SDN을 활용한 네트워크 검역시 패킷캡쳐 기능 개선 방안

송명욱*, 정준권*, 정태명**
*성균관대학교 전자전기컴퓨터공학과
**성균관대학교 정보통신대학
e-mail:*{mwsong, jkjung}@imtl.skku.ac.kr, **tmchung@ece.skku.edu

Improvement of Packet Capture in Network Quarantane using SDN

Myeong-Uk Song*, Jun-Kwon Jung*, Tai-Myoung Chung**
*Dept. of Electrical and Computer Engineering, SungKyunKwan University
**College of Information and Communication Engineering, Sungkyunkwan University

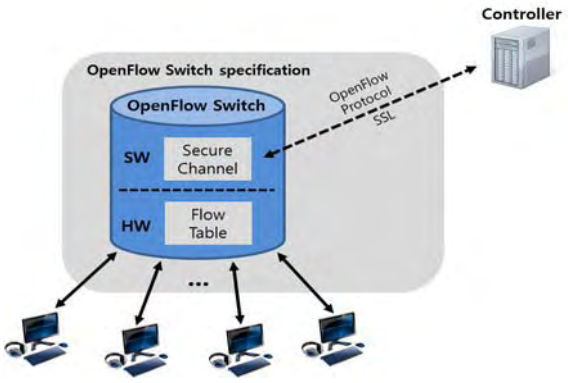
요 약

패킷 캡쳐는 IDS 및 IPS에서 가장 중요한 개념이다. 악성 패킷에 대한 시그니쳐를 탐지하여 사전에 차단할 수 있기 때문이다. OpenFlow를 이용하여 네트워크 패킷 요청 혹은 응답을 특화된 서버, 즉 인터넷검역소를 거친 후 종단 사용자에게 전달할 수 있다. SDN의 특성을 활용하여 종단 사용자는 어떤 프로그램도 설치하지 않고도 네트워크에 연결되어 있는 것만으로 가장 빠른 보안을 적용받을 수 있다. 본 논문에서는 SDN상에서 네트워크 검역을 위해 오픈 소스 Bro IDS를 이용하여 패킷을 캡쳐하는 방법과 발생한 문제와 그에 대한 해결법을 제안한다.

1. 서론

최근 클라우드 서버, 대용량 데이터 서버와 같은 대규모 트래픽이 발생하는 네트워크에서 패킷을 flow 단위로 제어하고 장비들이 논리적으로 분리하여 서비스 활용에 따른 비용을 감축하기 위해 소프트웨어-정의 네트워킹(Software-Defined Networking: SDN)을 사용하는 기업들이 늘어나고 있다.

SDN은 (그림 1)과 같이 컨트롤러, SDN 스위치로 구성되어 있으며, 일반적으로 포워딩 장치 간의 통신을 위해 OpenFlow라는 프로토콜을 사용한다.



(그림 1) SDN의 구성[1]

기존 네트워크에서는 네트워크 공격을 탐지하기 위해 방화벽(Firewall)이나 침입 탐지 시스템(Intrusion Detection Systems: IDS), 침입 차단 시스템(Intrusion Prevention Systems: IPS) 등을 사용하는데, 이러한 네트워크 공격 탐지 시스템들은 네트워크의 경계에 위치해야 하는 입지 결정 문제 (Location Problem)가 존재한다. 하지만 SDN에서 네트워크 공격 탐지 시스템을 구축할 경우, 시스템이 네트워크 경계에 존재하지 않더라도 OpenFlow와 Software-Define 기능을 이용하여 유입되는 모든 패킷을 수신할 수 있다는 큰 장점을 가진다.

본 논문에서는 네트워크로 송수신되는 패킷을 검사하고 유해 패킷을 사전에 차단하며, 검사 후 유해/정상 판정이 어려운 패킷에 대해 tagging을 하여 본래의 목적지로 포워딩하는 기능을 가진 침입 차단 시스템을 설계 및 구현한다. 또한 시스템 구현에 있어 TCP 패킷의 재조립 및 포워딩 과정에서 발생하는 문제에 대해 다루고 그 해결법에 대해 기술한다.

본 논문은 다음과 같이 구성된다. 2장에서는 시스템 구현을 위한 환경 구성을 설명하는 장이며, 3장에서는 구현상의 문제를 분류하고 분석한 결과에 따른 해결법을 기술한다. 마지막으로 4장에서는 본 시스템의 과제 및 발전 방향과 발생할 수 있는 문제점을 예상하고 해결 방법을 간략하게 기술하여 향후 SDN을 이용한 침입 차단 시스템을 연구하고 개발하는 데에 기틀을 마련한다.

2. 환경 구성

이 장에서는 네트워크 공격 탐지를 위한 환경 구성에 대해 기술한다.

2.1 SDN 환경

SDN에서 네트워크 관리자는 컨트롤러에 rule을 생성하고 OpenFlow 스위치에 rule을 적용한다. 이후 포워딩 장치마다 존재하는 흐름 테이블이 rule에 의해 업데이트되고, 모든 패킷의 흐름을 제어할 수 있게 된다[2].

이러한 SDN의 특성을 이용하면 네트워크로 유입되는 모든 패킷이 특화된 서버(Featured Server)를 경유하게 할 수 있는데, 본 논문에서는 이 서버를 검역소로 칭하고, 이 검역소는 IDS를 기반으로 구축한다.

논문에서 SDN으로 구성한 IDS는 다음과 같은 세 가지 장점을 가진다. 첫째로 IDS는 외부로부터 수신하는 incoming 패킷뿐만 아니라, 외부로 송신되는 outgoing 패킷까지 검역을 수행한다. 이 때문에 네트워크 내에 존재하는 종단 사용자는 요청 및 응답 패킷 모두에 대해 보안 위협으로부터 안전을 보장받을 수 있다. 둘째로 종단 사용자의 서버나 호스트에 유해 패킷 탐지를 위한 클라이언트를 설치할 필요가 없다. 셋째로 검역소에서 한번 유해한 패킷으로 판단이 될 경우 동일한 네트워크 내에 있는 사용자들에게 전파·전염될 우려가 없다[3].

2.2. Bro IDS

IDS로 가장 많이 사용되고 유명한 프로그램은 Snort이다. Snort는 패킷 스니핑(Sniffing), 로깅(Logging)과 책정된 rule의 시그니처 매핑으로 IDS의 역할을 수행한다. 이런 룰은 기본적으로도 제공되지만, 자체적인 룰을 만들어 패턴을 적용할 수 있게 함으로써 유저 커스터마이즈를 지원하고 있다[4].

더 많은 유해 패킷의 탐지를 위해 Snort 사용자들은 커뮤니티를 이루었다. 메일링 리스트와 블로그 등 여러 가지 형태로 Snort의 rule이 공유되고 있고, 그 수는 계속 증가하고 있다.

하지만, 기본적으로 제공되는 Snort의 최신 rule은 유료로 제공되며, 소스를 수정하여 컴파일하여 사용할 경우에는 GPL 라이선스에 따라 프로그램의 소스코드를 동봉하거나, 소스를 무료로 공개 배포해야만 한다.

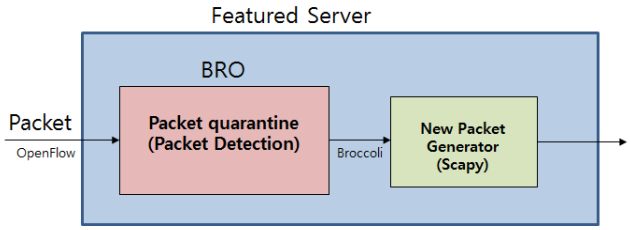
1995년, 버클리 대학에서는 Bro라는 이름의 IDS를 개발했다. Snort와 같이 자체적인 룰을 적용하여 IDS 기능을 수행할 수 있으며, 소스 변형 및 재배포에 아무런 제약이 없어 상업적 활용이 가능하다[5].

다른 이유로 Bro를 사용하는 이유는, 한 개의 특화된 서버로만 시스템을 구축시 성능상의 문제가 생길 수 있기 때문이다. 큰 규모의 네트워크에 한 개의 검역소로만 시스템을 구축시 많은양의 패킷을 처리하는데 지연시간이 발생할 수 있고, 여기에서 발생하는 지연시간은 결국 종단 사용자의 서비스 이용에도 영향을 미치게 된다. Bro는 자체적으로 클러스터 기능을 지원하여 여러 서버에서 패킷을 검사할 수 있게 하고 서버의 상태를 관리할 수 있는 기능을 제공하므로, 부하에 대한 내성을 갖출 수 있다.

Bro는 자체적인 rule 구성을 위해 .bro 라는 확장자를 가진 언어를 지원하고 있고, 이를 이용해 Bro의 rule과 동작 형태를 원하는 데로 설정할 수 있다. 하지만, 네트워크 관리자는 Bro가 제공하는 기능보다 더 많은 기능을 필요로 할 수도 있으며, 시스템 구현시 새로운 언어보다 사용자에게 더 친숙한 언어로 시스템을 구현하는 것을 선호한다.

이에 Bro는 커뮤니케이션 모듈(The Bro Client Communications Library: Broccoli)을 제공한다. 커뮤니케이션 모듈은 현재 Perl, Python, Ruby 등의 언어를 지원하며, Bro가 스니핑하는 정보들을 외부 환경에서 접근할 수 있게 된다. 본 논문에서는 Python을 기준으로 하였으며, 스니핑 대상은 TCP 및 HTTP 프로토콜을 기준으로 하였다. 이는 OpenFlow의 특성을 이용하여 모든 패킷을 대상으로 하는 것이 아니라, TCP 프로토콜에 해당하는 패킷만을 특화된 서버로 보내므로 어떤 프로토콜의 패킷이 들어올지 고려하지 않아도 된다는 것을 의미한다.

(그림 2)는 Bro를 사용한 네트워크 검역체계를 나타낸다. 이와 같이 구성된 네트워크 검역소의 IDS에는 OpenFlow에 의해 최종 목적지가 검역소로 설정된 패킷만이 남게 되는데, 종단 사용자는 유해한 패킷이 아닐 경우 온전한 패킷을 수신해야 정상적인 서비스를 받을 수 있다. 때문에, 외부 모듈에서는 종단 사용자에게 전달될 패킷을 새롭게 생성해 보내야만 한다. 이때, Bro의 기능만으로는 패킷을 생성하거나 포워딩할 수 없으므로 Scapy라는 툴을 사용한다[6].



(그림 2) 네트워크 검역체계

Scapy는 Python 기반의 패킷 툴로서, TCP, UDP, ICMP 프로토콜에 맞추어 패킷을 임의로 생성해 전송할 수 있다. 패킷 검사 후, 정상 패킷은 Scapy를 이용하여 새로운 패킷을 생성해 종단 사용자에게 보내고, 유해한 패킷일 경우에는 패킷을 생성하지 않음으로 패킷을 폐기할 수 있다.

3. 패킷 캡쳐 기능 개선

이 장에서는 Bro를 이용하여 시스템을 구축하며 발생했던 문제점을 구분하고, 그에 대한 문제 해결법을 기술한다.

3.1 파라미터

Bro에서 캡쳐한 TCP 프로토콜 패킷은 다시 외부 모듈로 전달되는 과정이 필요하다. 이는 Bro에서 지원하는 커

뮤니케이션 모듈인 Broccoli를 사용하여 외부 모듈로 정보를 전달할 수 있다. Broccoli는 Bro에 inject 하는 방식으로 Bro가 캡쳐하는 정보를 Bro가 아닌 다른 외부 모듈에서 획득하여 사용할 수 있으며, 이 모듈을 이용하여 Bro의 tcp_packet 이벤트가 패킷 캡쳐 후 파싱 하는 패킷의 출발지와 목적지 IP, 포트 번호, TCP flag를 알 수 있다. 이를 이용하여 Scapy로 flag 옵션을 설정해주고 TCP 패킷을 다시 생성하면 새로운 패킷을 생성해 보낼 수 있다. <표 1>은 Scapy를 이용한 TCP 파라미터 설정의 의사 코드를 나타낸다.

```
scapy.TCP(sport=srcPort,dport=dstPort,
flags=flags,seq=int(ack))
```

<표 1> Scapy를 이용한 TCP 파라미터 설정

수신한 파라미터는 순서를 변경해 사용해야 한다. Bro가 수신한 패킷은 다시 포워딩되어야 하는 패킷이므로, 목적지와 대상 발신지의 순서를 모두 반대로 변경해야 한다. 즉, Bro가 수신한 sequence number는 acknowledge number로 사용되어야 하며, acknowledge number는 sequence number로 사용하도록 한다.

또한, tcp_packet 이벤트에서 받아오는 sequence number와 acknowledge number는 패킷의 상대적인 값을 반환하기 때문에 0과 1의 값밖에 받을 수 없어 올바른 패킷을 생성할 수 없다.

이에 대한 문제의 해결 방법으로, Bro 스크립트에서 tcp_packet에서 반환되는 seq, ack 파라미터 값 대신, get_orig_seq 함수와 get_resp_seq 함수를 이용하여 패킷의 실제 값을 리턴 받아 사용할 수 있어야 한다. <표 2>는 .bro에서 리턴 받아 처리되는 파라미터의 예시를 보여준다.

```
pkt_generate(get_orig_seq(c$id),
get_resp_seq(c$id),contents);
```

<표 2> Bro에서 리턴받아 처리하는 파라미터 예시

3.2 독립된 프로토콜

Bro는 TCP 패킷과, 미리 정의된 잘 알려진 포트(Well-Known Port)를 이용한 서비스 패킷을 분리하여 이벤트를 발생시킨다. 대표적인 예로, HTTP 프로토콜 패킷은 일반적인 TCP 프로토콜의 이벤트로는 처리할 수 없다. HTTP 프로토콜과 같이 Bro에서 따로 이벤트로 정의하고 있는 프로토콜의 경우, 모두 프로토콜 이벤트를 정의해주어야 한다. 예를 들어 <표 3>과같이 HTTP 패킷을 의미하는 80포트의 TCP 패킷은 하나의 패킷을 완성하기 위하여 http_request, http_hdr, http_post_data, http_msg 4개 함수를 정의 및 선언하여 리턴되는 sequence number에 맞춰 패킷을 조립해야 한다.

```
@event
def frag_http_request(method,URI,version):
 #http request type, URI, method, HTTP version
@event
def frag_http_hdr(c, uid,key,data):
 #http header를 저장
@event
def frag_http_post_data(uid,data):
 #http post data가 있을 경우 처리
@event
def fin_http_msg(connection):
 #http 메시지를 저장
def assemble_http_pkt(c):
```

<표 3> Broccoli HTTP packet assemble 의사 코드

하나의 이벤트만 정의하면 되는 tcp_packet 이벤트와 달리 더 많은 과정이 필요하며, Bro에서 정의하고 있는 프로토콜은 모두 외부 모듈에서 따로 정의하여 패킷을 조립해야 하는 과정을 거쳐야 한다. 그리고 Bro 자체의 버그로 인해, SMTP와 같은 프로토콜은 이벤트를 정의하였더라도 제대로 이벤트 호출이 이루어지지 않는 버그도 확인할 수 있었다.

외부 모듈에서 프로토콜별로 모두 정의해 처리하게 되면 프로그램의 코드가 직관적이지 않게 되는 것은 물론, 정의하지 않은 프로토콜에 대한 검역이 제대로 이루어지지 않게 된다. 따라서, Bro가 정의하고 있는 tcp_packet 이벤트와 프로토콜별 이벤트를 사용하지 말고, 가장 원본 패킷에 가까운 값을 리턴해주는 new_packet 이벤트와 packet_contents 이벤트를 사용해야 한다.

<표 4>는 Bro의 모든 패킷을 처리할 수 있는 new_packet 이벤트의 의사 코드를 나타낸다. new_packet 이벤트는 Bro가 캡쳐한 패킷을 이벤트로 나누기 전의 상태, 즉 모든 패킷을 보여주는 이벤트로서, Bro가 분리하는 각종 TCP의 프로토콜을 하나의 TCP 패킷으로 처리할 수 있으며, 프로토콜별로 4~5개의 함수를 정의하여 조립하는 것 대신, packet_contents 이벤트에서 리턴되는 payload만 매칭하면 되므로 코드의 직관성을 지킬 수 있다.

```
event new_packet(c: connection, p: pkt_hdr)
{
  if ( p?$tcp )
  {
    #외부 모듈로 데이터 전달
  }
}

event packet_contents (c: connection, contents:
string)
{
   #외부 모듈로 데이터 전달
}
```

<표 4> Bro의 모든 패킷을 처리하는 new_packet 이벤트

3.3 패킷 생성

3.2절에서 기술했듯이, 검사가 끝난 정상 패킷은 Scapy를 이용해서 새로운 패킷을 생성해 종단 사용자에게 전달된다. 이때, 패킷의 payload를 조작할 경우 Tcp retransmission 에러가 발생하여 패킷이 보내지지 않고 커널에서 폐기된다는 큰 이슈가 존재한다. 따라서 패킷의 tagging은 TCP 프로토콜의 예약된 필드인 reserved 필드를 이용해 수행돼야 한다.

또한, new_packet 이벤트를 사용할 경우 TCP flag가 tcp_packet 이벤트에서 반환된 결과와 달리, flag가 숫자로 리턴된다. 이는 아래 <표 5>와 같이 TCP 패킷의 포함 flag의 총합으로서 16보다 클 경우 acknowledge값을 포함해주는 알고리즘을 추가하여 해결할 수 있다.

```
scapy.TCP(sport=tcp_pkt_hdr['dstPort'],
dport=tcp_pkt_hdr['srcPort'],
flags=int(tcp_pkt_hdr['flags']),seq=int(tcp_pkt_hdr['ac
k']), ack=int(tcp_pkt_hdr['seq']),reserved=7)
#ack가 포함된 TCP 패킷

scapy.TCP(sport=tcp_pkt_hdr['dstPort'],
dport=tcp_pkt_hdr['srcPort'],
flags=int(tcp_pkt_hdr['flags']),seq=int(tcp_pkt_hdr['ac
k']), reserved=9)
#일반 TCP 패킷
```

<표 5> Scapy를 이용한 TCP 파라미터 설정

숫자로 리턴되는 TCP flag 값은 Scapy에서 자동으로 처리된다. 따라서 아래 <표 6>과 같은 flag 값과 리턴 값을 매칭 시켜 flag의 값이 16이 넘을 경우 acknowledge number를 포함하도록 구현한다.

Return value	Meaning
1	FIN
2	SYN
4	RST
8	PSH
16	ACK
32	URG

<표 6> TCP 프로토콜 flag

4. 결론

본 논문에서는 SDN의 OpenFlow를 이용하여 적은 비용으로 IPS와 같은 검역체계를 구축할 수 있는 방법에 대해 다루었다. 검역소는 네트워크 상의 모든 패킷을 검사하고, 대처함으로써 규모가 큰 네트워크 구성에 최적화되어 있다. 또한, 종단 사용자들은 사용자들의 디바이스에 애플리케이션을 따로 설치할 필요가 없이 단순 네트워크의 연결만으로도 악성 패킷으로부터 보호받을 수 있다는 이점 또한 존재한다. 하지만 본 논문에서 제안한 검역체계는 MTU(Maximum Transmission Unit) 단위로 단편화된 패킷에 대해 재조립하는 과정이 포함되어 있지 않다는 한계점을 가지고 있다. 따라서 향후 연구로는 MTU 단위로 단편화되어 송·수신되는 패킷의 재조립을 수행하는 어셈블러를 구축하고, 재조립이 끝난 패킷에 대해 검사를 수행하는 과정을 추가한다. 또한, 본 논문에서 구현한 검역소는 네트워크 검역시 시그니처 기반의 정적 분석만을 수행하므로 행위 기반의 동적 분석 또한 추가되어야 한다. 논문에서 제안하는 Bro를 이용한 검역소를 통해 향후 SDN 기반 검역체계 연구 및 구현에 대한 기틀을 마련한다.

ACKNOWLEDGMENT

본 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 중점연구소지원사업으로 수행된 연구임 (2012-005861)

참고문헌

[1] N. McKeown, “openflow (Why Can’t I innovate in my Wiring Closet?),” www.openflow.org/documents/openflow.ppt

[2] 유재형, 김우성, 윤찬현. "SDN/OpenFlow 기술 동향 및 전망." KNOM Review 15.2, 2013

[3] 김태영, 정준권, 정태명. "SDN 기반 트래픽 모니터링 기법에 대한 동향 연구." 한국통신학회 학술대회논문집, 2014

[4] Roesch, Martin. "Snort: Lightweight Intrusion Detection for Networks." LISA. Vol. 99. No. 1. 1999

[5] Paxson, Vern. "Bro: a system for detecting network intruders in real-time." Computer networks 31.23, 1999

[6] Kobayashi, Tiago H., et al. "Using a packet manipulation tool for security analysis of industrial network protocols." Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on. IEEE, 2007