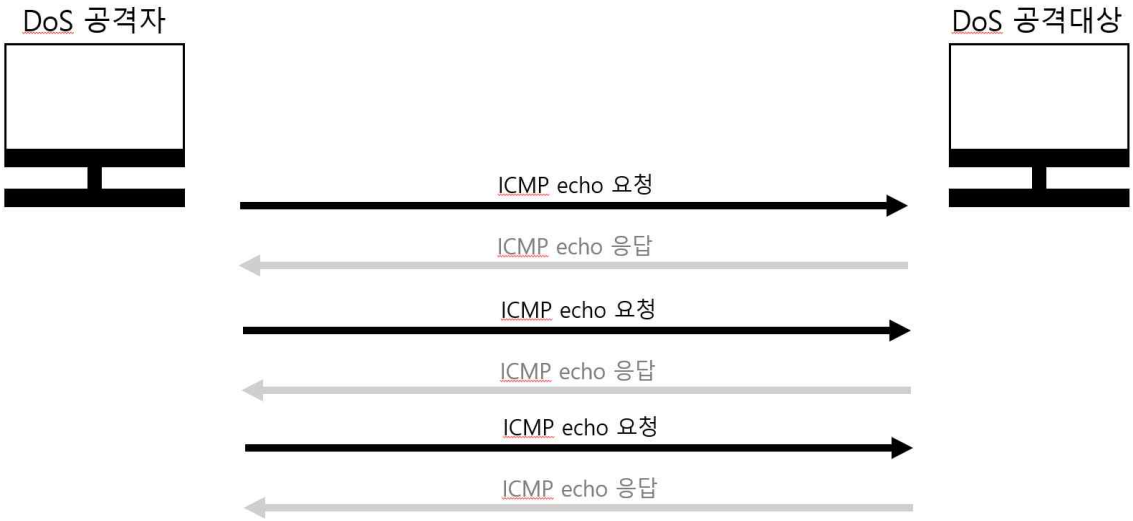

그림 2 ICMP flood DoS 공격

UDP flood 공격은 공격자가 공격 대상의 무작위한 포트에 UDP 패킷들을 전송하는 방식이다. 공격 대상은 수신된 datagram 과 연관된 에플리케이션을 검색하고, 관련된 에플리케이션이 존재하지 않으므로 공격자에게 “수신 불가” 패킷을 전송한다. 다량의 UDP flood 공격 트래픽이 전송될 경우, 공격 대상자의 시스템은 마비되어 정상적인 서비스 이용자들은 공격 대상자의 서비스에 정상적으로 접근이 불가능하다 [9].

DNS Amplification 공격은 reflection에 근거한 DoS 공격이며 보통 DDoS 공격의 형태로 진행된다. DoS 공격자는 공개된 DNS resolver의 기능을 이용하여 공격 대상자의 서버 및 네트워크를 증폭된 양의 트래픽을 이용하여 서버와 주변의 인프라스트럭처를 사용 불가능하게 만든다. DoS 공격자가 DNS resolver에게 대량의 응답 신호 트래픽을 야기하는 쿼리들을 전송함으로서, DoS 공격자는 상대적으로 적은 양의 트래픽을 전송함으로서 DoS 공격 대상자가 대량의 트래픽을 수신하도록 만든다 [2].

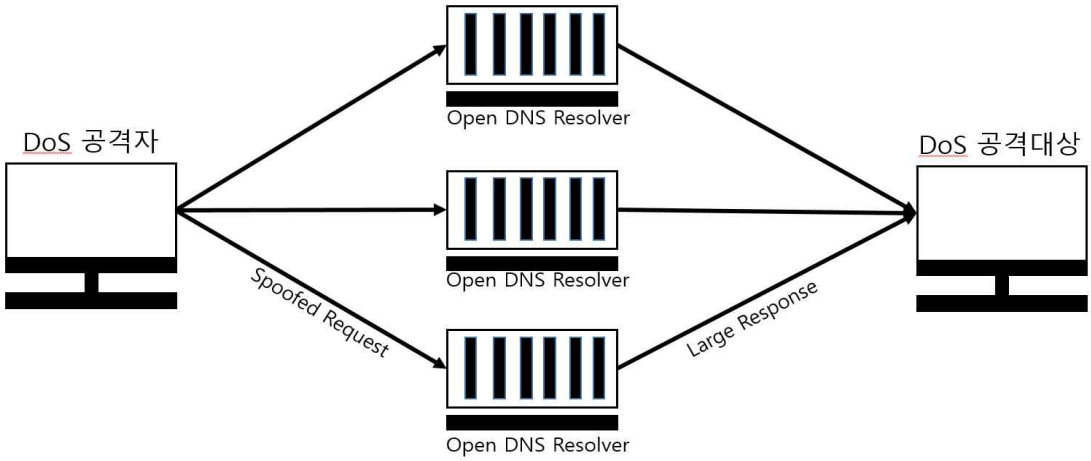

그림 3 Reflection Amplification DoS 공격

1.2. DoS 에플리케이션 공격

에플리케이션 공격은 웹 서버와 같은 에플리케이션 자체를 공격하여 에플리케이션이 사용자의 요청에 응답할 수 없는 상태로 만든다. 이러한 공격의 형태는 볼륨적인 측면에서 적은 양의 트래픽을 이용하며 프로토콜의 핸드 쉐이크 및 규칙을 이용한다. DoS 볼륨 공격과 다르게 DoS 에플리케이션 공격은 많은 양의 트래픽을 사용하지 않는다. DoS 에플리케이션 공격의 방식에는 BGP(Border Gateway Protocol) Hijacking, Slowloris, Slow Post, HTTP Flooding 등의 공격 방식이 존재한다 [10].

BGP Hijacking은 인터넷상에서 트래픽을 한 네트워크에서 다른 네트워크로 유도할 때 사용된다. 공격자는 자신이 특정한 또는 가상의 네트워크에 속해있다고 공격대상을 속이며 해당 정보가 실제 존재하는 다른 네트워크에서 받아들여질 경우, 트래픽은 실제 해당 수신지가 아닌 공격자의 주소에서 수신된다. 해당 공격 방식은 네트워크 지연을 증가시키며 공격 대상의 트래픽 유실로 인한 금전적인 손해를 입힐 수 있다 [11].

그림 4 Slowloris DoS 공격

Slowloris 공격은 부분적인 HTTP 요청을 이용하여 단일 컴퓨터와 공격 대상 웹 서버를 연결하고, 해당 연결을 최장 시간 유지한다. 공격 대상자의 해당 서비스는 속도가 느려지지만, 공격 대상의 다른 서비스와 포트들은 영향을 받지 않는다. Slowloris의 공격 연결들이 종료된 후에도 Slowloris는 계속해서 재연결을 시도하여 지속적으로 공격 대상자를 공격한다 [12].

Slow Post 공격은 공격자가 공격 대상의 웹 서버에게 헤더들 중 Content-Length의 값을 임의의 큰 값을 설정하여 전송하는 방식이다. 공격 대상의 웹 서버는 클라이언트에게 해당 크기의 메세지 수신 완료 시까지 연결을 유지하도록 만든다. DoS 공격자는 소량의 데이터를 느린 속도로 전송하여 웹 서버와의 연결을 최장 시간 유지, 다른 사용자에 대한 서비스를 불가능하게 만든다 [13].

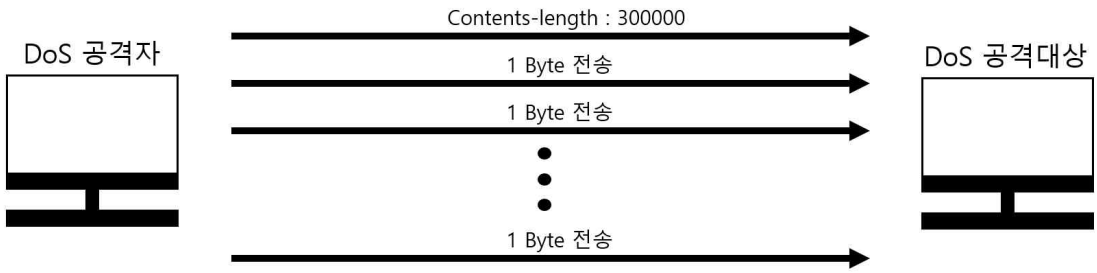

그림 5 Slowpost DoS 공격

HTTP Flooding 공격은 정상적인 HTTP GET 또는 HTTP POST 요청을 공격 대상의 웹 서버 또는 에플리케이션에 전송하는 방법이다. 이런 식의 요청을 많이 보내는 flooding 공격은 보통 botnet, 즉 말웨어 나 트로이 목마에 감염된 컴퓨터들의 그룹으로부터 행해진다. 본 기술 문서에서 기술할 데이터 중심 보안 기술을 위한 eBPF/XDP 기반 동적 패킷 모니터링 및 필터링 프로그램은 HTTP Flooding 공격에 대한 방어를 구현한 프로그램이다 [14].

1.3. 리눅스 방화벽

[3]에 따르면, 수년간 리눅스의 방화벽과 패킷 필터링은 iptables가 주를 이루었다. iptables가 처음 도입되었을 당시는 모뎀이 주를 이루었었으며 네트워크의 속도가 느렸었다. iptables는 수신되는 모든 패킷을 순차적으로 모든 방화벽의 규칙과 비교하여 검사한다. 이와 같은 방식은 규칙의 개수 증가 시, 패킷의 필터링을 위한 비용이 선형적으로 증가한다는 단점이 있다. 시간이 지나며 네트워크의 속도가 향상됨에 따라 방화벽의 규칙은 수십 개에서 수천 개로 증가하였고, 리눅스 방화벽의 속도 향상을 위하여 ipchains가 도입되었다. ipset은 IP 주소와 포트의 조합을 해쉬 테이블에 저장하여 전체적인 iptables의 규칙의 개수를 줄인다. 하지만 ipchains는

Kubernetes에는 적합하지 않다. Kubernetes의 한 컴포넌트는 iptables와 DNAT(Destination Network Address Translation) 규칙을 사용하여 서비스에 대한 로드 발런싱을 진행한다. Kubernetes는 서비스가 서비스를 제공하는 각각의 백 엔드에 대하여 iptables 다수의 iptables 규칙을 설정하며, Kubernetes에 서비스가 추가됨에 따라 검사가 진행되어야 할 iptables 규칙의 개수는 기하급수적으로 증가하기 때문이다. Kubernetes 서비스의 숫자가 증가함에 따라 지연율은 예상 불가능하게 변하며 성능 또한 저하된다. 추가적인 iptables의 한계점 중 하나는 iptables의 규칙이 하나 더 추가되기 위하여 규칙 전체의 리스트가 교체되어야 한다는 것이다. iptables는 20000개의 Kubernetes 서비스에 대하여 160000개의 iptables 규칙을 설치하기 위하여 약 5시간이 소요되는 정도의 성능을 보인다.

eBPF의 발전과 함께 별도의 커널 개발이 필요했던 기능들이 커널 개발 없이도 가능하게 되었다. eBPF를 이용하여 iptables를 완전히 대체하는 노력은 현재 계속되고 있으며, eBPF의 iptables 대체는 이미 존재하는 itables 클라이언트와 라이브러리들이 정상 동작하는 상태에서 이뤄지고 있다. 그림 6은 iptables, nftables와 bpfilter의 성능을 비교한다 [15].

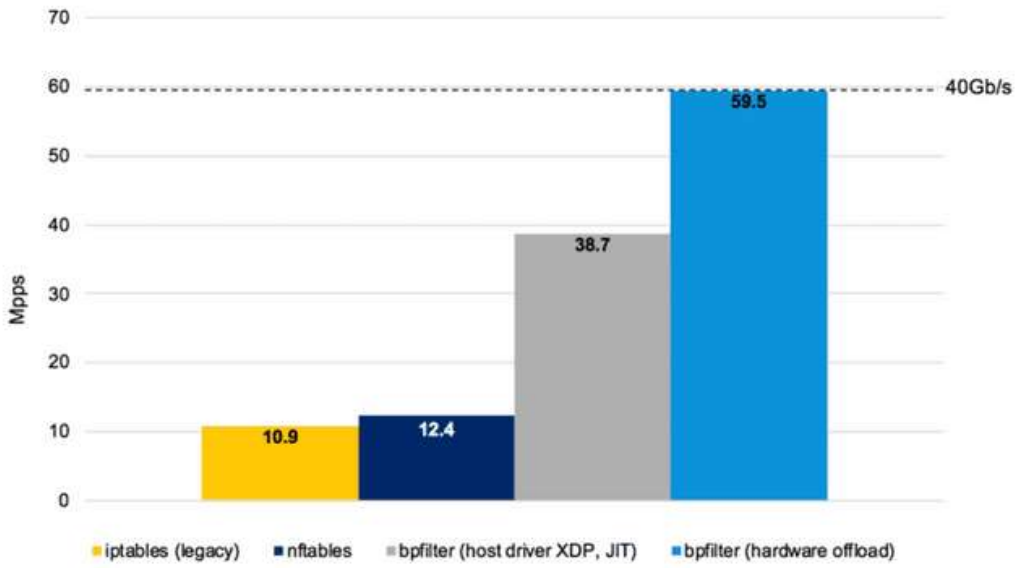

그림 6 iptables, nftables, bpfilter의 성능 비교

2. 본론

본 절에서는 실험을 위하여 사용한 DoS 공격 툴인 Goldeneye에 대해 설명한다. 또한, eBPF/XDP에 대한 설명과 함께 eBPF/XDP를 이용하여 구현된 모니터링 및 필터링, 그리고 Onion-Ring 가시화에 관해 설명한다. DoS 방어 성능 검증을 위하여 HTTP flooding DoS 공격 툴인 Goldeneye가 사용되었다. eBPF/XDP 프로그램과

실험은 HTTP flooding DoS 공격이 이미 탐지된 상황을 가정한다.

2.1. Goldeneye DoS 공격툴

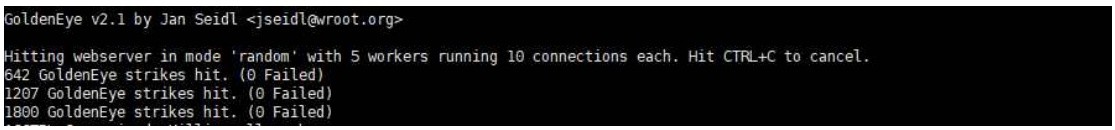

그림 7 Goldeneye DoS 공격

eBPF/XDP 프로그램의 DoS 공격 트래픽에 대한 패킷 모니터링 및 필터링 성능을 검증하기 위하여 Goldeneye DoS 공격툴을 이용한다. Goldeneye DoS 공격 툴은 DoS 테스트를 위하여 공개된 python 기반 오픈 소스 툴이다. Goldeneye DoS 공격 툴은 공격 대상의 컴퓨팅 자원을 모두 소모하게 해 서비스 불능 상태로 만드는 것이 목적이다. Goldeneye DoS 공격툴이 지원하는 운영체제는 Windows, MAC, 그리고 리눅스이다. Goldeney DoS 공격툴은 자신이 전송하는 공격 트래픽을 암호화하지 않으며, 자신의 IP 송신 주소를 다른 IP 송신 주소로 위조하는 IP spoofing 또한 하지 않는다. Goldeneye DoS 공격툴은 GUI 인터페이스를 지원하지 않으며 CLI 인터페이스만을 지원한다. Goldeneye DoS 공격툴은 지정된 공격 대상에게 에플리케이션 DoS 공격 중 HTTP flooding 공격을 진행하며 공격 대상은 IP 주소와 포트 번호를 이용하여 지정할 수 있다. 공격에 있어 몇 가지의 옵션을 설정할 수 있으며, 설정할 수 있는 옵션은 표 1과 같다 [16].

Flag	설명	보통값
-u	user-agent와 사용할 파일	random 발생
-w	동시에 발생시킬 작업자 수	50
-s	동시에 발생시킬 소켓 수	30
-m	HTTP method (get / post / random)	get
-d	Debug 모드 toggle	False
-n	SSL Certificate verification 끄기	True
-h	help 메세지 출력	-

표 1 Goldeneye DoS 공격툴 옵션

2.2. eBPF/XDP

eBPF는 리눅스 커널의 다양한 hook point에서 안전하게 bytecode를 실행시킬 수 있는 프로그램이다. eBPF의 전신인 BPF(Berkeley Packet Filter)는 1992년 처음 만들어졌으나, eBPF는 커널 3.18 에서 처음 등장하였으며, 이후부터 BPF는 cBPF(classic Berkeley Packet Filter)라는 이름으로 불리었다 [17]. eBPF 프로그램이 실행되는 단

계는 다음과 같다 :

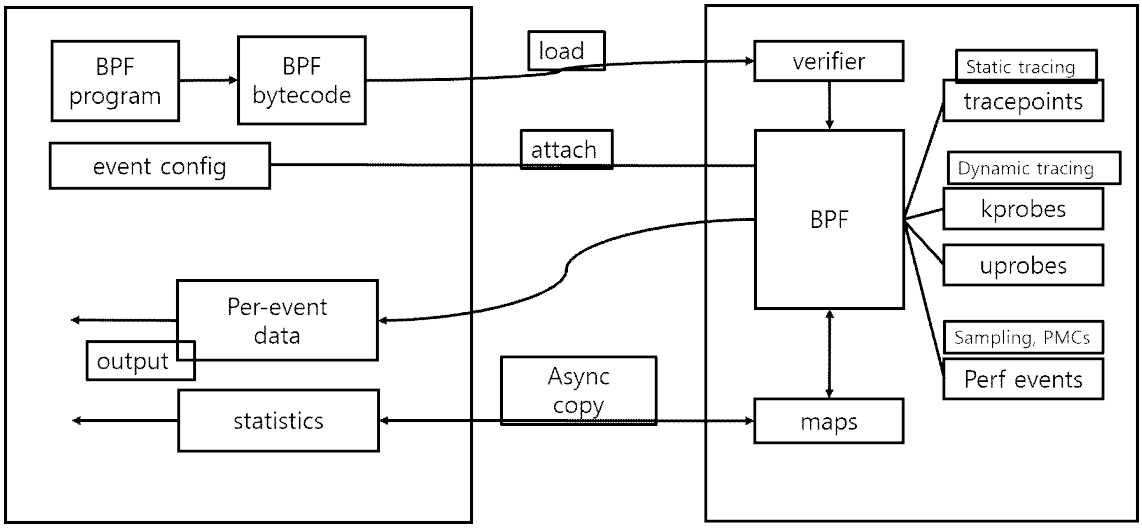

그림 8 eBPF 프로그램 동작 방식

1. 사용자 공간에서 커널 공간에 bytecode 와 program type을 전송한다.
2. 커널이 bytecode에 대한 verifier를 실행시키며 해당 eBPF 프로그램에 대한 안정성을 검사한다.
3. 커널이 bytecode를 native code로 JIT(Just In Time) 컴파일한 뒤 해당 코드의 자리를 찾아 해당하는 위치에 해당 코드를 부착한다.
4. 주입된 코드가 데이터를 ring buffer 및 generic key-value eBPF map에 저장한다.
5. 사용자 공간 프로그램이 ring buffer 및 eBPF map으로부터 결과값을 불러온다.

eBPF는 instruction set뿐만 아니라 key, value를 이용하여 데이터를 저장하는 eBPF map, 커널의 기능들과 상호작용을 지원하는 helper function, 타 eBPF 프로그램을 호출하는 tail call과 같은 infrastructure 또한 제공한다 [18]. LLVM은 eBPF의 back end를 제공하며 clang과 같은 툴을 이용하여 C 코드를 후에 커널에 로드 가능한 BPF object 파일로 변환한다. eBPF를 사용하는 커널의 subsystem 또한 eBPF의 infrastructure에 포함된다. eBPF/XDP는 eBPF 프로그램을 네트워킹 드라이버의 가장 초기 단계에 부착하여 패킷 수신 시 eBPF 프로그램을 동작시킨다. 네트워킹 드라이버 이전 단계에서의 패킷에 대한 소프트웨어적 처리는 불가능하므로 해당 방식은 최적의 패킷 처리 성능을 갖는다. eBPF/XDP는 네트워킹 스택이 패킷의 메타데이터를 추출하기 전 네트워킹 스택의 가장 아랫단에서 패킷을 처리한다. 반면, TC(Traffic Control) eBPF 프로그램들은 커널 스택 내에서 이후에 작동하므로 TC eBPF 프로그램들은 메타 데이터와 코어 커널 기능들을 사용할 수 있다.

eBPF의 Helper function들은 eBPF 프로그램이 코어 커널에 정의된 함수들을 이용

하여 커널과 데이터 통신을 할 수 있도록 도와준다. eBPF 프로그램의 타입에 따라 가용 helper function들은 달라진다. 예를 들어, 소켓에 부착된 eBPF 프로그램은 tc layer에 부착된 eBPF 프로그램 대비 helper function의 sub set만을 호출할 수 있다. lightweight 터널링을 위한 encapsulation과 decapsulation helper들은 lower tc layer의 eBPF 프로그램들만이 호출할 수 있지만, tc와 eBPF/XDP 프로그램들은 사용자 공간 프로그램에 알림을 주기 위한 용도의 helper function들이 호출할 수 있다.

eBPF 프로그램은 커널 공간 프로그램과 사용자 공간 프로그램으로 나눌 수 있으며, 상호 간의 통신을 위하여 eBPF map을 사용한다 [I8].

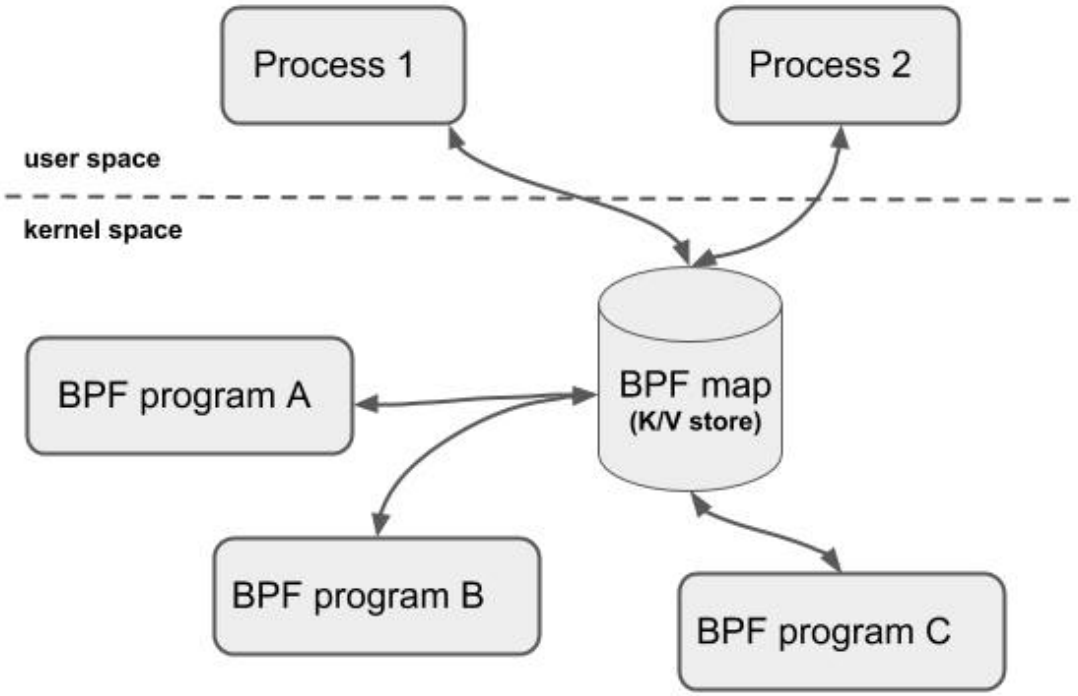

그림 9 eBPF map과 커널 공간과 사용자 공간

eBPF map은 커널 공간에 존재하며 key / value를 저장한다. eBPF map을 이용하기 위하여 eBPF 프로그램이 반드시 같은 타입의 프로그램일 필요는 없다. 단일 eBPF 프로그램은 최대 64개의 다른 eBPF map과 통신을 할 수 있다. eBPF map은 임의의 데이터 저장 및 읽기가 가능한 generic map과 helper function들과 함께 사용되는 non-generic map으로 분류된다. 본 기술 문서에서 서술하는 프로그램이 사용하는 eBPF map들은 generic eBPF map으로서 BPF helper function을 이용하여 eBPF map 값을 저장, 삭제 또는 읽어온다.

eBPF/XDP는 수신된 패킷들에 한하여 처리할 수 있다. eBPF/XDP가 수신된 패킷을 처리하는 위치는 eBPF/XDP의 동작 모드에 따라 다르며, eBPF/XDP 동작 모드

는 generic, native, hardware offload 모드가 있다. generic 모드는 소켓 퍼버가 패킷에 할당된 뒤 처리를 시작한다[7]. generic 모드는 eBPF/XDP 동작 모드 중 패킷 처리에 최장 시간이 필요하지만, 하드웨어 지원이 필요 없으므로 모든 NIC에서 사용 가능한 모드이다. native 모드는 수신 패킷의 네트워크 드라이버 초기 진입 상황에서 eBPF/XDP 프로그램을 실행시킨다 [8]. native 모드는 generic 모드 보다 더 나은 성능을 보여주지만, 하드웨어 드라이버의 지원이 필요하므로 모든 NIC에서 사용 가능한 모드는 아니다. hardware offload 모드는 eBPF/XDP 프로그램을 호스트 CPU 대신 NIC으로 완전히 offload 시킨다. hardware offload 모드는 eBPF/XDP의 모드들 중 최상의 성능을 보이지만, NIC의 하드웨어 지원이 필요하며 보통 smart NIC들은 eBPF/XDP를 지원한다.

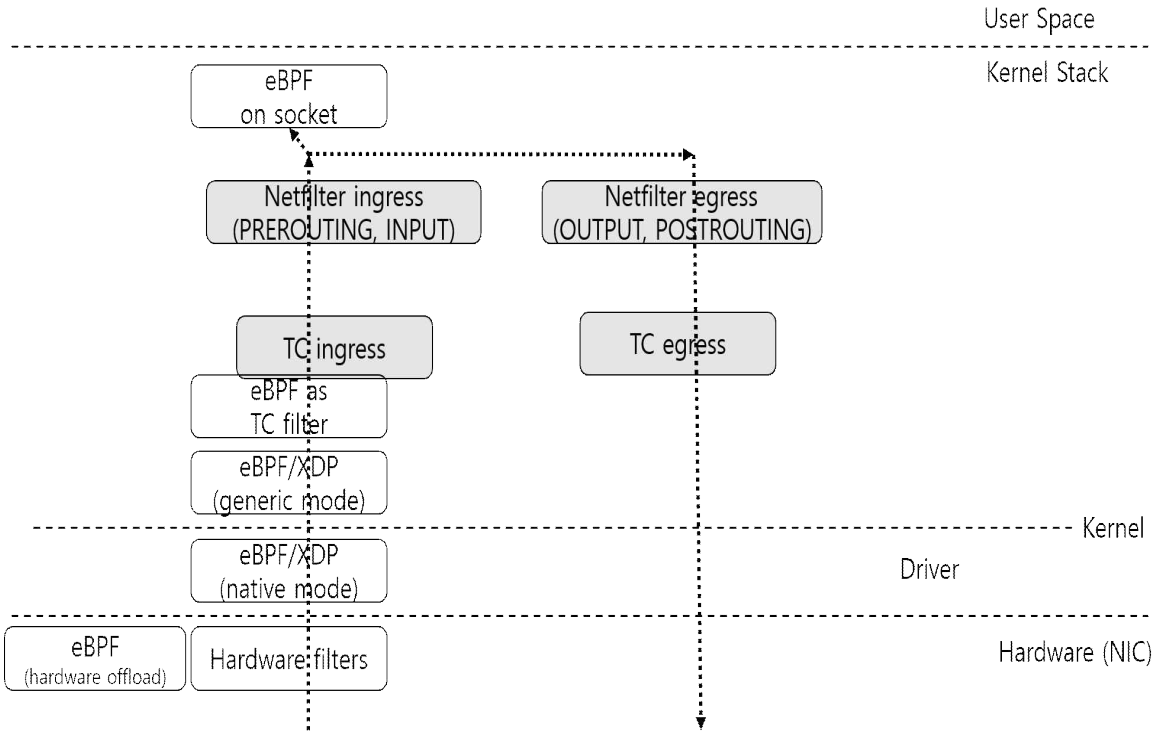

그림 10 eBPF/XDP 동작 모드에 따른 수신 패킷의 처리 위치

eBPF/XDP의 동작 모드와 수신 패킷의 처리 위치는 그림 10과 같으며 [20], eBPF/XDP는 수신 패킷을 처리하기 위한 몇 가지 동작들을 지원하며 이는 표2와 같다.

동작	동작 내용
XDP_PASS	패킷을 기본 네트워크 스택에 전달
XDP_DROP	드라이버/NIC 이 패킷을 누락시킴
XDP_TX	패킷-페이지를 수신되었던 NIC으로 재전송
XDP_REDIRECT	RAW 프레임을 다른 장비로 재전송
XDP_ABORT	eBPF 프로그램 에러 (패킷 누락시킴)

표 2 eBPF/XDP 동작과 동작 내용

2.3. eBPF/XDP 커널 공간 프로그램

eBPF/XDP 동적 모니터링 프로그램이 부착된 네트워크 인터페이스가 네트워크 패킷 수신 시 프로그램은 동작한다. eBPF/XDP 커널 공간 프로그램은 C언어로 작성되었으며, eBPF/XDP 커널 공간 프로그램이 부착된 네트워크 인터페이스의 NIC은 패킷 수신 시 네트워크 패킷의 길이를 확인한다. eBPF/XDP 커널 공간 프로그램은 수신된 네트워크 패킷의 길이가 완전한 이더넷 헤더 포함이 불가능한 작은 패킷을 누락시킨다. 누락되지 않은 네트워크 패킷의 분석을 위하여 eBPF/XDP 커널 공간 프로그램은 이더넷 헤더 분석을 위하여 리눅스 헤더인 'if_ether.h'를 이용, ethhdr 구조체를 선언한다. 선언된 구조체 내의 프로토콜 값을 명시한 'h_proto'의 값이 IPv4를 명시하는 상숫값인 'ETH_P_IP'의 값과 동일한지 확인한다. 본 기술 문서의 프로그램의 타겟 IP 프로토콜은 IPv4이므로 'ETH_P_IP'의 값과 다른 값을 'h_proto'에 명시한 네트워크 패킷들은 누락된다. 해당 검사가 진행되기 이전, 프로그램이 실행되는 물리 머신의 하드웨어에 따라 host byte order가 달라지는 점을 고려한다 [19].

```
117  struct ethhdr {
118          unsigned char   h_dest[ETH_ALEN];       /* destination eth addr */
119          unsigned char   h_source[ETH_ALEN];     /* source ether addr    */
120          __be16          h_proto;                /* packet type ID field */
121  } __attribute__((packed));
122
```

그림 11 if_ether.h 헤더 내용

수신되는 모든 네트워크 패킷들은 network byte order인 big-endian을 사용하지만, 프로그램이 실행되는 물리 머신의 host byte order는 little-endian일 가능성이 있으므로 eBPF/XDP 공간 커널 프로그램은 'ETH_P_IP'의 값을 host byte order에서 network byte order로 변경 후 변경된 값이 'h_proto'의 값과 동일한지 검사한다.

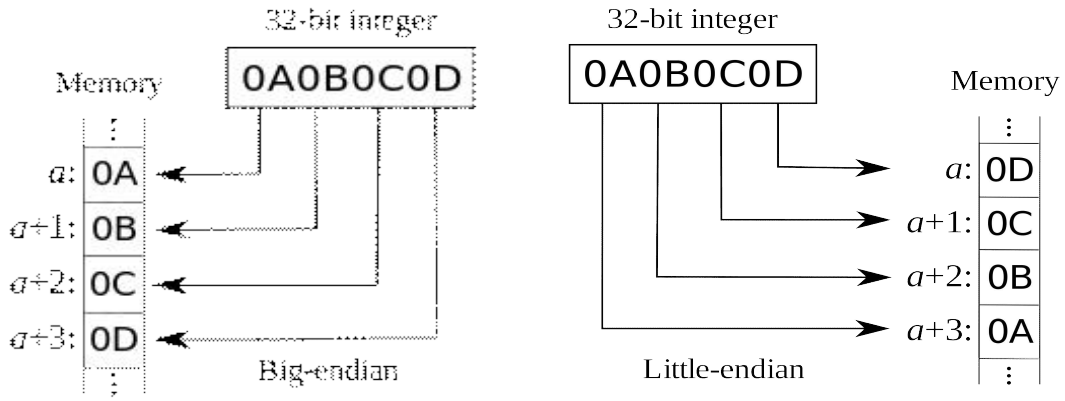

그림 12 리틀 엔디안과 빅 엔디안

eBPF/XDP 커널 공간 프로그램이 수신된 네트워크 패킷들이 프로그램에서 고려한 프로토콜인 이더넷과 IPv4 프로토콜을 갖는 것을 확인한 후, eBPF/XDP 커널 공간 프로그램은 IP 헤더 분석을 위하여 리눅스 헤더인 'ip.h'에 포함된 'iphdr' 구조체 포인터를 선언한다. 선언된 구조체의 포인터는 IPv4 헤더가 시작되는 주소를 가리킨다. eBPF/XDP 커널 공간 프로그램은 수신된 IP 패킷의 길이가 완전한 IP 헤더를 포함 가능한지 검사한다. IPv4 헤더 이후의 첫 번째 바이트 값의 주소가 수신된 IP 패킷의 첨단 이후 첫 주소 바이트보다 크다면, 현재 eBPF/XDP 커널 공간 프로그램이 분석 중인 수신된 IP 패킷은 정상적인 IPv4 패킷보다 짧은 것으로 판정한다. eBPF/XDP 커널 공간 프로그램은 정상적인 IPv4 헤더 포함이 불가능한 짧은 네트워크 패킷을 누락한다. 누락되지 않은 네트워크 패킷은 정상적인 IPv4 헤더로 간주하며 송신 IPv4 주소 확인을 위하여 'ip.h' 구조체 내의 'saddr' 변수에 저장된 값을 확인한다.

```
95  #endif
96          __u8    tos;
97          __be16  tot_len;
98          __be16  id;
99          __be16  frag_off;
100         __u8    ttl;
101         __u8    protocol;
102         __sum16 check;
103         __be32  saddr;
104         __be32  daddr;
105         /*The options start here. */
106 };
```

그림 13 IPv4 헤더 내용

eBPF/XDP 커널 공간 프로그램은 누락되지 않은 네트워크 패킷의 IPv4 송신 주소 값을 eBPF map에 저장된 필터링 리스트의 값들과 비교한다. eBPF/XDP 커널 공간 프로그램은 특정 eBPF map을 필터링 리스트로 이용하며, 해당 리스트에 필터링할 IP 송신 주소들을 명시한다. 분석 중인 네트워크 패킷의 IPv4 송신 주소 값이 DoS 공격 트래픽을 전송하는 기기의 IPv4 주소값과 일치한다면 해당 네트워크 패킷은 누락된다. 이 시점까지 누락되지 않은 패킷은 정상적인 사용자가 eBPF/XDP 커널 공간 프로그램이 설치된 물리 서버에서 작동 중인 서비스의 이용을 위하여 접근한 것으로 판단하여 누락시키지 않는다. 정상적인 사용자의 트래픽으로 분류된 네트워크 패킷의 IPv4 주소 값은 정상적으로 수신된 네트워크 패킷의 IPv4 송신 주소 값을 저장하는 eBPF map에 저장되며, 해당 송신 IPv4 주소 값과 페어링 된 수신된 네트워크 패킷의 수를 저장하는 eBPF map의 숫자의 값을 증가시킨다.

2.4. eBPF/XDP 사용자 공간 프로그램

eBPF/XDP 사용자 공간 프로그램은 BCC(BPF Compiler Collection)을 이용하여 Python으로 작성되었으며, Onion-Ring 가시화 프로그램과의 통신을 위하여 Kafka를 이용한다. BCC는 eBPF를 이용한 커널 트레이싱을 위한 툴 키트이다. BCC를 이용하여 eBPF 프로그램을 더 쉽게 만들 수 있으며, eBPF 프로그램의 front-end를 lua 및 python으로 만들 수 있도록 도와준다. eBPF/XDP 프로그램 실행 전, Kafka의 zoo keeper, broker, 그리고 해당 zoo keeper와 broker가 이용할 topic이 생성된 상태에서 프로그램이 실행되어야 한다. Kafka는 분산 스트리밍 플랫폼으로서, 메시지 큐와 비슷한 레코드 스트림을 이용, 실시간 데이터를 전송하는 오픈 소스 프로그램이다 [4]. 카프카는 클러스터의 형태로서 다수의 서버를 이용 가능하며, 본 프로그램의 성능 검증을 위한 실험 시 컨테이너화된 3개의 카프카 서버를 클러스터로 이용하였다. 카프카의 코어 API는 총 4개이며 표 3과 같다.

코어 API	역할
Producer	에플리케이션이 Kafka topic 들에 대한 레코드 스트림을 제공 시 사용
Consumer	에플리케이션이 topic들을 구독하여 레코드 스트림을 처리 시 사용
Streams	에플리케이션이 스트림 처리기로서 동작할 수 있게 함
Connector	Producer, Consumer의 생성 및 작동에 관여

표 3. Kafka의 코어 API들

eBPF/XDP 사용자 공간 프로그램은 eBPF/XDP 커널 공간 프로그램이 eBPF map에 저장한 수신 네트워크 패킷의 IPv4 송신 주소 값을 읽는다. eBPF/XDP 커널 공간 프로그램이 eBPF map에 저장한 수신된 네트워크 패킷의 IPv4 송신 주소 값은 사람들이 보편적으로 사용하는 IP 주소(i.e., 192.168.1.1)의 형식이 아닌 기계와 더 가까운 주소 값(i.e. 14145922)의 형식을 갖는다. eBPF/XDP 사용자 공간 프로그램은 기계와 가까운 주소값을 사람과 더 가까운 형태의 IP 주소로 변환한다. eBPF/XDP 사용자 공간 프로그램은 캐릭터 스트링으로 저장된 IP 주소값을 정수형으로 변환한 후 해당 정수를 이진법으로 변환, 캐릭터 스트링의 형태로 저장한다. 해당 캐릭터 스트링의 길이가 24일 경우 4개의 0을 스트링의 앞부분에 붙여 스트링의 길이를 28로 연장한다. 길이 28의 캐릭터 스트링은 길이 8의 캐릭터 스트링으로 분리되며, 나누어진 4개의 캐릭터 스트링은 반대의 순서로 저장된다. 나누어진 4개의 캐릭터 스트링은 각각 10진수로 변환된 후 각각의 캐릭터 스트링들 사이에 점을 더하여 하나의 스트링으로 합쳐진다.

eBPF/XDP 사용자 공간 프로그램은 eBPF/XDP 커널 공간 프로그램이 eBPF map에 저장한 수신된 패킷의 개수를 불러와 사람과 친밀한 형태의 IPv4 주소로 변환된 해당 네트워크 패킷의 IPv4 송신 주소와 함께 저장한다. 저장된 IPv4 송신 주소와 해당 주소의 수신 네트워크 패킷 개수는 하나의 세트로 구성되어 Onion-Ring 가시화 프로그램에 Kafka를 이용하여 5초 간격으로 전송된다.

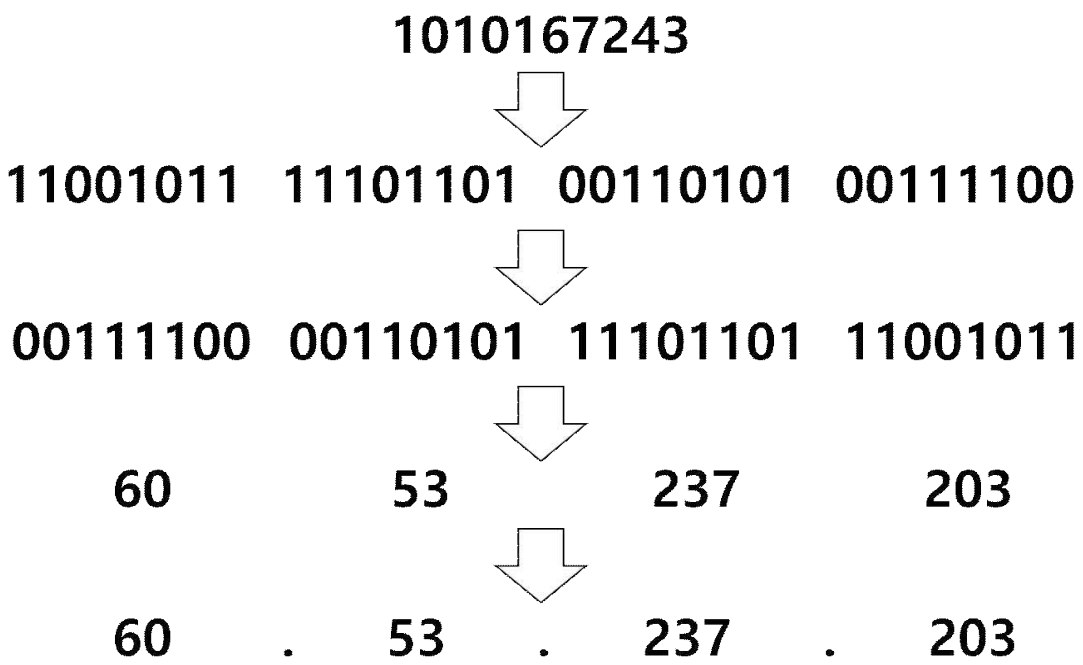

그림 14 IP 주소의 변환 과정

2.5. Onion-Ring 가시화 프로그램

Onion-Ring 가시화 프로그램은 Python을 기반으로 작성되었다. Onion-Ring 가시화 프로그램은 Python 그래프 라이브러리 중 하나인 'Plotly'의 'Sun Burst Chart'를 이용한다 [21]. 'Plotly'는 scatter plots, line charts, bar charts, pie charts, error bars, heatmaps 등 여러 종류의 그래프를 지원하는 라이브러리이다. 'Plotly'가 지원하는 그래프들 중 하나인 'Sun Burst Chart'는 여러 계층의 링을 이용하여 조직도를 가시화하는 그래프이다. 'Sun Burst Chart'를 이용한 Onion-Ring 가시화 프로그램은 물리 서버와 해당 물리 서버가 보유한 네트워크 인터페이스를 가시화한다. 또한, Onion-Ring 가시화 프로그램은 'Sun Burst Chart'의 사용자와의 상호작용 기능을 이용하여 물리 서버와 물리 서버의 각 네트워크 인터페이스의 네트워크 트래픽 상태를 FireFox 및 Chrome과 같은 웹 브라우저로 가시화한다.

eBPF/XDP 사용자 공간 프로그램으로부터 전송받은 데이터를 기반으로 물리 서버의 네트워크 인터페이스의 네트워크 트래픽 상태를 가시화한다. eBPF/XDP 사용자 공간 프로그램으로부터 전송된 데이터는 캐릭터 스트링이며 수신된 네트워크 트래픽의 IPv4 송신 주소와 해당 IPv4 송신 주소에서 전송된 패킷의 개수가 단일 공백으로 구분되어 있다. Onion-Ring 가시화 프로그램은 eBPF/XDP 사용자 공간으로부터 전송된 데이터 내에 포함된 IPv4 송신 주소와 전송된 패킷의 개수를 나누어 저장한다.

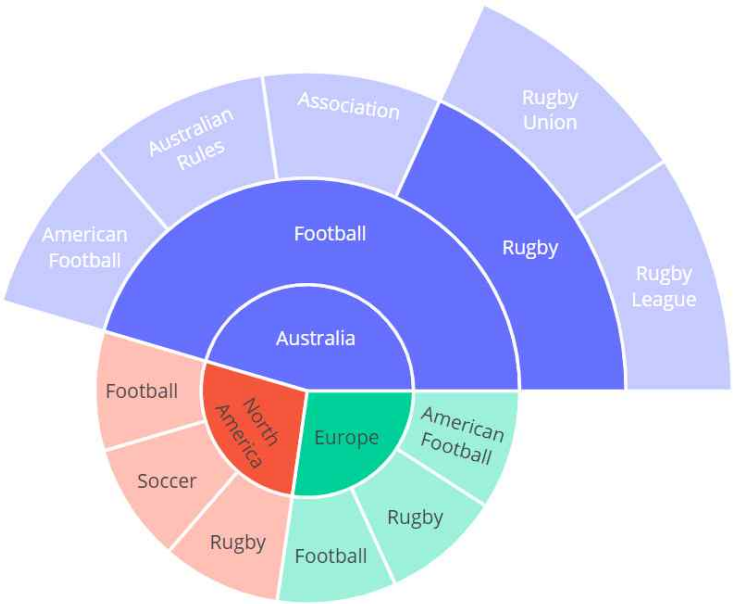

그림 15 Ploly Sun Burst Chart

Onion-Ring 가시화 프로그램은 1초간 수신된 패킷의 개수에 따라 해당 네트워크 인터페이스에 해당하는 그래프의 색을 변환한다. 그래프 색에 대한 정책은 표 4와 같다.

색	수신 패킷 개수
회색	0
노란 녹색	1-700
연한 녹색	701-1400
녹색	1401-2100
적색	2101-2800

표 4 Onion-Ring 가시화 프로그램의 네트워크 패킷 가시화 정책

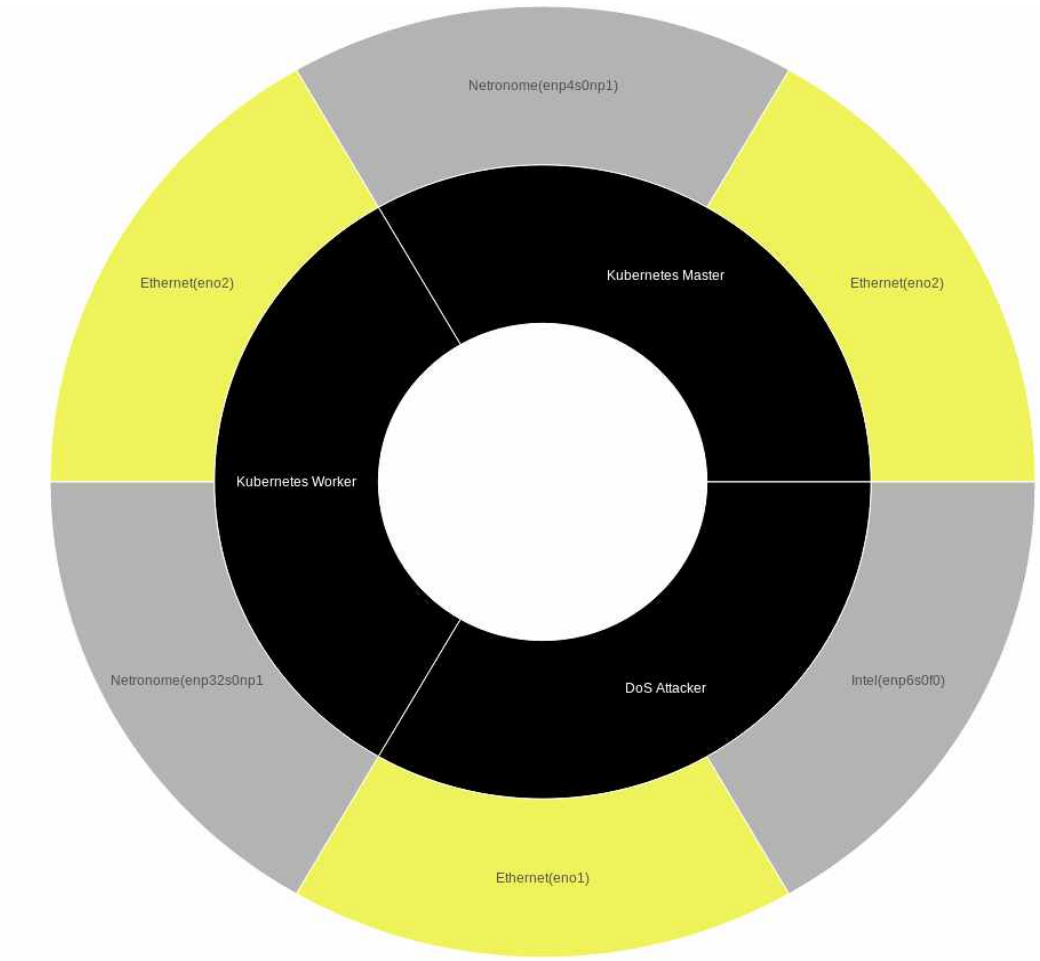

그림 16 실험 환경의 Onion-Ring 가시화

Onion-Ring 가시화 프로그램은 수신된 네트워크 패킷들의 IPv4 송신 주소를 확인하는 방법을 제공한다. Onion-Ring 가시화 프로그램의 네트워크 인터페이스 그래프 위에 마우스를 약 2초간 정지시킬 시, 해당 네트워크 인터페이스에서 수신 중인 네

트워크 패킷들의 IPv4 송신 주소가 나열된다. Onion-Ring 가시화 프로그램은 수신 중인 패킷의 개수와 해당 패킷들의 IPv4 송신 주소를 터미널에 출력한다. 해당 출력물은 본 문서의 실험 결과 부분에서 확인할 수 있다.

2.6. eBPF/XDP 기반 동적 패킷 모니터링 및 필터링 프로그램의 패킷 필터링 성능 검증

eBPF/XDP 기반 동적 패킷 모니터링 및 필터링 프로그램의 패킷 필터링 성능을 검증하기 위하여 그림 17과 같은 실험 환경을 구성한다.

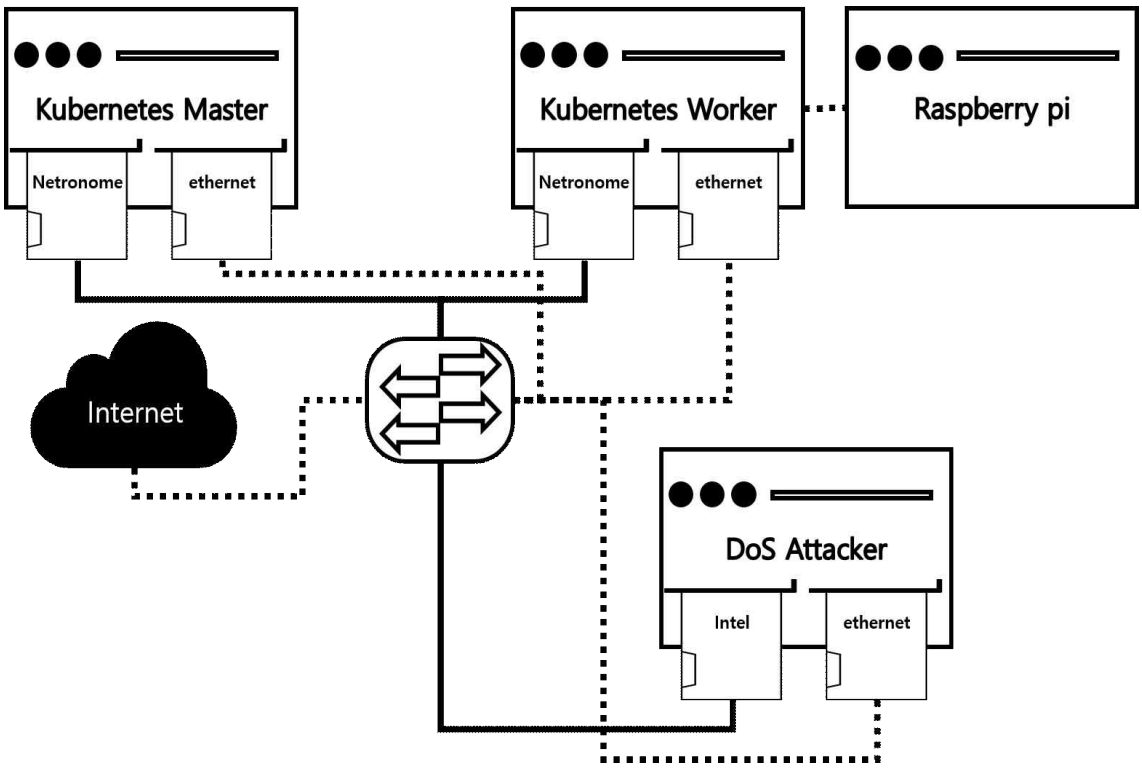

그림 17 eBPF/XDP 기반 DoS 공격 패킷 필터링 및 모니터링 실험 환경

실험을 위하여 총 세 대의 물리 서버(Kubernetes Master, DoS Attacker, 그리고 Kubernetes Worker)와 한 대의 라즈베리 파이(Kubernetes Worker)가 사용되었다. 라즈베리 파이는 Kubernetes Worker에 연결되어 Kubernetes Worker에게 자신에게 연결된 온,습도 센서의 데이터를 전송한다. 세 물리 서버는 각각 두기의 NIC(Network Interface Card)을 장착하였으며, 하나는 실험 환경을 구성하는 기기 간의 통신을 위하여, 다른 하나는 외부의 인터넷과 통신하기 위하여 사용된다. 기기들의 네트워크 인터페이스들은 모두 단일 스위치를 거쳐 실험 환경 네트워크 내부의 기기들 또는 외부의 인터넷과 연결된다. 물리 서버 Kubernetes Worker에서는

Kubernetes orchestration을 이용한 microservices architecture 기반 IoT-cloud service인 Smart Energy IoT-Cloud service가 실행 중이다 [5]. DoS Attacker는 HTTP flood DoS 공격툴인 Goldeneye를 이용, DoS 공격 트래픽을 Kubernetes Worker에게 전송한다. Kubernetes Worker는 eBPF/XDP hardware offload를 지원하는 Netronome사의 NIC을 이용하여 DoS Attacker의 DoS 공격 트래픽을 수신한다. 실험은 총 3회 진행된다. 1회차 실험은 DoS 공격 트래픽을 전송하지 않은 상태에서 Kubernetes Worker의 호스트 CPU 사용률을 측정한다. 2회차 실험은 DoS 공격 트래픽을 전송하며 Kubernetes Worker의 호스트 CPU 사용률을 측정한다. 이때 DoS 공격 트래픽은 필터링하지 않는다. 3회차 실험은 DoS 공격 트래픽을 전송하며 Kubernetes Worker에서 eBPF/XDP hardware offload를 이용, DoS 공격 트래픽을 필터링하며 kubernetes Worker의 CPU 사용률을 측정한다. DoS 공격은 평균 1.03 MBps, 2320 pps의 공격 트래픽을 생성한다. 실험 결과는 표5 와 그림 18과 같다.

회차	DoS 공격	eBPF/XDP	평균 CPU 사용률(%)	최대 CPU 사용률(%)	최소 CPU 사용률(%)
1	X	X	2.24561	8.63	0.68
2	O	X	17.9154	23.62	15.94
3	O	O	2.44902	4.35	1.02

표 5 회차별 실험 결과

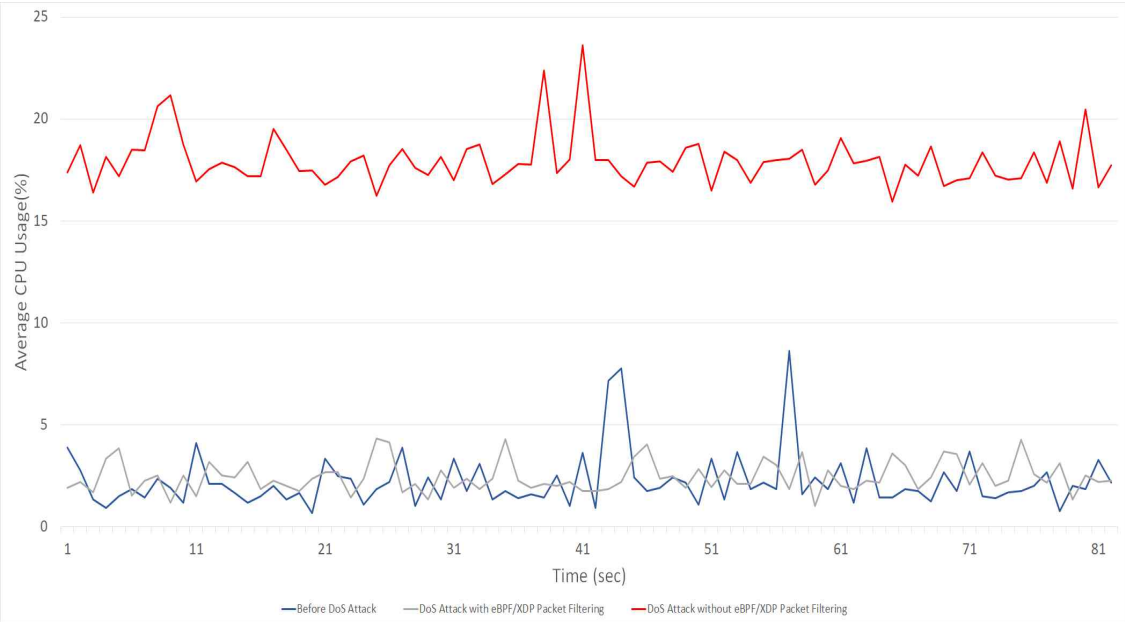

그림 18 시간에 대한 평균 CPU 사용률 그래프

실험 결과 DoS 공격이 진행될 시, 평균 CPU 사용률은 17.9154%로, DoS 공격이 없는 상태 대비 약 8배 정도 CPU 사용률이 증가함을 확인하였다. DoS 공격 트래픽을 eBPF/XDP hardware offload를 이용하여 필터링 시, DoS 공격 트래픽이 없는 경우 대비 약 0.02% 의 host CPU 사용률 증가를 확인하였다. 2회차 실험 시, 필터링 되지 않은 DoS 공격 트래픽이 Kubernetes Worker의 네트워크 인터페이스에 수신되는 것을 Onion-Ring 가시화 프로그램을 이용하여 확인할 수 있으며, 그 결과는 그림 19와 그림 20과 같다.

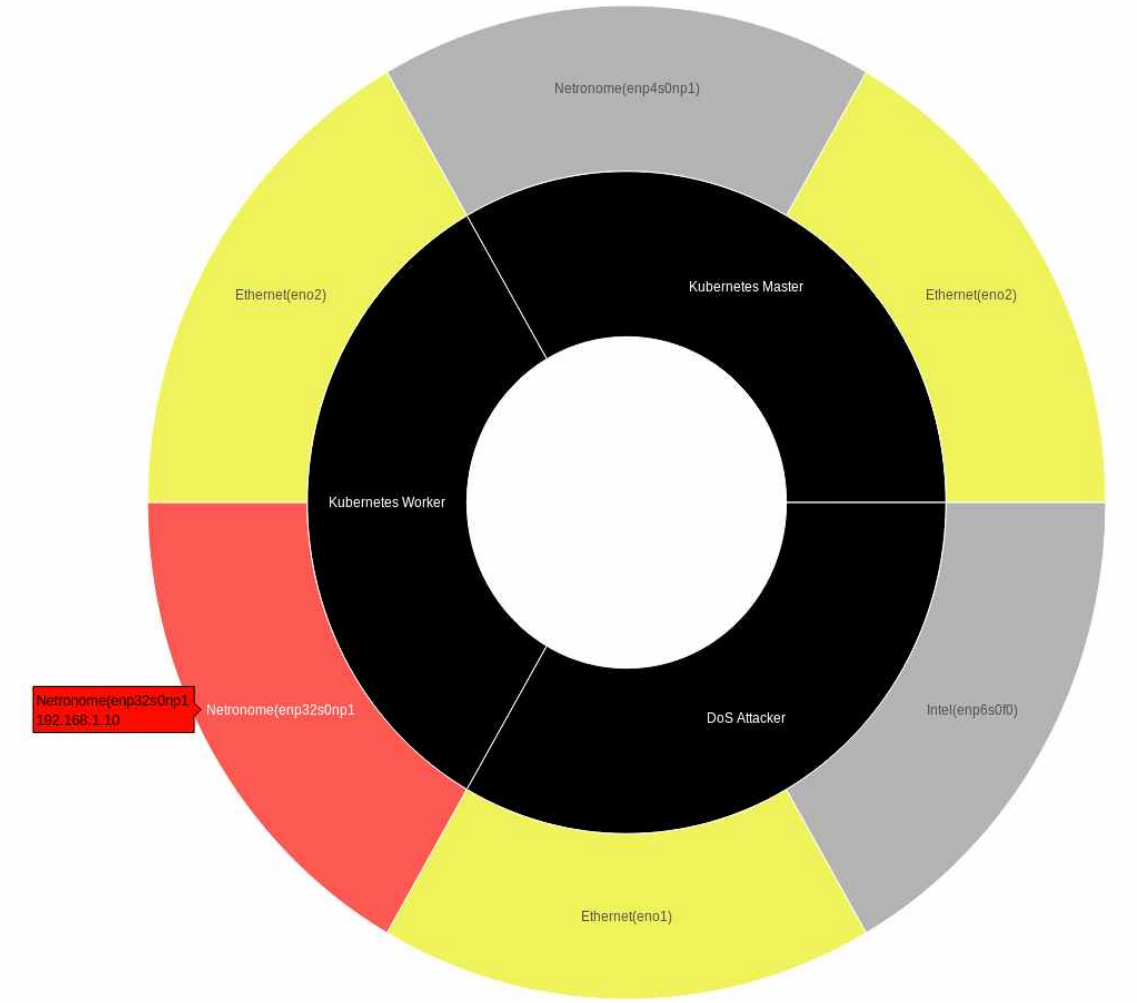

그림 19 DoS 공격 트래픽을 수신중인 Kubernetes Worker

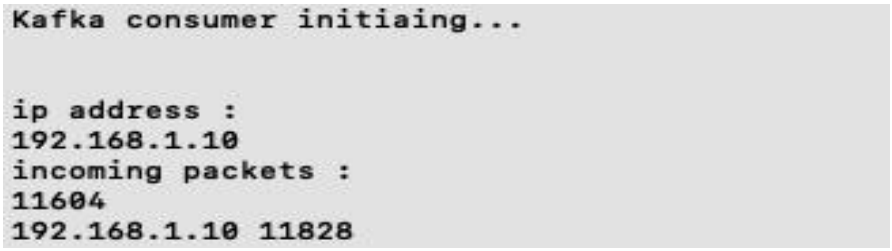

그림 20 Onion-Ring 가시화 프로그램의 터미널 출력물

DoS 공격 트래픽을 수신 중인 네트워크 인터페이스 위에 DoS 공격 트래픽을 수신 중인 DoS Attacker의 private IP 주소가 명시되어 있음을 확인할 수 있다. Onion-Ring 가시화 프로그램의 터미널 출력물을 통하여 5초간 11604개의 패킷이 192.168.1.10의 IPv4 송신 주소를 갖는 서버로부터 전송되었음을 확인할 수 있다. 또한, 그림 16과 비교하여 공격을 수신 중인 물리 서버의 네트워크 인터페이스, Netronome 이 회색에서 적색으로 변환된 것을 확인할 수 있다.

3. 결론

eBPF/XDP 기반 동적 패킷 모니터링, 필터링 프로그램은 더 많은 수신 IP에 대한 값들을 처리할 수 있도록 발전시킬 계획이다. 현재의 프로그램은 수신되는 패킷들에 대하여 한정된 수의 IP 주소에 대해서만 처리할 수 있다. 해당 기능은 eBPF map의 크기를 증가시키고, eBPF/XDP 사용자 공간 프로그램에서 수집되는 IP 주소를 처리하는 기능을 구현할 예정이다. Onion-Ring 가시화 프로그램은 3차원 가시화 프로그램으로 확장할 계획이다. 수신되는 네트워크 패킷의 네트워크 레이어 별로 수집한 뒤, 각 레이어에 대하여 Onion-Ring 가시화를 진행하고 각 레이어 별 Onion-Ring을 모아 3차원으로 가시화를 진행할 예정이다.

References

[1] SHIN, S., SONG, Y., LEE, T., LEE, S., CHUNG, J., PORRAS, P., YEGNESWARAN, V., NOH, J., AND KANG, B. B. Rosemary: A robust, secure, and high-performance network operating system. In Proceedings of the 21th ACM Conference on Computer and Communications Security (CCS'14)

[2] Cloudflare, "DNS Amplification Attack," [Online]. https://cloufdlfare.com/learning/ddos/dns-amplification-ddos-attack/

[3] Cilium, "Why is the kernel community replacing iptables with BPF?," [Online]. https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables[

[4] Kafka, "Introduction," [Online]. https://kafka.apache.org/intro

[5] Lee, Seunghyung, et al. "Relocatable Service Composition based on Microservice Architecture for Cloud-Native IoT-Cloud Services." Proceedings of the Asia-Pacific Advanced Network 48: 23-27

[6] Netscout, "DDoS Protection 101:What is DDoS? Volmuetric DDosS Attacks," [Online]. https://www.netscout.com/what-is-ddos/volumetric-attacks

[7] Netscout, "ICMP Flood Attacks," [Online]. https://www.netscout.com/what-is-ddos/icmp-flood

[8] T. Fu and T.-S. Chou, ৷৷An analysis of packet fragmentation attacks vs. snort intrusion detection system, International Journal of Computer Engineering Science (IJCES), 2012.

[9] F. Lau, SH Rubin, MH Smith, and L. Trajkovic. Distributed denial of service attacks. In 2000 IEEE International Conference on Systems, Man, and Cybernetics, volume 3, 2000.

[10] Y. Xie and S.-Z. Yu. A Novel Model for Detecting Application Layer DDoS Attacks. In First International Multi-Symposiums on Computer and Computational Sciences., volume 2, pages 56–63, June 2006.

[11] Butler, K., Farley, T.R., McDaniel, P., Rexford, J.: 'A survey of BGP security issues and solutions', Proc. IEEE, 2010, 98, (1), pp. 100–122

[12] D. Moustis and P. Kotzanikolaou, "Evaluating security controls against HTTP-based DDoS attacks," Systems and Applications (IISA), 2013 Fourth International Conference on Information, Intelligence,, 2013.

[13] M. Zolotukhin, T. Hämäläinen, T. Kokkonen and J. Siltanen, "Increasing web service availability by detecting application-layer DDoS attacks in encrypted traffic," 2016 23rd International Conference on Telecommunications (ICT), Thessaloniki, 2016.

[14] W. -Z Lu, S. –Zg. Yu, "An HTTP Flooding Detection Method Based on

Browser Behavior," International Conference on Computational Intelligence and Security, Volume 2, November 3-6, 2006, pp.1151 - 1154.
[15] Netronome, "FRnOG 30 : Faster Networking à la française," [Online]. https://www.netronome.com/blog/frnog-30-faster-networking-la-francaise/
[16] Github, "goldeneye," [Online]. https://github.com/jseidl/GoldenEye
[17] Brendangregg.com, "Linux Extended BPF (eBPF) Tracing Tools," [Online]. http://www.brendangregg.com/ebpf.html
[18] Cilium, "BPF and XDP Reference Guide," [Online]. https://docs.cilium.io/en/v1.6/bpf/
[19] Wikipedia, "Endianness," [Online]. https://en.wikipedia.org/wiki/Endianness
[20] Netronome, "Open Source Packet Filtering: eBPF at FOSDEM'19," [Online]. https://www.netronome.com/blog/open-source-packet-filtering-bpf-fosdem19/
[21] Plotly, "Sunburst Charts in Python," [Online]. https://plot.ly/python/sunburst-charts/

K-ONE 기술 문서

- 이 문서의 기술적인 내용은 프로젝트의 진행과 함께 별도의 예고 없이 변경될 수 있습니다.

- 본 문서와 관련된 문의 사항은 아래의 정보를 참조하시길 바랍니다.
  (Homepage: http://opennetworking.kr/projects/k-one-collaboration-project/wiki, E-mail: k1@opennetworking.kr)

작성기관: K-ONE Consortium
작성년월: 2019/11