http://dx.doi.org/10.7840/kics.2014.39B.6.397

소프트웨어 정의 네트워크 기반 리플렉터넷

박 태 준◆, 이 승 수*, 신 승 원°

A Reflectornet Based on Software Defined Network

Taejune Park◆, Seungsoo Lee*, Seoungwon Shin°

요 약

소프트웨어 정의 네트워킹(SDN)은 기존의 네트워킹 구조와는 상이하게 제어 계층과 데이터 계층이 분리되어 있어 데이터 계층을 중앙 집중 형태로 제어할 수 있는 네트워킹 구조로서 차세대 네트워킹 기술로 강력히 거론되고 있다. 이러한 기술을 잘 활용하면, 새롭고 다양한 네트워크 기능을 어플리케이션의 형태로 개발이 가능하여, 현재 해당 분야에 대한 연구가 활발히 이루어지고 있다. 뿐만 아니라, 새로운 라우팅 기능 등과 같은 기본적인 네트워크 기능 외에도 네트워크 보안 기능 또한 SDN 기술을 활용하면 재설계가 가능하여 흥미로운 네트워크 보안 애플리케이션들이 다수 제안되었다. 그러나 현재까지 제안된 네트워크 보안 애플리케이션들은 대부분 기존의 네트워크 기능을 SDN기술을 활용하여 재구현하였기 때문에, SDN이 제공하는 많은 기능들을 효과적으로 사용하지는 못하였다. 따라서 본 논문에서는 SDN기술을 사용하여 악성 및 의심스러운 네트워크 공격 시도를 허니팟과 같은 감시/분석 시스템으로 재전송(redirect)해주는 기능인 리플렉터넷(Reflectornet)애플리케이션을 설계하고 구현하였다. 또한, 해당 애플리케이션의 성능과 실용성을 실험을 통해 검증한다. 본 논문의 결과는 SDN 기술을 사용하여 더 지능적이고 진보된 네트워크 보안 애플리케이션을 설계하는 방법에 대한 연구를 촉진하는데 큰 기여를 할 수 있을 것이다.

Key Words : ignaSDN, Security, Reflectornet, Honeypot

ABSTRACT

Software-Defined Networking (SDN), which separates the control plane from the data plane and manages data planes in a centralized way, is now considered as a future networking technology, and many researchers and practitioners have dived into this area to devise new network applications, such new routing methods. Likewise, network security applications could be redesigned with SDN, and some pioneers have proposed several interesting network security applications with SDN. However, most approaches have just reimplemented some well-known network security applications, although SDN provides many interesting features, They didn't effectively use them. To investigate if we can use SDN in realizing sophisticated network security applications, we have designed and implemented an advanced network security application, Reflectornet, which redirects malicious or suspicious network trials to other security monitoring points (e.g., honeypot). In addition, we have tested its performance and practicability in diverse angles. Our findings and some insights will encourage other researchers to design better or intelligent network security applications with SDN.

◆ First Author : KAIST Graduate School of Information Security, taejune.park@kaist.ac.kr, 학생회원
° Corresponding Author : KAIST Graduate School of Information Security, claude@kaist.ac.kr, 정회원
* KAIST Graduate School of Information Security, lss365@kaist.ac.kr, 학생회원
논문번호 : KICS2014-06-219, Received June 2, 2014; Revised June 18, 2014; Accepted June 18, 2014

Ⅰ. 서 론

1.1 서론

사용자의 트래픽 소비 패턴의 변화와 클라우드, 빅 데이터등 새로운 이슈로 인해 네트워크가 복잡하게 변해감에 따라 기존의 네트워크 인프라스트럭처(Infrastructure)와 아키텍쳐(Architecture)의 한계가 드러나기 시작하였다. 이에 새로운 네트워크에 대한 수요가 대두하였고 그 해결책으로 최근 주목 받고 있는 것 중 하나가 소프트웨어 정의 네트워크(Software Defined Network, SDN)이다.[1]

SDN은 기존의 네트워크처럼 이더넷-스위치-라우터에 이어지는 계층적인 구조를 지니는 것이 아니라 데이터의 전송을 흐름(Flow)단위로 처리하며, 이러한 흐름은 제어 계층(Control plane)을 통하여 중앙에서 제어 되며, 데이터 계층(Data plane)을 통하여 전송되는데, SDN은 기존의 네트워크 장비들(예를 들어 네트워크 모니터링, 방화벽 등)의 기능들을 제어계층에서 애플리케이션(Application)형태로 동작시키게 된다. 이를 통해 기존의 네트워크 장비들이 지니는 구현하기 어려운 기능들을 포함하여 새로운 기능을 어렵지 않게 네트워크에 추가 할 수도 있다. 이와 같이 SDN이 지닌 유연성으로 덕분에 오픈플로우(OpenFlow)라는 SDN 표준 기술의 확산을 장려하고 더불어 학계에서는 오픈플로우에 기반을 둔 여러 가지 새로운 네트워크 기술들이 쏟아져 나오고 있다. 물론 학계뿐만이 아닌 Google, Facebook, Microsoft등의 소위 일컫는 거대 IT기업들도 이미 자사의 데이터센터(Data center)를 SDN기반의 네트워크를 적용시키고 있으며, 성과들이 속속들이 보고되고 있다.[2-8,13,14]

SDN을 통하여 네트워크 보안 관련 기술을 구현하고자 하는 시도도 기존에 많이 있었다. 하지만 대부분의 SDN기반의 보안 연구는 컨트롤러(Controller)기반의 보안 혹은 오픈플로우 흐름표(OpenFlow FlowTable) 분석에 국한되었을 뿐 SDN의 강점을 활용하여 실제 환경에서 보안 기능을 구현한 연구는 부족한 실정이다.

우리는 SDN의 장점을 잘 살릴 수 있는 보안기능 중 하나로 리플렉터넷(Reflectornet)을 꼽고 있다. 리플렉터넷은 네트워크 보안 장비중 하나로, 악의적이라 판단되는 호스트의 요청을 허니팟(Honeypot)으로 보내어 행동에 대한 로그를 유지 하고 분석을 할 수 있도록 해준다. 리플렉터넷의 기능을 구현하기 위해서는 패킷 헤더(Packet header)의 출발지 주소(Source address)와 도착지 주소(Destination address)에 대한 변조가 필요하며, 어떤 패킷 흐름을 허니팟으로 보낼지에 대한 흐름 테이블(Flow table)이 필요하다.

기존의 네트워크에서 이러한 기능을 수행하기 위해서는 새로운 보안 장비의 구매를 위한 비용문제는 물론 해당 장비를 위치시킬 물리적인 위치고려, 장비를 연결시키기 위해 기존 장비와의 연결 및 설정 등의 작업이 추가적으로 소요 된다. 그래서 인터넷 서비스 공급자(Internet Service Provider, ISP), 데이터센터등 거대한 네트워크를 다루는 곳에서는 매우 쉽지 않은 작업이다.

하지만 SDN/OpenFlow는 자체적으로 패킷 헤더에 대한 변환 기능을 제공을 해주므로 해당 기능을 수행하는 애플리케이션을 제작하는 방법을 통해 상대적으로 매우 쉽게 적용할 수 있다.

이 논문에서는 리플렉터넷을 소프트웨어 정의 네트워크를 통하여 구현하였으며, 실제 동작을 확인해본다.

1.2 관련 연구

기존에도 SDN을 활용한 보안 연구가 몇 가지 있다.

R. S. Barga 등의 연구[3]는 흐름규칙표를 읽어와 이를 토대로 DDoS(Distributed Denial of Service)공격을 탐지한다. 이 연구는 흐름표만 읽어오는 것에서만 SDN을 활용했다는 점이 차이가 있다.

S. Shirali-Shahreza 등의 연구[9]는 효율적으로 네트워크 표본 수집을 하기 위한 방법을 제시한다. 이 방법은 오픈플로우 메시지를 별도로 추가한다는 점에서 본 연구와 차이가 있다.

CloudWatcher[7]은 클라우드 환경에서 기존의 보안 장비를 SDN에 적용시킬 수 있는 방법에 대해 논한다. 이 논문은 SDN에 보안 애플리케이션을 직접 올린다는 점에서 차이가 있다.

M. Yu등의 연구[10]은 효율적인 네트워크 흐름 측정 방법을 제공해준다. "Avant-guard"[11]는 SDN의 보안성을 조금 더 향상시켰다. 이 연구들은 데이터계층을 직접 수정을 했기에 우리 연구와 차이가 있다.

상기한 연구들과 우리 연구는 SDN의 기능을 적극적으로 사용하면서 기존의 것들을 수정을 하지 않는다는 점이 차이가 있다. 우리 연구와 비슷한 연구로는 "Flowsense"[12]가 있는데 이 연구는 오픈플로우 메시지를 이용해 네트워크 감시를 할 수 있게 해주는 애플리케이션인데 SDN의 특징을 잘 활용한 연구 중 하나이다. 우리 연구는 SDN의 기능을 조금 더 능동적으로 이용한다는 점에서 차이가 있다.

Ⅱ. 본 론

2.1 소프트웨어 정의 네트워크와 오픈플로우

본 절에서는 소프트웨어 정의 네트워크가 무엇인지 간략하게 알아보며, SDN인터페이스 중 하나인 오픈플로우가 무엇인지에 대해 알아본다.

2.1.1 소프트웨어 정의 네트워크

소프트웨어 정의 네트워크는 크게 데이터 계층(=기반 계층 Infrastructure layer), 제어 계층, 애플리케이션 계층으로 나뉜다. 데이터 계층은 제어 계층의 특정 인터페이스를 통해 제어를 받는 계층으로서, 데이터 흐름의 전송을 담당한다. 제어계층은 데이터 흐름을 제어하는 계층으로서 애플리케이션과 네트워크 서비스들을 통하여 데이터 흐름이 어떤 동작을 할지-라우팅을 할 것인지, 전달을 할 것인지, 거절할 것인지 등-를 결정한다. 또한 데이터 계층의 동작들을 추상화 하여 API형태로 애플리케이션 레이어에 기능들을 제공한다. 애플리케이션 계층은 제어 계층이 제공한 API들을 이용하여 네트워크의 여러 기능들을 수행할 수 있도록 한다. Figure 1은 SDN 설계를 나타내고 있다.[1]

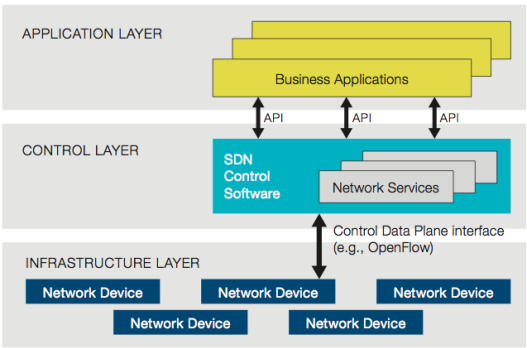
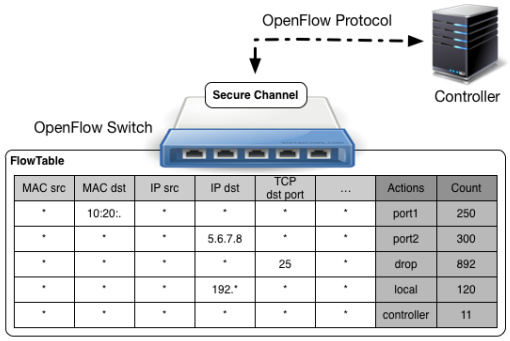


Fig. 1. Software-Defined Network Architecture.

2.1.2 오픈플로우

현재 데이터계층을 제어하기 위하여 제어계층간의 통신에 이용되는 인터페이스 중 현재 가장 많이 사용되는 것이 오픈플로우이다. 오픈플로우는 SDN의 첫 번째 표준으로서 오픈플로우를 지원하는 스위치는 흐름표(Flow table)를 가지고 있으며, 컨트롤러와 별도의 채널을 통해 연결된다. Figure 2는 간단히 오픈플로우 스위치와 컨트롤러와의 관계를 나타낸 것이다.



MAC src	MAC dst	IP src	IP dst	TCP dst port	...	Actions	Count
*	10:20:.	*	*	*	*	port1	250
*	*	*	5.6.7.8	*	*	port2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Fig. 2. OpenFlow. The OpenFlow switch has a Flow table.

(1)흐름표

오픈플로우 스위치(혹은 오픈플로우 지원 스위치 OpenFlow enable switch)들은 흐름표를 가지고 있다. 이 흐름표는 흐름규칙(Flow rule)을 나타내는데 흐름규칙은 일치 항목(Match field), 동작 항목(Action field), 바이트 수(Byte count)와 패킷의 수(Packet count)로 구성되어 있다.

일치 항목은 스위치 포트, 맥 주소(MAC Address), IP주소(IP Address), 포트 주소(Port Address)등 각 계층별 헤더에 대한 정보들로 구성되어 있는데, 스위치로 들어온 패킷을 분석하여 만족하는 일치 항목을 검사한다. 일치항목은 특정한 값을 지정할 수 도 있고, 무정의(無定義, Don't care)항목을 지정할 수도 있다. 일치항목에 해당하는 항목을 찾으면 해당 패킷은 동작항목에 있는 내용을 수행하게 되는데 가능한 동작은 전송(Forward), 대기열저장(Enqueue), 무시(Drop), 그리고 항목변조(Modify-Field)가 있다. 동작들은 동시에 두 개 이상도 입력이 가능하므로, 여러 동작을 복합적으로 수행할 수 있다. 바이트 수와 패킷의 수는 해당 일치항목을 만족시킨 총 바이트 양과 패킷의 개수를 의미하며 네트워크 통계정보 확인에 이용된다.

(2) 동작

패킷이 오픈플로우 스위치에 들어오게 되면 스위치의 흐름표를 먼저 검사하게 된다. 만약 흐름표에 만족하는 패킷이 있는 경우에는 지정된 동작을 수행하게 된다. 예를 들어 Figure 2의 스위치에 도착지 IP주소가 5.6.7.8인 패킷이 들어오면 해당 패킷은 스위치 포트2로 나가게 된다. 만약 만족하는 항목이 테이블에 존재하지 않으면 해당 패킷의 정보를 포함하는 오픈플로우 메시지(OpenFlow Message)중 새로운 패킷의

도착을 알리는 PackettIn 메시지가 컨트롤러로 전달된다. 컨트롤러는 해당 메시지를 보고 어떠한 동작을 수행할지 결정하게 된다.

2.2 리플렉터넷

본 장에서는 리플렉터넷이 무엇인지 알아보고, 기존의 리플렉터넷 장비를 활용할 때의 문제점, 그리고 SDN을 이용하여 구현할 때의 이점과 방법에 대해 기술한다.

2.2.1 동작원리

리플렉터넷은 네트워크 보안 장비중 하나로, 특정 호스트의 행위를 허니팟(Honeypot)으로 보내어 행동에 대한 로그를 유지 하고 이상 행위에 대한 분석을 좀 더 쉽게 할 수 있도록 해준다. 단순히 네트워크 감시(Network monitoring)나 네트워크 침입 탐지 시스템(Network Intrusion Detection System, NIDS)과 다른 점은 이상행위에 대해 경고나 차단을 하지 않고 적극적으로 응한다는 점이다. 대신 정상적인 서비스에는 지장이 없도록 이상행위를 하는 트래픽은 허니팟으로 보낸다.

Figure 3에서 악성행위를 하는 호스트(B)가 특정 서비스(C)로 접근을 시도 한다고 가정해보자. 이 접근을 리플렉터넷이 감지를 하게 되면, 목적지 주소(C)를 본래의 서비스가 아닌 허니팟(D)의 주소로 바꾸고, 이를 허니팟(D)으로 보내준다. 허니팟은 악성 행위에 대해서도 정상적인 응답을 악성 호스트(B)에게 해준다. 이때 돌아가는 패킷에 대해서도 리플렉터넷이 중간에 출발지 주소(=허니팟)를 서비스(C)의 주소로 바꾸어 최종적으로 이를 수신한 악성 호스트 입장에서는 본래의 서비스(C)가 응답을 해준 것으로 보이게 한다.

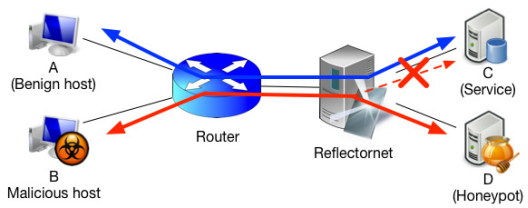


Fig. 3. Reflectornet function. Reflectornet changes the destination address. So the malicious host can't reach the original service. Instead it connects with the honeypot.

2.1.2 기존 네트워크에서 리플렉터넷의 문제점

가장 기본적인 문제는 리플렉터넷을 추가적으로 설치하는 경우 네트워크 망 구성(Topology)이 바뀌게 된다는 점이다. 소규모 네트워크에서는 그 영향이 크지 않지만 인터넷 서비스 공급자(ISP), 데이터센터등 대규모의 네트워크를 운용하는 곳에서는 이를 설치할 공간 확보가 쉽지 않을뿐더러 설치하는 동안에는 망을 서비스 도중에 끊어야 하는 문제점이 발생한다. 그리고 리플렉터넷 장비를 별도로 구매해야 하는 비용 문제도 고려해야 한다.

또한, 내부 망이나 클라우드 네트워크 환경에서 이 문제는 더욱 커진다. Figure 4를 보자.

악성 호스트 G는 외부에서 들어오는 내부의 B까지의 경로 상에 리플렉터넷이 존재하여 허니팟(F)과 대응하게 된다. 하지만 내부에 존재하는 악성 호스트 D는 경로 상에 리플렉터넷이 없어서 서비스 B까지 도달가능하게 되며, 어떠한 공격 흔적도 네트워크상에 남길 수 없다. 이를 위해서는 각 네트워크 장비(위 그림에서는 라우터) 사이사이 마다 모두 리플렉터넷을 도입(Distributed reflectornet)을 해야 하지만 비용문제를 고려해야 하기 때문에 실제로 도입하기 어렵다.

사실 이러한 측면은 비단 리플렉터넷만의 문제는 아니다. 방화벽(Firewall), 침입탐지/방지시스템(IPS, IDS)등도 위와 같은 문제점이 발생한다. 즉, 기존 네트워크의 유연성과 확장성과 관련된 문제점이라 볼 수 있다.

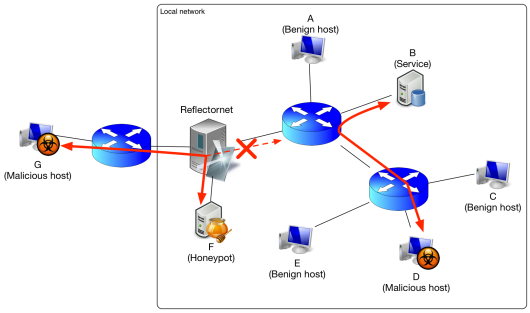


Fig. 4. Problem of the reflectornet in the traditional network. It can't detect or protect the attack which origins from the local network.

2.1.3 오픈플로우를 이용한 구현

오픈플로우를 활용하면 리플렉터넷의 동작을 비교적 간단하게 적용할 수 있다. 기본적으로 오픈플로우는 동작항목을 통해 패킷 헤더의 정보를 쉽게 변경시킬 수 있다. 즉, 악성이라 여겨지는 호스트가 출발지 주소인 데이터 흐름이 들어오면, 컨트롤러의 리플렉터넷이 악성 호스트가 출발지인 플로우의 목적지 주소를 허니팟으로 바꿔주는 규칙과, 출발지가 허니팟인 플로우의 출발지 주소를 본래의 서비스 주소로 바꿔주는 규칙을 추가하면 된다.

Figure 5는 SDN기반의 리플렉터넷을 보여주고 있다.

Figure 5를 살펴보면 SDN스위치에 어떠한 흐름 규칙이 있다. 이 흐름규칙은 컨트롤러에 설치된 리플렉터넷에 의해 작성된 규칙이다. 각 데이터들은 작성된 흐름규칙에 따라 각 데이터 흐름이 움직이게 되는데 이로 인하여 A호스트에서 서비스(C)로 가는 흐름은 정상적으로 도달이 되지만 악성 호스트(B)에서 서비스(C)로 가는 호스트는 목적지 주소가 허니팟(D)으로 변경이 되고, 허니팟(D)으로 보내진다. 허니팟은 이에 대한 정상적인 응답을 해주게 되는데 이때 본래의 서비스(C)가 해주는 것처럼 보이기 위해 출발지 주소를 허니팟(D)이 아닌 서비스(C)의 주소로 변경시킨 뒤 B로 전달하게 된다.

Figure 6과 같이 내부에서의 악성행위에 대한 리플렉터넷 동작도 별다른 추가 장비 필요 없이 가능하다. SDN 컨트롤러에 설치된 리플렉터넷 애플리케이션은 각 스위치에서 올라오는 PacketIn메시지를 확인하고 적당한 흐름 규칙을 스위치에 입력해준다.

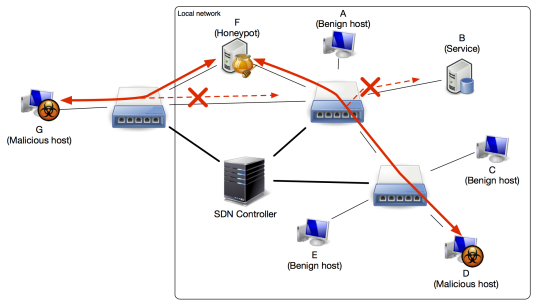


Fig. 5. SDN based the distributed reflectornet system. The malicious host can't reach the service even if it occurs from the local network. (Thick lines are the OpenFlow channel)

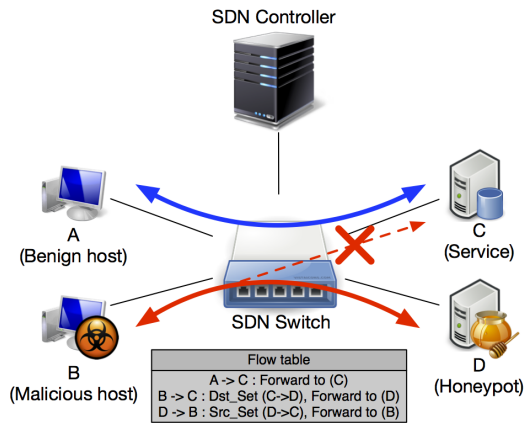


Fig. 6. SDN based Reflectornet. Both hosts want to connect with the service. However the malicious host can't connect with the service, instead connect with the honeypot.

Ⅲ. 실 험

3.1 구현 및 실험환경

3.1.1 구현

본 장에서는 구현한 리플렉터넷 애플리케이션에 관한 설명과 동작 방식에 대해 알아보도록 하겠다.

(1) 구현

본 실험에서 사용될 리플렉터넷은 오픈플로우 컨트롤러 중 하나인 플루드라이트(Floodlight)를 이용하여 제작하였다. 플루드라이트는 자바(Java)기반의 컨트롤러로서 현재 가장 많이 사용되는 컨트롤러 중 하나이다.

Figure 7은 구현한 리플렉터넷의 구성도이다. 스위치가 보낸 오픈플로우 메시지를 수신 받는 'PacketIn' listener 가 있고 처리한 패킷을 다시 재전송 해주기 위한 'PacketOut' sender, 그리고 흐름규칙을 스위치에 입력하기 위한 'FlowMod' sender가 있다. 그리고 외부에 인터페이스를 제공해주기 위하여 REST API Provider를 통해 REST API를 제공하며, 리플렉터넷이 작동하였을 시 기록을 유지하기 위한 Log writter, 그리고 어떤 패킷에 리플렉터 동작을 수행할지를 결정하는 RuleManager가 있다.

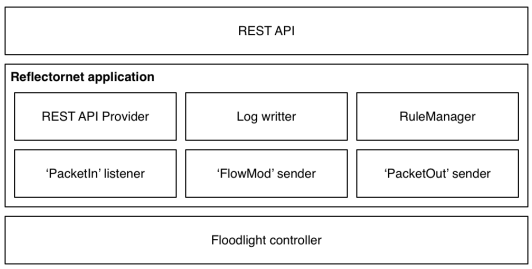


Fig. 7. Reflectornet architecture

(2) 동작 방식

Figure 8을 통해 동작방식을 확인해보자. 이때 스위치에는 어떠한 흐름규칙도 정의되어있지 않다고 가정한다. 먼저 (1) REST API를 이용해 관리자가 규칙을 RuleManager에 입력한다. 이 규칙은 어떤 조건에 리플렉터넷을 활성화 시킬지에 대한 것이다. (2) 악성 호스트(B)가 서비스(C)를 향해 패킷을 보내면 이를 중간에 스위치가 수신하게 된다. 스위치에는 현재 아무런 흐름규칙이 정의되어 있지 않으므로, (3) 해당 패

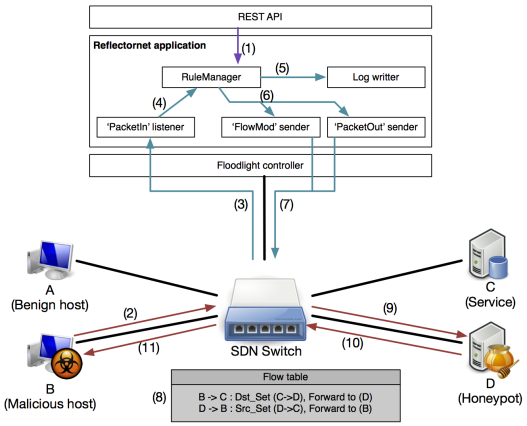


Fig. 8. Work process

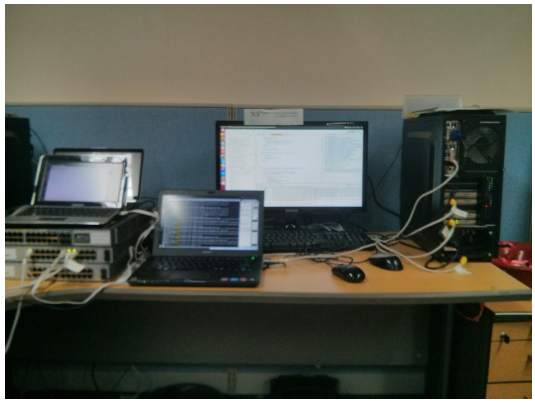


Fig. 9. Test environment

킷의 정보를 컨트롤러로 보내게 된다. 리플렉터넷 애플리케이션에서 이 정보를 수신하고, (4) 이 정보를 RuleManager에 전달한다. RuleManager는 해당 패킷 정보와 관리자가 입력한 규칙을 비교한 후, 리플렉터넷의 동작 여부를 결정한다. 만약 일치하는 규칙이 없을 경우에는 정상적인 전달규칙을 내리거나, 다른 전달 애플리케이션(Forwarding application)의 동작을 따른다. 일치하는 항목이 있으면 (5) 해당 정보에 대한 기록을 남기고 (6)(7)(8)스위치에 'FlowMod' sender를 통해 리플렉터넷 동작 규칙을 입력하고 (9) 'PacketOut' sender를 통해 패킷을 다시 전송시켜준다. 이후 허니팟(D)이 (10) 응신을 하게 되면 스위치에서 흐름규칙을 확인하고 해당 규칙에 맞추어 악성 호스트(B)에게 전달한다. 이후의 통신은 스위치에 흐름규칙이 정해졌기 때문에 (3)~(7)의 과정은 생략되고 바로 악성 호스트(B)와 허니팟(D)끼리 통신이 가능하다.

3.2 실험

3.2.1 실험

본 절에서는 구현한 리플렉터넷의 동작을 확인하고 이에 대한 토의를 해본다.

(1) 실제 동작

실험 환경은 네트워크 망 구성은 HP E3800 스위치를 활용하여 Figure 8의 망을 구현하였다. Figure 9은 실제 구현현 모습이다.

허니팟은 다이오네아(Dionaea)[16]를 사용하였으며 기본적인 로그와 패킷 데이터들을 기록 및 저장하는 기능을 수행한다. 실험은 Figure 8와 같이 악성 호스트에서 서비스에 접근하려할 때 리플렉터넷의 동작과 허니팟의 동작을 확인해보는 방식으로 진행하였다.

Figure 10 은 플루드라이트에서 리플렉터넷이 동작하는 화면이다. 일치하는 규칙을 항목을 발견하면 목

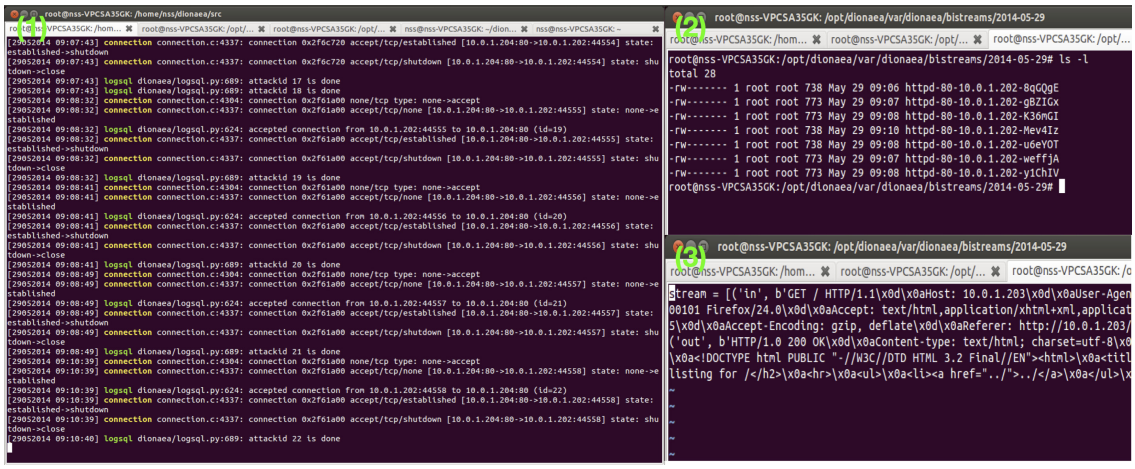


Fig. 10. honeypot behavior (1) Real time log (2) Collected list (3) Collected data

적지 주소를 허니팟 주소로 변경시켜주는 것을 확인할 수 있다.

Figure 11은 허니팟의 동작을 보여주는 그림이다. (1)은 허니팟 소프트웨어에서 실시간으로 찍히는 로그 데이터로 어떤 데이터가 어디에서 왔는지 잘 나타내준다. (2)은 수집한 데이터들을 보여주는 것이고, (3)은 실제 수집 데이터를 나타낸다.

```
09:08:28.540 INFO [n.f.j.JythonServer:debugserver-main] Starting DebugServer on :6655
------------------------------------------------------------
** Source IP [10.0.1.202] is matched !
** Destination  address [10.0.1.203]/[8C:89:A5:A7:7E:AE]
** Modified to
** ReflectorNet address [10.0.1.204]/[F0:BF:97:E9:49:84]
------------------------------------------------------------
** Source IP [10.0.1.202] is matched !
** Destination  address [10.0.1.203]/[8C:89:A5:A7:7E:AE]
** Modified to
** ReflectorNet address [10.0.1.204]/[F0:BF:97:E9:49:84]
```

Fig. 11. The reflectornet works. If the rule is matched, the flow is going to forward to honeypot.

(2)측정

측정은 패킷을 각각 1Mbps, 10Mbps, 100Mbps, 300Mbps, 500Mbps, 1000Mbps의 속도로 보내어 그 처리량과 컨트롤러에서의 CPU사용량을 측정하였다.

Figure 12를 보면 패킷 전송 속도가 증가하면 할수록 처리량이 급격하게 떨어지는 것을 확인할 수 있다. 이는 오픈플로우 스위치의 한계로서, 복잡한 작업인 패킷의 헤더 정보를 바꾸는 것에서 기인한다.

Figure 13은 CPU 성능을 나타낸다. 우리가 구현한 리플렉터넷은 통신 중에 컨트롤러가 개입하는 경우는 최초의 딱 한번이다. 즉, CPU의 사용량에는 큰 변화가 없다. 하지만 Figure 13을 살펴보면 Figure 12와 마찬가지로 전송 속도가 증가할수록 하락하는 그래프가 나타나지만, 실제 컨트롤러가 동작하는 구간은 딱 한 구간 밖에 없으므로, 이는 사실 어느 정도 오차 범위 이내이다.

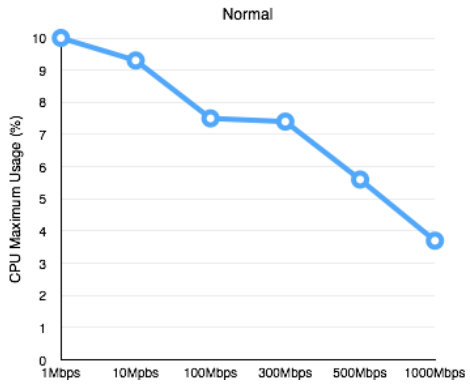


Fig. 12. CPU Maximum Usage.

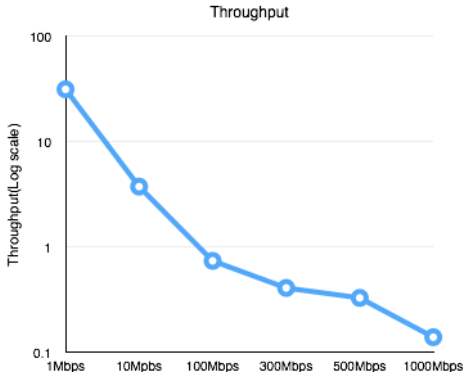


Fig. 13. Throughput. It is a log scale graph.

(3) 토의

우리는 리플렉터넷의 동작을 소프트웨어 정의 네트워크 환경에서 실제로 구현을 해보았다. 특히 본 애플리케이션은 오픈플로우 프로토콜을 수정을 하거나 별다른 동작을 추가하지 않고 있는 그대로, 그리고 SDN이 가지는 장점을 십분 활용했다는 점에서 큰 의의가 있다고 생각한다.

실제 물리적인 리플렉터넷과 별반 다르지 않게 매우 잘 동작하는 것을 확인할 수 있다. 단지 기본적인 네트워크 환경에서 리플렉터넷 애플리케이션 하나만 추가 하면 별도의 추가비용 없이 네트워크에 새로운 기능을 구현할 수 있다는 것은 큰 장점이다. 추가적으로 리플렉터넷뿐만 아니라 다른 보안 애플리케이션들을 함께 동작시키면 더욱 안전한 네트워크 사용이 가능해 질 것이다. 하지만 이럴 경우 애플리케이션 들간에 규칙이 서로 충돌하거나 어긋나는 문제점이 발생할 수 도 있지만, 이는 기존의 Rule 충돌을 감지하고 막을 수 있는 기술들[15]를 활용하여 효과적으로 해결할 수 있을 것으로 기대한다.

처리량의 경우는 HP E3800 스위치가 펌웨어 업데이트(Firmware update)형식으로 오픈플로우를 지원을 하기 때문에 오픈플로우의 기능들을 소프트웨어적으로 처리한다. 따라서 출발지 주소나 목적지 주소 같은 것을 바꾸는 작업은 복잡한 작업이기 때문에 스위치가 그 처리량을 따라가지 못하는 한계가 있었다. 오픈플로우 기능을 하드웨어 적으로 처리해주는 전문 오픈플로우 스위치를 사용한다면 처리량은 급격히 올라갈 것으로 생각된다.

CPU사용량은 매우 적었다. 우리 실험에서는 최고 10%의 사용량을 보이기는 하였지만 실제 동작 흐름을 살펴보면 CPU가 사용되는 구간은 단 한 구간이므로 크게 영향을 미치지는 못할 것으로 생각된다.

즉, SDN/오픈플로우를 통해 구현한 리플렉터넷은 저비용, 낮은 자원사용량으로 구현된다고 결론 내릴 수 있다.

Ⅳ. 결 론

소프트웨어 정의 네트워크는 각종 네트워크의 기능들을 애플리케이션 형태로 운용을 하기 때문에 기존의 네트워크와는 다르게 매우 유연하다. 단순한 네트워크 감시부터 복잡한 네트워크 보안 기능들 까지 저비용으로 다양한 기능들을 네트워크 망에 추가할 수 있다. 하지만 기존에 있던 많은 연구들은 SDN의 장점을 잘 살리진 못하고 기존에 있던 기능들을 답습하는 수준에 그쳤다. 우리는 SDN의 장점을 극대화하면서 유용한 네트워크 기능이 무엇이 있을지 고민을 해보았고 한 가지 예시로 리플렉터넷을 구현을 해보았다. 리플렉터넷은 출발지 주소와 도착지주소에 대한 변조가 필요한데 오픈플로우는 이러한 기능을 기본적으로 제공해주므로, 놀라울 만큼 간단하게 구현하고 적용할 수 있다. 또한 소프트웨어로 구성되어있는 만큼 돌발 상황에서 하드웨어에 비해 상대적으로 빠르게 대처할 수도 있다.

아직까지는 처리량 문제로 인하여 당장 실제 네트워크를 구현하여 적용 하기는 조금 이른 면이 없지 않아 있다. 하지만 이는 오픈플로우 전용 스위치 등의 기술 개발로 통하여 충분히 해결될 수 있으리라 믿는다.

향후에는 단순히 네트워크 기능들을 SDN으로 구현만 하는 것이 아닌, SDN을 이용하여 구현하여 이점이 있는 기능, SDN의 장점을 잘 살릴 수 있는 기능들에 대해 고민을 해보고 연구를 해보아야 할 것이다.

References

[1] ONF Market Education Committee, Software-Defined Networking: The New Norm for Networks, ONF White Paper, Palo Alto, US: Open Networking Foundation, Apr. 2012.

[2] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. Workshop on Research in Enterprise Networks*, pp. 11-18, Barcelona, Spain, Aug. 2009.

[3] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Proc. IEEE Conf. Local Computer Networks(LCN)*, pp. 416-423, Denver, CO, USA, Oct. 2010.

[4] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. Usenix Conf. Networked Systems Design and Implementation (NSDI'12)*, pp. 351-364, San Francisco, California, Apr. 2012.

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *Proc. Usenix Conf. Networked Systems Design and Implementation (NSDI'10)*, San Jose, pp. 17-17, California, Apr. 2010.

[6] R. Sherwood, G. Gibb, K. K. Yap, and G. Appenzeller, "Can the production network be the testbed," in *Proc. USENIX Operating System Design and Implementation(OSDI)*, pp. 1-6, Vancouver, BC, Canada, Oct. 2010.

[7] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Proc. Workshop Secure Network Protocols (NPSec'12), co-located with IEEE ICNP'12*, pp. 1-6, Austin, TX, USA, Oct. 2012.

[8] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *Proc. Annu. Network and Distributed System Security Symp. (NDSS'13)*, San Diego, CA, USA, Feb. 2013.

[9] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in openflow controller with flexam," in *IEEE Annu. Symp. High-Performance Interconnects*, pp. 49-54, San Jose, CA, USA, Aug. 2013.

[10] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, pp. 29-42, Berkeley, CA, USA, Apr. 2013.

[11] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, “Avant-guard: Scalable and vigilant switch flow management in software-defined networks,” in *Proc. ACM Conf. Comput. Commun. Security (CCS’13)*, pp. 413-424, Berlin, Germany, Nov. 2013.

[12] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. “Flowsense: monitoring network utilization with zero measurement cost,” in *Proc. Int. Conf. Passive and Active Measurement (PAM’13)*, pp. 31-41, Hong Kong, China, Mar. 2013.

[13] S. Jain et al, “B4: Experience with a globally-deployed software defined WAN,” in *Proc. ACM SIGCOMM 2013*, pp. 3-14, Hong Kong, China, Aug. 2013.

[14] Facebook, Open Computer Project(2011), Retrieved May, 29, 2014, from http://www.opencompute.org/projects/networking/

[15] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for openflow networks,” in *Proc. Workshop Hot topics in software defined networks(HotSDN’12)*, pp. 121-126, Helsinki, Finland, Aug. 2012.

[16] E. Levy, “Dionaea: On the automatic collection of malicious code samples through honey pot farms,” in *Invited talk at the CASCON 2005 Workshop on Cybersecurity,* Toronto, Canada, Oct. 2005.

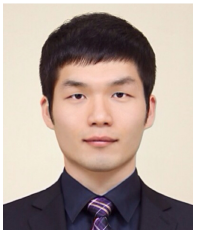
박 태 준 (Taejune Park)

2013년 8월 : 한국해양대학교 IT 공학부 졸업

2013년 9월~현재 : 카이스트 정보보호대학원 석사과정

<관심분야> 보안, SDN, 시스템

이 승 수 (Seungsoo Lee)

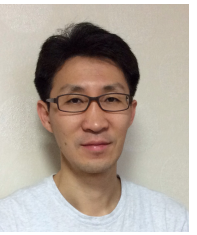


2014년 2월 : 숭실대학교 컴퓨터공학부 졸업

2014년 3월~현재 : 카이스트 정보보호대학원 석사과정

<관심분야> SDN, 네트워크

신 승 원 (Seungwon Shin)



2000년 2월 : 카이스트 전기전자공학부 석사

2013년 8월 : Texas A&M University Electrical and Computer Engineering 박사

2013년 9월~현재 : 카이스트 정보보호대학원 조교수

<관심분야> 보안, SDN, 클라우드, 봇넷