소프트웨어 정의 네트워크(SDN)를 대상으로 한 퍼즈테스팅 관련 연구 조사

위성일*, 손수엘*
*한국과학기술원 정보보호대학원
e-mail: {seongil.wi, sl.son}@kaist.ac.kr

A Study on Fuzzing Tools for Testing Software-Defined Networks

Seongil Wi*, Sooel Son*
*Graduate School of Information Security, KAIST

요 약

최근 주목 받고 있는 소프트웨어 정의 네트워크(SDN: Software-Defined Networks)는 기존 네트워크 운용의 비효율성과 복잡성을 근본적으로 해결하기 위해 등장한 개방형 네트워크 인프라이다. SDN 시스템이 점차 상용화, 개방화되는 시점에서, 내재되어 있는 보안적 위협을 줄이기 위하여 효율적이고 자동화된 취약점 탐지의 필요성이 대두되고 있다. 본 논문에서는 자동화된 소프트웨어 테스트 기법 중 하나인 퍼즈테스팅이 SDN 에 적용되어야 할 이유를 살펴보고자 한다. 또한, 기존에 관련된 연구의 분석을 통해 현재 학계의 연구동향을 파악하고 앞으로의 연구 방향성을 제시한다.

1. 서론

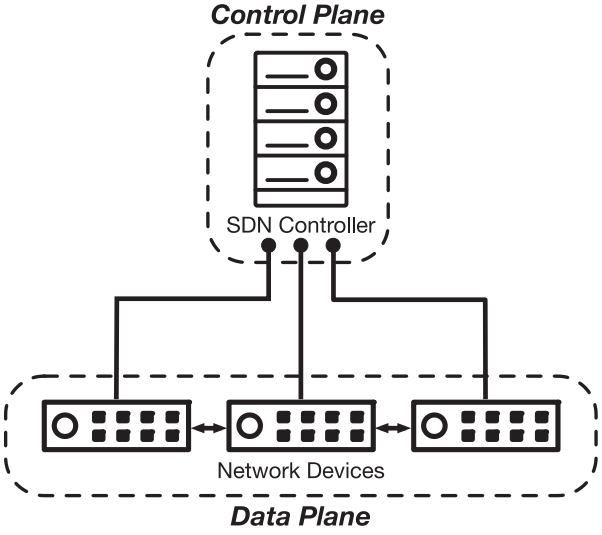
최근 많은 관심을 받고 있는 소프트웨어 정의 네트워크(SDN: Software-Defined Network)는 전통적인 네트워크 구조가 가지고 있는 경직성 및 설정 갱신의 지연과 같은 문제해결을 위하여 도입된 차세대 네트워크 시스템이다. SDN 은 기반이 되는 OpenFlow [4] 프로토콜과 함께 표준화가 되면서 산업화 및 개방화 또한 급속히 이루어지고 있다 [8], [9].

그러나, 이와 함께 대두되는 사이버보안의 문제 [5], [6], [7]로 인하여 SDN 인프라에 대한 보안 및 방어의 필요성 또한 급속히 증가하고 있는 실태이다. 특별히 SDN 시스템을 대상으로 하는 수동적인 보안 점검과 감시를 넘어, 선제적으로 취약점을 찾는 시스템의 자동화에 대한 많은 연구들이 이루어 지고 있다.

본 연구에서는 자동화된 취약점 탐지 기법 중 하나인 퍼즈테스팅, 퍼징(Fuzzing)이 SDN 을 테스트하기 위하여 필요한 이유와, 이에 대한 적용 사례 및 연구 동향에 대해 살펴보고자 한다.

2. SDN 의 개념과 퍼즈테스팅의 필요성

SDN 은 기존의 네트워크상에서 한 장비에 같이 운용되어 왔던 제어평면(Control Plane)과 데이터평면(Data Plane)을 분리하는 개념이다(그림 1). 특별히 논리적으로 중앙집권화된 제어평면을 둠으로써 관리자로 하여금 복잡한 트래픽을 중앙집권화된 시각으로 쉽게 관리 및 설정을 할 수 있게 하고, 네트워크에 대한 추상화 및 소프트웨어 프로그래밍을 통해 네트워크 경로 설정과 통신기능을 다양한 방식으로 제어할 수 있게 하였다.



(그림 1) SDN System 의 개요

SDN 이 급속도로 산업화 및 개방화가 되고 표준화가 이뤄짐에 따라, 이에 대한 사이버보안의 문제가 대두되고 있는 추세이다. 특별히, 중앙집권화된 하나의 제어평면에서의 보안적 문제는 심지어 전체 네트워크를 마비시킬 수 있는 치명적인 영향을 가지고 있

다. 때문에, SDN 시스템 전반에 대한 테스트 및 취약점 탐지의 중요성 또한 높아지고 있다. 그러나, 이에 대한 수동적 테스트 [15], [16] 에는 시간적, 인력적 한계가 존재한다. 이를 극복하기 위하여, SDN 시스템을 대상으로 수행하는 자동화된 취약점 탐지에 대한 연구가 필요로 된다. 본 논문에서는 자동화된 취약점 탐지 기법 중 하나인 퍼즈테스팅이 SDN 에 적용된 연구 사례를 살펴보고자 한다.

퍼즈테스팅, 퍼징(Fuzzing)은 1990 년 Miller [10]에 의해 개발된 기법으로서, 임의의 입력 값을 생성하고 그 입력 값으로 프로그램을 실행 하는 형식을 통하여 프로그램을 테스트하는 기법이다. 이 기술은 오늘날 크게 발전되어 [12], [13], [14] 소프트웨어의 버그를 탐지하기 위한 테스트 기법의 전반을 아우르고 있다. 그 효과 또한 상당하여, 실제로 수많은 소프트웨어 보안 취약점을 찾아냈고 [11], 테스트 기법도 매우 간단하여 실제 산업에서도 많이 쓰이고 있다.

3. 현존하는 SDN 시스템 퍼즈테스팅 연구 조사

3.1 현존하는 SDN 시스템 퍼즈테스팅 연구

DELTA [1] 는 현존하는 OpenFlow 프로토콜의 제어흐름(Control Flow)을 Symmetric 제어흐름, Asymmetric 제어흐름, Intra-Controller 제어흐름으로 구분하고, 이 흐름에 속하는 SDN 구성요소의 공격 시나리오를 찾기 위한 블랙박스 퍼즈테스팅 알고리즘을 제안하였다. 특별히, DELTA 는 제어흐름을 기반으로 현재 SDN 컨트롤러의 운용상태(Operational State)를 유추하고, 그에 부합한 퍼즈테스팅 시나리오를 구상하게 된다. 이를 위해, 제어평면–데이터평면 사이의 채널을 감시하여 제어흐름 및 메세지 내용을 무작위로 변형시키기 위한 채널 에이전트(Channel Agent)와 SDN 어플리케이션에서의 제어흐름 및 메세지 내용을 변형시키기 위한 어플리케이션 에이전트(Application Agent)를 활용하여 전에는 알지 못했던 공격 시나리오를 찾는 방법을 사용한다.

BEADS [2] 는 데이터평면에서 일어날 수 있는 공격시나리오를 탐색하기 위하여 악의적인 OpenFlow 스위치와 호스트를 이용한 퍼즈테스팅 공격탐색기법을 제안하였다. 악의적인 스위치에서는 지나가는 OpenFlow 패킷에 대하여 구문론적, 의미론적 속성을 분석하여 Drop, Duplicate, Delay, Change 연산을 이용한 패킷의 무작위 변형을 시도하며, 악의적인 호스트에서는 ARP 패킷에 대하여 ARP 스푸핑이 가능하도록 패킷을 변형하는 방식으로 SDN 시스템 전반에 대한 퍼즈테스팅을 시도한다.

AIM-SDN [3] 은 SDN 컨트롤러의 Network Management Datastore Architecture(NMDA) 구조상에 존재하는 Datastore 취약점을 찾기 위하여 퍼즈테스팅 툴을 제안하였다. 특별히, 이 연구에서는 SDN 어플리케이션에서 컨트롤러의 특정 Datastore 에 저장한 데이터가 다른 Datastore 에 동기화되지 않는다거나, 정보 자체의 소유권이 특정되지 않게 되는 의미론적 차이(Semantic Gap)를 이용한 공격시나리오를 밝혀내고자 하였다. 이를 위하여 Datastore 와 연관된 SDN 서비스에게 무작위로 RESTful 요청 패킷을 보내고 응답 패킷을 확인하는 퍼즈테스팅 방식을 시도하였고, 그 결과 NMDA 구조를 이용하여 설정과 실제 네트워크의 상태가 다를 수 있는 문제나 도스(Denial of Service)공격이 가능함을 밝혀냈다.

STS [17] 는 SDN 컨트롤러의 취약점을 찾고, 이에 대한 효율적인 분석을 위하여 퍼즈테스팅을 트러블슈팅(Troubleshooting) 시스템에 도입하였다. 특별히 버그를 찾기 위하여 품질보증(QA: Quality Assurance) 테스트 환경과 비슷한 환경을 만들어 SDN 컨트롤러에 대한 취약점을 찾고 트레이스(Trace)를 모으고자 하였다. 이를 위하여 기본적인 정보 전송을 담당하는 스위치와 호스트를 구성하여 테스트베드(Testbed)를 마련하였고, 각 연결 채널에서 무작위로 패킷에 대한 Drop 및 Delay 연산을 수행할 수 있도록 하여, 다양한 시나리오를 주입 할 수 있는 테스트 툴을 제안하였다.

그 외에도 Yao *et al.* 은 SDN 데이터평면에 있는 OpenFlow 스위치의 테스트를 위하여 새로운 형식모델(Fomal Model)과 함께 체계적인 블랙박스 테스팅 방법을 제안하였다 [18].

3.2 위협모델(Threat Model)에 따른 분류

SDN 시스템은 다양한 구성요소에서 삽입하는 이벤트의 순서와 메세지에 따라 다양한 시나리오를 구상할 수 있게 된다. 즉, 이에 대한 테스트를 위해서는 *시나리오 기반 퍼즈테스팅* 의 특성을 가지고 있어야 한다(§3.3). 이를 위하여, 각 연구에서는 테스트환경에 부합하는 위협모델을 먼저 제시하는 구조를 가지고 있다. 위협모델에서는 어떤 구성요소가 공격 및 테스트를 위하여 이용가능한지 가정해 놓고, 이들을 이용하여 퍼즈테스팅을 진행한다.

<표 1> SDN 퍼즈테스팅 툴의 위협모델 비교

Fuzzer Name	SDN Controller	SDN Switch	SDN Application	Host	Channel
DELTA [1]	✗	✗	✓	✓	✓
BEADS [2]	✗	✓	✗	✓	✗
AIM-SDN [3]	✗	✓	✓	✓	✗
STS [17]	✗	✓	✗	✓	✓

본 연구에서는 대표적인 SDN 퍼즈테스팅 툴의 위협모델을 비교하였다<표 1>. 각 컬럼에 대한 마크(✗ 또는 ✓)는 어느 구성요소를 활용하여 퍼즈테스팅을 진행하는지 나타낸다. 즉, ✓표시는 해당 요소가 이미 공격자에 의하여 장악되었기 때문에 테스트를 위하여 이용이 가능하다는 것을 의미하고, ✗표시는 퍼즈테스트의 대상이 될 수 있음을 의미한다.

볼 수 있듯이, 모든 툴의 위협모델에서 SDN 컨트롤러는 테스트의 대상이 됨을 알 수 있다. SDN 컨트롤러는 중앙집권화된 제어평면에서도 가장 중요한 역할을 하므로, 이에 대한 취약점 탐지가 매우 중요함을 내포하고 있다. 또한, 각 툴이 단 하나의 구성요소

<표 2> SDN 퍼즈테스팅 툴의 취약점 및 공격 탐지 기법 비교

Fuzzer Name	Controller			Application	Switch / Network				
	Reachability	Performance	Storage Poisoning	Reachability	Reachability	Performance	Forwarding Rule Poisoning	Host Reachability	Error Packet
DELTA [1]	✓	✓	✓	✓	✓	✓	✗	✓	✓
BEADS [2]	✗	✓	✗	✗	✓	✓	✓	✗	✓
AIM-SDN [3]	✗	✗	✓	✗	✗	✗	✓	✗	✓
STS [17]	✓	✗	✗	✓	✓	✗	✓	✓	✗

만을 이용하여 퍼즈테스트를 진행하는 것이 아니라, 여러 개의 구성요소 이벤트를 중첩시켜 테스트를 진행하는 것을 볼 수 있다.

3.3 취약점 및 공격 탐지 기법에 따른 분류

본 장에서는 각 퍼즈테스팅 도구가 SDN 시스템에 미칠 수 있는 보안적 영향(Security Impact)을 알기 위하여, 테스트과정에서 어떤 방식으로 취약점 및 공격을 탐지하는지 비교한다.

<표 2>는 각 SDN 퍼즈테스팅 툴의 취약점 및 공격을 탐지하는 기법을 비교하여 나타내고 있다. 각 컬럼에 대한 마크는 해당 툴이 유효한 공격을 탐지하기 위해서 SDN 의 어떠한 구성요소 및 상태를 확인하는지를 나타낸다. 즉, ✓표시는 공격에 대한 유효성 검사를 위하여 툴이 생성한 시나리오마다 해당 요소를 평가한다는 것을 의미하고, ✗표시는 해당 요소를 무시한다는 것을 의미한다.

<표 2>의 컬럼의 Reachability, Performance, Poisoning, Error Packet 에 대한 의미는 다음과 같다.

Reachability. 해당 요소에서 크래쉬(Crash)가 나서 종료가 됐거나 통신 채널(Channel)이 끊어져서, 본래 연결되어야 할 다른 구성요소로부터 접근이 불가능 한 경우(e.g. 컨트롤러내에서의 크래쉬로 인하여 어플리케이션에서 접근이 불가능한 경우).

Performance. 해당 요소의 작업량(Workload) 증가 등으로 인하여 성능이 평소에 비해 낮아지는 경우(e.g. 컨트롤러에서 측정한 지연 속도(Latency)나 처리량(Throughput)이 현저하게 낮아지는 경우).

Poisoning. 해당 요소의 내부 저장공간에 저장되어 있는 설정데이터가 바뀌어서 네트워크 상태에 영향을 주는 경우(e.g. 스위치에 있는 전달규칙(Forwarding Rule)이 바뀌어 전체 토폴로지(Topology)에 영향을 주는 경우).

Error Packet. Reachability 와 연관이 된 요소로, 해당 요소에 요청패킷을 보냈을 때, 처리하지 못하여 오류메세지가 응답으로 온 경우(e.g. 컨트롤러에서 변형된 메세지를 스위치에 보낼 경우, 파싱(Parsing)에 실패하여 에러메세지로 응답을 하는 경우).

볼 수 있듯이, SDN 퍼즈테스팅 툴에서 다양한 시나리오를 만들어 내기 때문에, 일반적인 퍼즈테스팅의 크래쉬 및 새니타이저(Sanitizer)만을 활용하는 취약점 감지 방법을 적용하지 않는다. 대부분의 SDN 퍼즈테스팅 툴은 컨트롤러, 어플리케이션, 스위치 및 네트워크의 구성요소와 상태를 관찰함으로써 공격의 성공여부를 찾아내는 접근방법을 이용한다.

3.4 연구동향

본 장에서는 기존의 SDN 퍼즈테스팅 연구들이 가지는 특성과 동향을 정리한다.

시나리오 기반 퍼즈테스팅. SDN 시스템은 복잡한 분산 환경과 계층 시스템 및 프로토콜 구조를 가지고 있다. 따라서, 현존하는 연구에서는 프로그램 내부에 있는 하나의 실행 진입점(Entry Point)에 무작위 인풋을 집어넣는 일반적인 퍼즈테스팅의 방법을 SDN 시스템에 그대로 적용시키지 않고 있다. 마찬가지로, 테스트의 결과를 확인하고 재생산(Reproduce) 하는 방법에 대하여서, 크래쉬 및 새니타이저를 이용하는 일반적인 방식 또한 고수하지 않는 추세이다. 퍼즈테스팅 기법을 SDN 에 적용시키기 위하여 대부분의 연구에서는 (§3.2)에서 언급하였던 위협모델을 제시하고, 이에 맞는 시뮬레이션 환경에서 여러가지 이벤트를 결합시켜 의미론적 공격 시나리오를 생성하는 방식을 취한다.

모델 기반(Model-based) 퍼즈테스팅. 대부분의 연구에서는 *시나리오 기반 퍼즈테스팅* 과 연관되어 모델 및 문법을 기반으로 하는 퍼즈테스팅 기법을 이용한다. 특별히, SDN 시스템은 OpenFlow 프로토콜과 RESTful API 를 이용하기 때문에, 이들의 구문론적, 의미론적 속성과 복잡한 메세지의 구조를 이해해야만 SDN 과 연관된 구성요소를 대상으로 퍼즈테스팅 기법을 적용시킬 수 있다. 예를 들어, DELTA [1] 에서는 End-to-End 통신의 흐름을 파악하기 위하여 OpenFlow 프로토콜의 상태도(State Diagram)를 이용하였고, 메세지의 필드(Field)를 조작하기 위해서 필드의 구문론적 정보를 이용하였다. 또한, BEADS [2] 에서는 패킷 필드의 의미론적 속성까지 이용하여 (e.g. 어떤 필드에 IP 주소나 Port 번호가 기입되는지 인지하는 것) 의미상은 유효하지만 문제가 되는 필드를 조사 할 수 있도록 하였다.

블랙박스(Black-box) 기반 퍼즈테스팅. ONOS [20],

OpenDaylight(ODL) [21], Floodlight [22] 과 같이 많은 SDN 구성요소들의 소프트웨어가 개방화됨에도 불구하고, 상용화된 SDN 시스템과 펌웨어의 소스코드를 얻기 힘든 환경을 고려하여, 대부분의 연구가 프로그램 내부의 소스코드를 보지 않고 분석을 진행하는 블랙박스 기반의 퍼즈테스팅에 초점을 맞추고 있다.

4. 결론 및 제언

본 논문에서는 SDN 시스템을 대상으로 하는 퍼즈테스팅의 필요성을 살펴보고, 기존의 연구들을 분석함과 동시에 그들이 공통적으로 가지는 특성을 조사하였다.

SDN 시스템이 점차 네트워크 인프라로 구축되고 상용화되는 시점에서, 이에 대한 효율적이고 자동화된 취약점 분석과 탐지는 필수적인 요소가 될 것이다. 앞으로도 더 많은 똑똑한(Smart) 방식을 활용하고, 쉽게 적용이 가능한 SDN 퍼즈테스팅 연구가 발현될 것을 기대한다.

Acknowledgement

이 논문은 2018 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2018-0-00254, SDN 보안 기술개발)

참고문헌

[1] Lee, S., Yoon, C., Lee, C., Shin, S., Yegneswaran, V., and Porras, P. A. “DELTA: A Security Assessment Framework for Software-Defined Networks,” in *Proceedings of the Network and Distributed System Security Symposium*, 2017.

[2] Jero S., Bu X., Nita-Rotaru C., Okhravi H., Skowyra R., Fahmy S. “BEADS: Automated Attack Discovery in OpenFlow-Based SDN Systems,” in *International Symposium on Research in Attacks, Intrusions, and Defenses*, 2017.

[3] Dixit, V. H., Doupé, A., Shoshitaishvili, Y., Zhao, Z., & Ahn, G. J. “AIM-SDN: Attacking Information Mismanagement in SDN-datastores,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[4] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... and Turner, J. “OpenFlow: enabling innovation in campus networks,” in *ACM SIGCOMM Computer Communication Review*, 2008.

[5] Xu, L., Huang, J., Hong, S., Zhang, J., & Gu, G. “Attacking the brain: Races in the sdn control plane,” in *Proceedings of the USENIX Security Symposium*, 2017.

[6] D. Kreutz, F. M. V. Ramos, and P. Verissimo. “Towards secure and dependable software-defined networks,” in *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013.

[7] Shin, S., and Gu, G. “Attacking software-defined networks: A first feasibility study,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.

[8] Sun, S., Gong, L., Rong, B., & Lu, K. “An intelligent SDN framework for 5G heterogeneous networks,” in *IEEE Communications Magazine*, 2015.

[9] 이승익, 이종화, 신명기, 김형준, & 손승원. (2014). 스마트인터넷을 위한 SDN 및 NFV 표준기술 동향 분석.

[10] Barton P Miller, Louis Fredriksen, and Bryan So. “An empirical study of the reliability of UNIX utilities,” in *Communications of the ACM*, 1990.

[11] AFL-CVE, https://github.com/mrash/afl-cve.

[12] Yun, I., Lee, S., Xu, M., Jang, Y., and Kim, T. “QSYM: A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing,” in *Proceedings of the USENIX Security Symposium*, 2018.

[13] Han, H., and Cha, S. K. “IMF: Inferred Model-based Fuzzer,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[14] Rawat, S., Jain, V., Kumar, A., Cojocar, L., Giuffrida, C., and Bos, H. “Vuzzer: Application-aware evolutionary fuzzing,” in *Proceedings of the Network and Distributed System Security Symposium*, 2017.

[15] OFTest: OpenFlow Switch Test Framework, https://github.com/floodlight/oftest.

[16] Florence: SDN Security Test Framework, https://github.com/snrism/florence-dev.

[17] Scott, C., Wundsam, A., Raghavan, B., Panda, A., Or, A., Lai, J., ... and Acharya, H. B. “Troubleshooting blackbox SDN control software with minimal causal sequences,” in *ACM SIGCOMM Computer Communication Review*, 2015.

[18] Yao, J., Wang, Z., Yin, X., Shiyz, X., and Wu, J. “Formal modeling and systematic black-box testing of sdn data plane,” in *Proceedings of the IEEE International Conference on Network Protocols*, 2014.

[19] Canini, M., Venzano, D., Peresini, P., Kostic, D., and Rexford, J. “A NICE way to test OpenFlow applications,” in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[20] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., ... and Parulkar, G. “ONOS: towards an open, distributed SDN OS,” in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014.

[21] Medved, J., Varga, R., Tkacik, A., and Gray, K. “Opendaylight: Towards a model-driven sdn controller architecture,” in *IEEE International Symposium on*, 2014.

[22] Floodlight: SDN OpenFlow controller, https://github.com/floodlight/floodlight.