

## K-ONE 기술 문서 #46

# Security-Mode ONOS for Multi-layer Access Control

Document No. K-ONE #46

Version 0.3

Date 2019-11-25

Author(s) 이수열, 강희도, 강한이

■ 문서의 연혁

버전	날짜	작성자	내용
0.1	2019. 04. 14	강희도	Multi-layered Security-Mode ONOS 기본내용 수정 작성
0.2	2019. 11. 10	이수열	Multi-layered Security-Mode ONOS를 통한 실제 애플리케이션 공격 방어 및 관련 부분 작성
0.3	2019. 11. 25	강한이	오탈자 수정

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

소프트웨어 정의 네트워킹은 기존 네트워킹 아키텍처와 달리, 컨트롤 플레인과 데이터플레인을 분리시켜 모든 연산을 중앙의 컨트롤러가 담당하게하고 데이터플레인은 컨트롤러로부터 받은 rule에 따라 단순히 패킷을 포워딩시키는 차세대 네트워킹 아키텍처이다. 이러한 아키텍처의 장점은 중앙의 컨트롤러가 모든 네트워킹을 관리하기 때문에 관리의 효율성이 매우 크고, 여러 네트워크 기능들을 애플리케이션 형태로 컨트롤러위에서 실행시킴으로써 기존 네트워크 구조에서는 할 수 없었던 혁신적인 기능을 구현할 수 있게 해준다. 컨트롤러는 하나의 네트워크 소프트웨어로써 다양한 API들을 제공하며 소프트웨어 정의 네트워킹 네트워크를 관리하는 핵심 기능을 제공한다. 현재 오픈소스 기반의 컨트롤러들, 예를 들어 ONOS, Floodlight, Opendaylight 등이 활발하게 개발되고 있는 상황이다. 이러한 컨트롤러들은 오픈소스 기반이기 때문에 누구든 자신들만의 애플리케이션들을 구현하여 배포 할 수 있으므로, 혁신적인 애플리케이션이 개발될 수 있는 환경을 제공해준다.

이러한 장점에 반해 큰 단점중 하나는 배포된 SDN 애플리케이션에 악성코드나 버그가 포함되어있을 수 있다는 것이다. 현재 오픈소스기반의 컨트롤러들의 아키텍처를 보게 되면 SDN애플리케이션 하나가 컨트롤러에서 제공하는 모든 API들과 시스템 API들까지 모두 제한 없이 접근이 가능하다. 이로 인해 SDN 애플리케이션은 자신이 원하는 행위, 예를 들어 실행되고 있는 SDN보안애플리케이션들을 강제로 종료시켜버리는 행위들을 할 수 있다.

본 기술문서에서는, 악성코드나 버그가 포함되어있는 SDN 애플리케이션의 행위를 제한하기 위해 ONOS 컨트롤러위에 액세스컨트롤 메커니즘을 구현한 Security-Mode ONOS에 대한 소개와, Security-Mode ONOS의 최신 기능에 대해 설명한다. Security-Mode ONOS를 사용하면 안전한 소프트웨어 정의 네트워킹 환경을 구축할 수 있다.

## Contents

### K-ONE #46. Security-Mode ONOS with Host-level Access Control

1. 서론 .....	7
1.1. 배경지식 .....	7
1.2. SDN 컨트롤러 아키텍처의 문제점 .....	8
1.3. 악성애플리케이션을 이용한 SDN컨트롤러 공격예제 .....	9
1.4. 악성애플리케이션에 대한 기존 연구 .....	12
2. 본론 .....	14
2.1. Security-Mode ONOS 란? .....	14
2.2. Security-Mode ONOS 퍼미션 모델 .....	14
2.3. Security-Mode ONOS 디자인 .....	26
2.4. Security-Mode ONOS 구현 .....	27
2.5. Security-Mode ONOS 추가기능 .....	28
2.6. 일반적인 Security-Mode ONOS 사용법 .....	31
2.7. 가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS 사용법 .....	33
2.8. 다중 레이어에서의 접근 제어를 통합한 Multi-layered Security-Mode ONOS .....	37
2.9 Multi-layered Security-Mode ONOS를 통한 실제 애플리케이션 공격 방어 .....	38
2.10 호스트 레벨 접근제어 기능을 추가한 Security-Mode ONOS 사용법 .....	46
3. 결론 .....	52

## 그림 목차

그림 1 소프트웨어 정의 네트워킹 구조 .....	7
그림 2 오픈소스 기반 SDN 애플리케이션 배포과정 .....	8
그림 3 ONOS 컨트롤러 아키텍처 .....	9
그림 4 컨트롤러 강제종료 공격 시나리오 예제 .....	10
그림 5 컨트롤러 강제종료 공격으로 인해 발생하는 문제 .....	11
그림 6 컨트롤러 정보 변경 공격 시나리오 .....	11
그림 7 컨트롤러 정보 변경으로 인해 발생하는 문제 .....	12
그림 8 Security-Mode ONOS 퍼미션 모델 플로우차트 .....	15
그림 9 SM ONOS-VN의 동작 시나리오 .....	19
그림 10 SM ONOS-VN 퍼미션 모델 플로우 차트 .....	21
그림 11 SM ONOS-VN이 추가된 권한 파일 형식 .....	23
그림 12 SM-ONOS HAC 애플리케이션 액세스 컨트롤 예시 .....	24
그림 13 Host-level Access Control 기본 메커니즘 .....	25
그림 14 Security-Mode ONOS 디자인 .....	26
그림 15 Permission gap을 활용한 보안강화기능 결과화면 .....	29
그림 16 Security-Mode ONOS unit test 케이스 예시 .....	30
그림 17 Policy 파일 형식 예제 .....	31
그림 18 퍼미션 부여되지 않은 애플리케이션을 활성화시키려는 경우 .....	32
그림 19 Security-Mode ONOS review 명령어 사용법 .....	32
그림 20 Security-Mode ONOS accept 명령어 사용법 .....	33
그림 21 Policy 파일 형식 예제 .....	34
그림 22 퍼미션을 부여하지 않은 상태에서의 활성화 경고 메시지 .....	35
그림 23 최종적으로 허가된 fwd 애플리케이션의 퍼미션 목록 .....	36
그림 24 가상 네트워크 접근제어 기능 실행 방법 .....	36
그림 25 Security-Mode ONOS 가상네트워크 접근 제어 테스트 결과 1 .....	37
그림 26 Security-Mode ONOS 가상네트워크 접근 제어 테스트 결과 2 .....	37
그림 27 Multi-layered Security Mode ONOS 접근제어 개요 .....	38
그림 28 디바이스의 포트를 disable하는 코드 .....	39
그림 29 DDos 방어 애플리케이션의 초기 퍼미션 .....	40
그림 30 INTENT : 가장 추상적인 프로그래밍 모델 .....	41
그림 31 호스트 간의 Intent를 설치하는 코드 .....	41
그림 32 DDos 방어 애플리케이션의 퍼미션 accept .....	42

그림 33 DDoS 방어 애플리케이션의 활성화 .....	42
그림 34 STORAGE_WRITE 접근 거부 오류 .....	43
그림 35 DDoS 방어 애플리케이션에 STORAGE_WRITE 권한 추가 .....	43
그림 36 INTENT_WRITE 접근 거부 오류 .....	44
그림 37 INTENT_WRITE에 포함되어 있는 API들 .....	44
그림 38 INTENT 설치 실패 .....	44
그림 39 DDoS 방어 애플리케이션에 필요한 권한 모두 추가 .....	44
그림 40 INTENT 설치 성공 .....	45
그림 41 VNET 4번 id에대한 READ 권한 부여 실패 .....	46
그림 42 VNET_READ_4 접근 거부 오류 .....	46
그림 43 Host-level Access Control 애플리케이션 ADD 명령어 예시 .....	47
그림 44 Host-level Access Control 애플리케이션 REMOVE 명령어 예시 .....	47
그림 45 Host-level Access Control 애플리케이션 GET 명령어 예시 1 .....	48
그림 46 Host-level Access Control 애플리케이션 GET 명령어 예시 2 .....	48
그림 47 Host-level Access Control 애플리케이션 Web 서비스 .....	48
그림 48 Host-level Access Control 애플리케이션 권한 부여 테스트 .....	49
그림 49 Host-level Access Control 애플리케이션 권한 부여 테스트 결과 1 .....	49
그림 50 Host-level Access Control 애플리케이션 권한 부여 테스트 결과 2 .....	50
그림 51 Host-level Access Control 애플리케이션 권한 제거 테스트 .....	50
그림 52 Host-level Access Control 애플리케이션 권한 제거 테스트 결과 1 .....	50
그림 53 Host-level Access Control 애플리케이션 권한 제거 테스트 결과 2 .....	51

## 표 목차

표 1 Security-Mode ONOS 지원 Northbound API permission Type .....	18
표 2 SM ONOS-VN의 권한이 추가된 전체 권한 목록. ....	20
표 3 JUnit 기본 주석(Annotation) 예시 .....	30

## K-ONE #46. Security-Mode ONOS for Multi-layer Access Control



## 1. 서론

본 서론에서는, Security-Mode ONOS를 이해하기 위한 배경지식과 현재 ONOS 컨트롤러가 가지고 있는 문제점, 그리고 실제 악성 애플리케이션을 이용한 공격예제를 통해 Security-Mode ONOS의 필요성에 대해 서술하며, 악성 애플리케이션에 의해 발생할 수 있는 문제점을 방지할 수 있는 기존 연구들을 소개한다.

### 1.1. 배경지식

소프트웨어 정의 네트워킹은 기존 네트워킹 구조와는 달리 컨트롤플레인과 데이터 플레인을 분리하는 차세대 네트워킹 아키텍처로써 학계와 산업계의 큰 관심을 받고 있다. 소프트웨어 정의 네트워킹의 구조는 그림 1과 같이 중앙 집중화 된 하나의 컨트롤러가 모든 네트워크를 관리하는 구조이다. 소프트웨어 정의 네트워킹에서 컨트롤러는 하나의 네트워크 소프트웨어로써 핵심기능들이 Core에서 제공되며 이 Core는 다양한 애플리케이션 개발을 위한 여러 서비스들을 통해 API들을 제공한다.

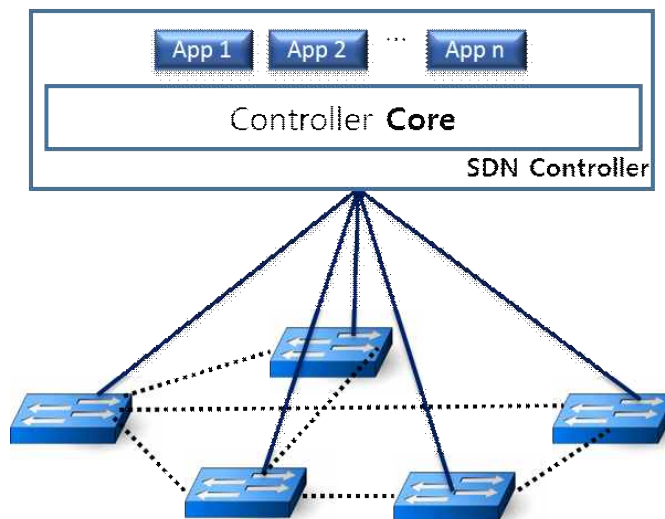


그림 1 소프트웨어 정의 네트워킹 구조

이 컨트롤러는 현재 오픈소스 기반으로 활발하게 개발이 진행되고 있으며, 대표적인 오픈소스 기반의 컨트롤러들은 ONOS[6], Opendaylight[5], Floodlight[11]가 있다. 오픈소스 기반의 컨트롤러들은 API들을 공개함으로써 어떠한 소프트웨어정의 네트워킹 애플리케이션 개발자들도 자신들만의 혁신적인 새로운 애플리케이션들을 개발하고 마음껏 배포할 수 있는 환경을 제공해준다. 일반적인 오픈소스 기반의 SDN 애플리케이션 배포과정은 그림 2와 같이 SDN 애플리케이션 개발자가 애플리케이션을 만들고 SDN 앱스토어를 통해 배포하는 과정을 거치게 된다.

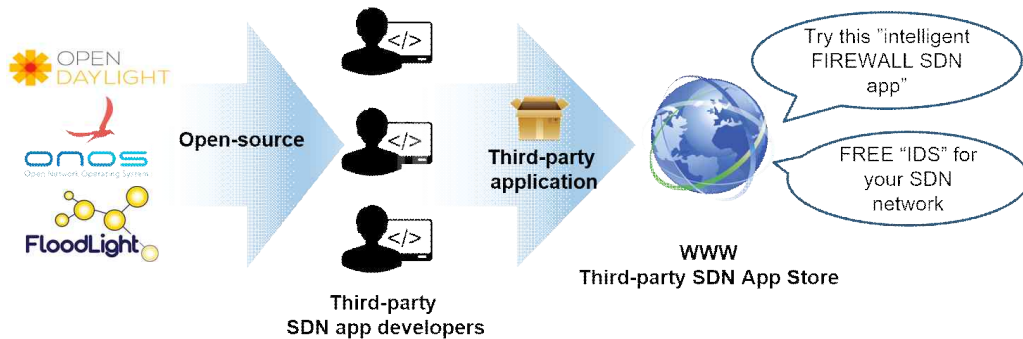


그림 2 오픈소스 기반 SDN 애플리케이션 배포과정

이러한 오픈소스 기반의 컨트롤러들은 혁신적인 애플리케이션이 개발될 수 있는 환경을 제공해줄 수 있다는 큰 장점에 반해, 큰 단점이 존재하게 되는데, 인증되지 않은 개발자들도 자신들의 애플리케이션을 만들어 배포할 수 있다는 점에서 문제가 발생하게 된다. 일반적인 윈도우나 운영체제위에서 실행되는 애플리케이션들도, 인증되지 않은 개발자들이 운영체제에서 제공하는 API들을 이용하여 자신들만의 애플리케이션을 만들고 배포 할 때 악성코드를 심어놓고 배포 할 수가 있는 문제점이 똑같이 존재한다. 따라서 사용자가 다운받고 실행 할 때 굉장히 주의를 기울여야 하는데, 현재 윈도우 운영체제 같은 경우 애플리케이션의 악성코드를 검사하고 치료하는 백신 등이 많이 연구되어졌다. 윈도우가 아닌 SDN의 애플리케이션이 배포 되는 과정과 조금 더 비슷한 환경을 예를 들어 설명하자면, 안드로이드 플랫폼의 애플리케이션이다. 안드로이드 플랫폼위에서 돌아가는 애플리케이션들도, 인증되지 않은 개발자들도 안드로이드 운영체제에서 제공하는 API들을 통해 자신들만의 애플리케이션을 만들고 앱스토어를 통해 배포할 수가 있다. 이에 따라 배포된 애플리케이션들 중 악성코드가 포함되어 있을 수 있는데, 안드로이드에서는 백신도 존재하지만 악성 애플리케이션들의 행위를 제한하기 위한 액세스컨트롤 메카니즘이 존재한다. 안드로이드 같은 운영체제에서는 이러한 보호 메카니즘이 존재하는 반면, 현재 개발되어지고 있는 SDN 컨트롤러들은 악성 애플리케이션으로부터 컨트롤러를 보호하는 보호 메카니즘 자체가 존재하지 않으며 아키텍처 자체가 취약한 상황이다.

## 1.2. SDN 컨트롤러 아키텍처의 문제점

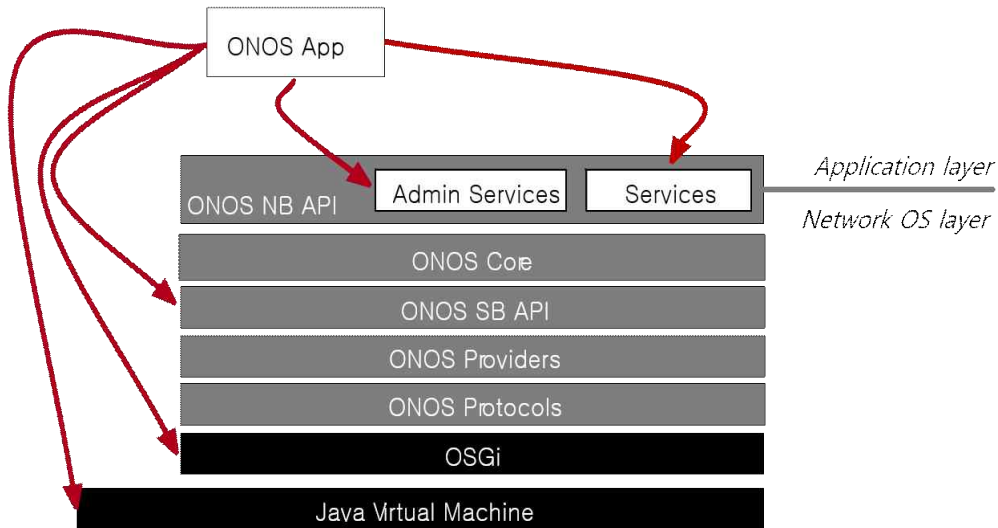


그림 3 ONOS 컨트롤러 아키텍처

오픈소스 기반의 SDN 컨트롤러들이 취약한 이유를, 현재 오픈소스 기반의 SDN 컨트롤러들 중 가장 활발하게 개발되어지고 있는 컨트롤러인 ONOS의 아키텍처를 예로 들어 설명해보겠다. 현재 ONOS 컨트롤러의 아키텍처는 그림 3과 같다. 그림 3에서 볼 수 있듯이, 하나의 ONOS 컨트롤러는 JVM과 OSGi 위에서 실행되며, 그 위에서 각종 ONOS 애플리케이션들을 실행시켜 네트워크 기능들을 제공한다. 이 ONOS 애플리케이션은 ONOS 컨트롤러와 같은 하나의 JVM, OSGi 위에서 동작하는 것을 볼 수 있는데, 이는 하나의 애플리케이션이 JVM을 강제종료 시켜버리면 전체 컨트롤러가 죽어버리는 굉장히 큰 문제를 발생 시킬 수 있다. ONOS 애플리케이션은 현재 OSGi와 JVM에서 제공하는 API들에 제한 없이 접근이 가능하여 시스템 콜을 호출해 악성행위를 할 수 있으며 ONOS 코어에서 제공하는 각종 서비스들에 제한 없이 접근이 가능하여 서비스들에서 제공하는 API들을 이용하여 컨트롤러에 악영향을 미치는 행위를 할 수 있다. ONOS 컨트롤러를 포함한 대부분의 오픈소스 기반의 SDN 컨트롤러들은 이와 똑같은 문제를 가지고 있는데, 이러한 취약성에 대해 지적인 국제적인 여러 연구들[1][9][10][8] 이 존재한다.

### 1.3. 악성애플리케이션을 이용한 SDN컨트롤러 공격예제

본 1.3절에서는, 1.2절에서 서술한 현재 SDN 컨트롤러 아키텍처로 인해 발생하는 취약성에 대해 지적인 대표적인 논문[1]에서 소개한 취약점 내용을 바탕으로 실제 악성애플리케이션을 이용한 SDN 컨트롤러 공격시나리오를 설명함으로써 SDN 앱스토어로 부터 다운받은 애플리케이션이 악성일 때, SDN 컨트롤러에 어떠한 영향을 미칠 수 있는지 기술한다.

## o 컨트롤러 강제종료 공격 시나리오

SDN 앱스토어를 통해 그림 4와같이 IDS 서비스라고 소개되어있는 앱을 다운받았다고 가정해보자. 애플리케이션의 소개부분에서, 검은색으로 써져있는 글만 본다면 정상적인 IDS 애플리케이션이고 이 애플리케이션을 다운받은 유저는 아무런 의심 없이 다운받고 실행하게 된다.

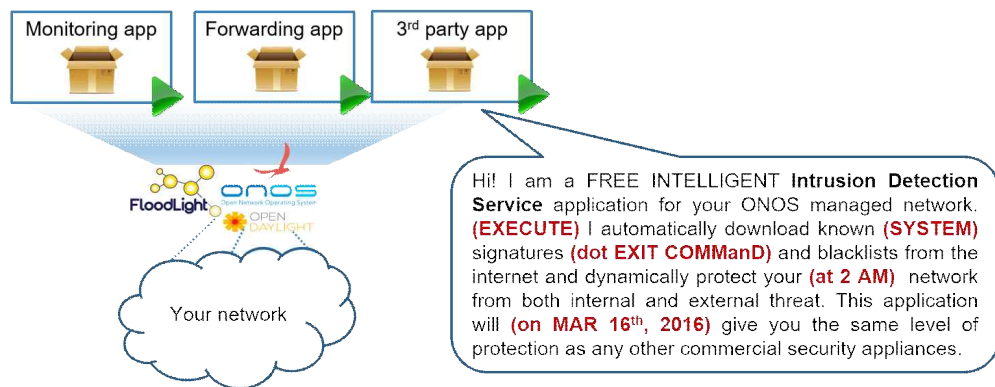


그림 4 컨트롤러 강제종료 공격 시나리오 예제

하지만, 이 애플리케이션이 사실은 소개에 검은색으로 써져있는 글에 대한 행위를 하는 것뿐만 아니라, 빨간색으로 써져있는 내용에 해당되는 행위 즉 정해진 날짜에 JVM을 종료시키는 System.exit() API를 호출한다고 가정해보자. 이러한 경우 다운 받은 이 애플리케이션을 컨트롤러에 실행시키게 되면 한동안은 정상적인 IDS의 기능을 하다가, 2016년 3월16일 오전2시에 System.exit() API를 호출하게 된다. 이러한 경우, 기존 연구[1]에서 보였듯이 SDN 애플리케이션이 컨트롤러와 같은 JVM위에서 실행되므로 SDN 애플리케이션 하나가 JVM을 종료시키는 System.exit() API를 호출하게 되면 그림 5와같이 SDN 컨트롤러 자체가 죽게 되고, 결과적으로는 해당 컨트롤러가 관리하는 전체 네트워크가 마비되는 현상이 발생되게 된다.

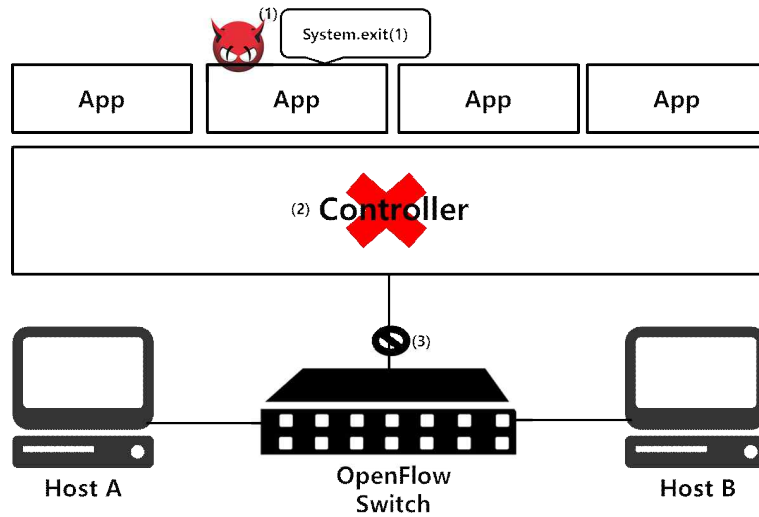


그림 5 컨트롤러 강제종료 공격으로 인해 발생하는 문제

#### o 컨트롤러 정보 변경 공격 시나리오

SDN 앱스토어를 통해 위의 공격시나리오와 똑같이 IDS 애플리케이션이라고 소개되어있는 앱을 다운받았다고 가정해보자. 그림 6에서 보이는 것과 같이 애플리케이션의 소개부분에서, 검은색으로 써져있는 글만 본다면 위의 시나리오와 똑같이 IDS 애플리케이션이고 이 애플리케이션을 다운받은 유저는 아무런 의심 없이 다운받고 실행하게 된다.

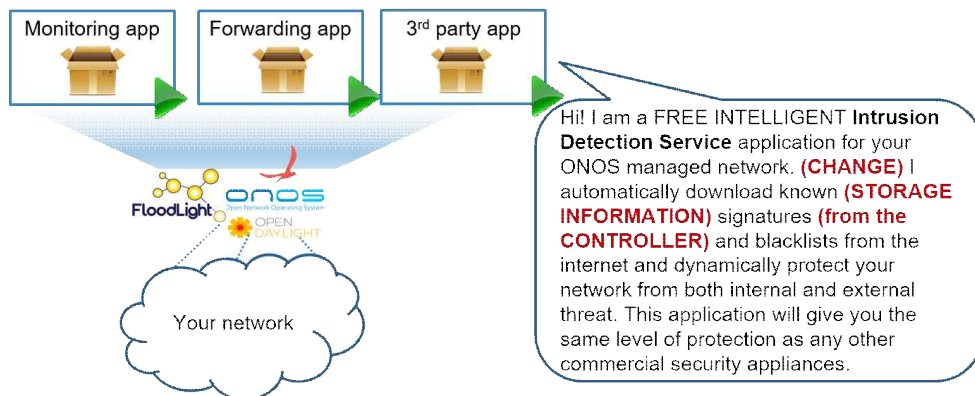


그림 6 컨트롤러 정보 변경 공격 시나리오

하지만 이 애플리케이션은 사실상 빨간색으로 써져있는 부분의 행위, 즉 컨트롤러에 저장되어있는 네트워크 정보들을 임의로 바꿔버리는 행위를 하는 애플리케이션이었다고 가정해보자. SDN 컨트롤러 위에서 실행되는 애플리케이션들은 SDN 컨트롤러 스토리지에 저장되어있는 정보를 이용하여 연산을 하고 데이터프레임에 물을

내리기 때문에, 다운받은 애플리케이션이 컨트롤러가 가지고 있는 네트워크 정보를 그림 7과 같이 임의로 바꿔버린다면 다른 애플리케이션들은 실제 네트워크정보가 아닌 다운받은 악성 애플리케이션이 바꾼 네트워크정보를 이용하여 연산하게 되므로 잘못된 연산을 수행하게 되고 이에 따라 해당 컨트롤러가 관리하는 네트워크가 정상적으로 동작할 수 없도록 만든다.

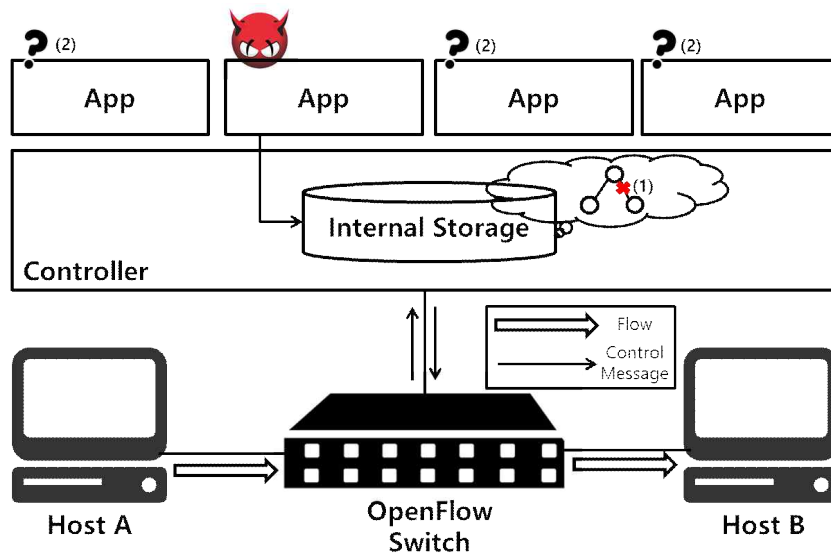


그림 7 컨트롤러 정보 변경으로 인해 발생하는 문제

#### 1.4. 악성애플리케이션에 대한 기존 연구

악성 애플리케이션에 의해 발생하는 문제점을 사전에 방지하기 위해서는, 다운로드 받은 애플리케이션을 실행시키기 전에 다운로드 받은 애플리케이션이 어떤 행위를 하는지에 대한 사전분석이 필요하다. 사전분석은 기본적으로 소스코드를 보며 해당 애플리케이션이 어떠한 API를 호출하고 어떠한 행위를 하는지 줄 단위로 수작업 분석해야 한다. 이는 굉장히 시간이 많이 걸리는 작업이고, Third-party 애플리케이션의 경우 애플리케이션의 소스코드를 공개하지 않고 실행파일만 배포하기 때문에 분석하기가 어렵다.

애플리케이션 행위분석에 있어서 사람의 수작업 분석을 줄여주기 위해, 애플리케이션 정적분석과 동적 분석에 대한 여러 연구가 진행되어져 왔다. 정적 분석의 경우, 애플리케이션을 직접 실행시키지는 않고 애플리케이션이 어떠한 API를 호출하는지 어떠한 순서로 호출 하는지 등을 소스코드를 기반으로 자동으로 분석하여 행위를 분석해준다. 하지만 이는 여전히 소스코드가 필요하며 동적으로 코드가 생성되거나 하는 부분에서는 행위분석이 안되기 때문에 행위분석 결과가 정확하지 않다는 문제점을 가지고 있다. 이러한 한계점을 극복하기 위해, 동적 분석이 사용된다. 동적 분석은 실제 애플리케이션에 인풋 값을 넣어서 실행시키며 어떠한 행위를 하

는지 분석하는 기술이다. 동적 분석에서 가장 중요한 것은, 애플리케이션의 코드가 최대한 많이 실행되어야 정확한 행위분석이 가능하므로 애플리케이션 코드에 최대한 적합한 인풋 값들을 생성해내어 애플리케이션의 코드가 최대한 많이 실행되도록 (Code-Coverage) 하는 것이다. 애플리케이션에 적합한 인풋 값들을 생성해내는 데에는 symbolic-execution[14]이라는 기술이 사용되게 되는데, 이 symbolic execution은 cost가 너무 크며 인풋데이터를 사람이 수작업으로 symbolic하게 설정해줘야 하는 문제점이 있다. 또한 이러한 동적 분석과 정적분석을 이용한 행위분석은 유저가 행위 분석된 결과를 가지고 악성인지 아닌지 판단하거나 기존 악성 애플리케이션들을 분석한 결과를 기반으로 폴리시를 만들고 이를 기반으로 자동으로 판단하게 된다. 유저가 행위 분석된 결과를 분석하는 것은 실수가 포함될 수 있으므로 실수로 실행 시키게 되면 악성 애플리케이션의 행위를 제어하지 못하게 된다.<sup>1</sup>

SDN의 경우 아직까지 악성애플리케이션이 나타나지 않았기 때문에 직접 폴리시들을 정하고 동적 분석이나 정적분석을 이용하여 악성 애플리케이션에 대한 판단을 하는 것은 불가능하다. 이러한 문제점 때문에, 동적 분석과 정적분석을 이용하여 악성 SDN 애플리케이션에 의해 발생하는 문제점을 사전에 방지하기엔 충분하지 않다. 따라서 SDN 악성 애플리케이션의 행위를 사전에 방지할 수 있는 동시에 런타임으로 SDN 애플리케이션의 행위를 제한하는 메카니즘이 필요하다.



## 2. 본론

본 절에서는, SDN 악성 애플리케이션으로부터 SDN컨트롤러를 보호하는 메카니즘을 ONOS 컨트롤러 위에 구현한 Security-Mode ONOS에 대한 설명, 아키텍처와 함께 Security-Mode ONOS에 대한 사용법을 기술한다.

### 2.1. Security-Mode ONOS 란?

1.4절에서 기술하였듯이, SDN 악성 애플리케이션의 행위를 사전에 방지할 수 있는 동시에 런타임으로 SDN 애플리케이션의 행위를 제한하는 메카니즘이 필요하다. 악성 SDN 애플리케이션으로부터 SDN 컨트롤러를 보호하기 위해서는 악성애플리케이션을 사전에 한번 방지하면서도 동시에 런타임으로 애플리케이션의 행위를 제한하는 메카니즘은 안드로이드에서 제공하는 액세스컨트롤 메카니즘이다. 안드로이드 플랫폼에선, 개발자가 애플리케이션을 개발하고 나서 패키징 하여 앱스토어에 배포하기 전에 manifest 파일에 자신이 개발한 안드로이드 애플리케이션이 사용하는 퍼미션들을 적고 패키징 하여 배포하게 한다[7]. 안드로이드 앱스토어로 부터 유저가 이렇게 배포된 애플리케이션을 설치하려 하는 경우, 안드로이드 플랫폼은 애플리케이션이 어떠한 퍼미션을 사용하는지 유저에게 보여줌으로써 애플리케이션이 어떠한 행위를 하는지 알리고 설치할지 말 것인지에 대한 동의를 받게 된다. 유저가 동의를 하고나면 애플리케이션이 manifest 파일에 적혀있는 퍼미션들을 부여받고 설치되게 되는데, 만약 설치된 애플리케이션이 실행도중 부여된 퍼미션에 벗어나는 행위를 하게 되면 행위가 제한되게 되는 액세스컨트롤 메카니즘을 가지고 있다. 이러한 환경은 유저가 자신이 설치하는 애플리케이션이 어느 행위를 하는지에 대한 정보를 제공하고, 설치된 애플리케이션이 런타임에 자신에게 부여된 퍼미션에 해당되는 행위만 할 수 있게 제한함으로써 유저에게 안전성을 제공해주게 된다.

이러한 액세스컨트롤 메카니즘을 SDN에 적용한다면, 악성 SDN 애플리케이션을 SDN 앱스토어에서 다운받게 되어도, 유저는 다운받은 SDN 애플리케이션이 요청하는 퍼미션을 한번 보고 악성 SDN 애플리케이션에 대한 1차적 필터링이 가능하다. 또한 각 SDN 애플리케이션들이 자신에게 부여된 퍼미션에 해당하는 API만 사용할 수 있도록 제한하므로 동적으로 런타임에 코드를 생성하여 악성행위를 하는것에 대해서도 안전해진다. Security-Mode ONOS는 이러한 안드로이드의 액세스컨트롤 메카니즘을 ONOS에 맞게 적용한 것이다. Security-Mode ONOS는 기본적으로 OSGI Bundle단위로 퍼미션을 부여하여 액세스컨트롤을 하게 된다. Security-Mode ONOS에 대한 자세한 설명을 다음 2.2절부터 기술한다.

### 2.2. Security-Mode ONOS 퍼미션 모델



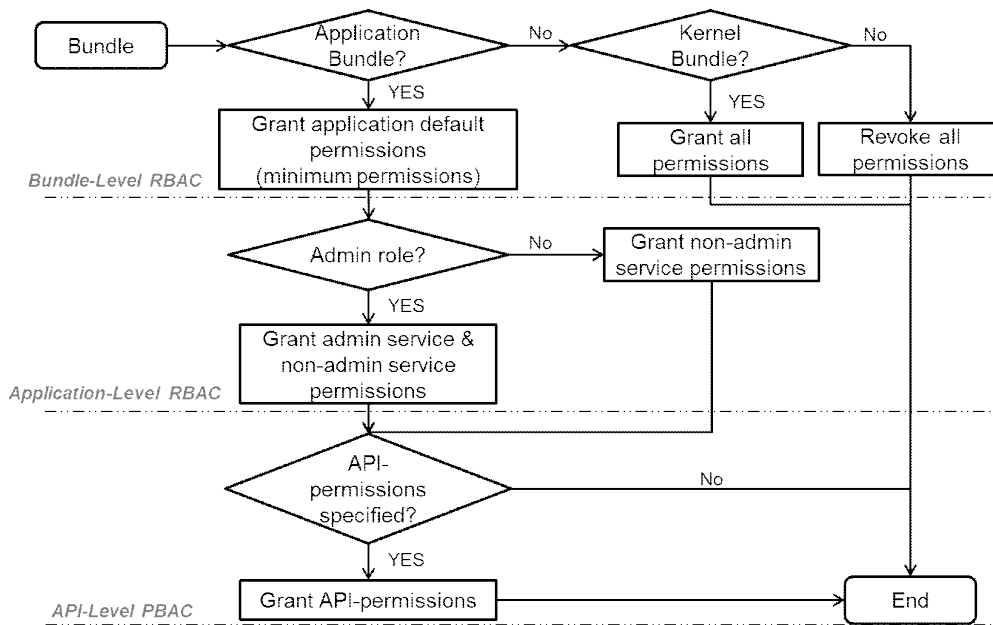


그림 8 Security-Mode ONOS 퍼미션 모델 플로우차트

Security-Mode ONOS는 안드로이드 시큐리티 메카니즘처럼, 개발자가 자신의 ONOS 애플리케이션을 개발하고 사용하는 퍼미션들을 app.xml에 명시하고 패키징 하여 배포해야만 한다. Security-Mode ONOS가 활성화된 상황에서 애플리케이션을 실행시키기 위해선, app.xml 파일에 애플리케이션의 퍼미션들이 작성되어있어야만 한다. 현재 Security-Mode ONOS의 퍼미션 모델은 다음과 같다. (1) Bundle Level Role based access control (2) Application Level Role based access control (3) API Level Permission based access control. 그림8 플로우차트에 Security-Mode ONOS 퍼미션 모델에 의해 액세스 컨트롤되는 순서가 정리되어져있다. 아래에서, 퍼미션 모델에 대한 설명과 이 퍼미션 모델에 의해 어떻게 액세스컨트롤이 이루어지는지에 대한 자세한 설명을 한다.

#### o Bundle Level Role based access control

ONOS컨트롤러는 현재 OSGI 프레임워크에서 apache-karaf[3] 를 이용하여 번들단위로, 애플리케이션을 포함한 여러 모듈들을 동적으로 실행시키고 종료시키는 기능을 제공한다. 따라서 ONOS컨트롤러에 액세스컨트롤을 적용하기 위해서는 이 번들단위의 액세스컨트롤을 지원해야 한다. Security-Mode ONOS에서는 1차적으로 이러한 번들 단위의 역할기반 액세스컨트롤을 다음과 같이 지원한다. 만약 번들이 애플리케이션 번들이 아닌 ONOS kernel에 관련된 번들일 경우 신뢰할 수 있는 번들이기 때문에 모든 API들을 사용할 수 있는 퍼미션을 해당 번들에게 부여해 준다. 반면 번들이 애플리케이션 번들일 경우, 신뢰 할 수 없는 번들이기 때문에 번들이 실행될 수 있는 최소한의 퍼미션만을 부여해주게 된다.

#### o Application Level Role-based Access Control

현재 ONOS의 Northbound에서 제공하는 서비스들을 보게 되면, 크게 AdminService와 일반 Service 두 가지 타입으로 나뉘어져 있다. AdminService의 경우, 컨트롤러가 가지고 있는 정보를 직접적으로 지우거나 수정할 수 있는 API들이 포함되어 있는 Service이다. 예를 들어, DeviceAdminService는 컨트롤러에서 논리적 디바이스 정보를 지워버릴 수 있는 API를 제공한다. 이러한 AdminService들은 ONOS 컨트롤러와 ONOS 컨트롤러가 관리하는 네트워크에 매우 큰 영향을 미칠 수 있는 서비스들이며 일반적인 애플리케이션들이 사용하면 안되는 API들이기 때문에 따로 분리되어 있다. 이러한 AdminService에서 제공하는 API들을 이용하면 매우 손쉽게 SDN 네트워크를 마비시킬 수 있는 위험성이 있다. 일반적인 Service들의 경우에는 대부분 컨트롤러에 저장되어 있는 정보를 읽어오거나, 컨트롤러에 정보를 쓰는 API들을 제공하는데, 이 API들은 일반적인 애플리케이션이 기능구현 내용에 따라 사용해야 하는 API들이지만, 이 API들을 이용하여서도 악성행위를 할 수 있는 위험성이 존재한다. 예를 들어, ONOS 컨트롤러위에 방화벽 같은 보안 애플리케이션을 실행시켜 네트워크에 보안서비스를 제공하고 있는 경우, 악성 애플리케이션이 FlowRuleService를 이용하여 보안 애플리케이션에서 내린 룰을 지워버리고 자신이 원하는 rule을 삽입할 수 있다. 이러한 경우 해당 ONOS 컨트롤러는 더 이상 보안 서비스를 제공하지 못하는 상황에 빠지게 된다.

엑세스컨트롤 퍼미션모델을 만들기 위해선 ONOS의 이러한 구조를 이해하고 고려해야만 하는데, AdminService들에서 제공하는 API들의 경우 절대 Third-Party 애플리케이션들이 사용해서는 안되는 API이고, 일반 Service들에 대해서는 Third-Party 애플리케이션들이 사용가능은하지만 fine-grained 엑세스컨트롤이 필요할 것이다. 따라서 Security-Mode ONOS에서는 bundle level access control에 이어, 2차적으로 application level의 역할기반 엑세스컨트롤을 적용하였다. Security-Mode ONOS에 적용된 application level 역할기반 엑세스컨트롤은, 번들이 ADMIN 역할일 경우 AdminService들에서 제공하는 모든 API와 일반 Service들에서 제공하는 모든 API들에 대해 접근할 수 있도록 하는 반면, 번들이 User역할일 경우 오직 일반 Service들에 대해서만 접근할 수 있도록 설계되어 있다.

#### o API Level Permission based Access Control

ONOS애플리케이션의 행위를 fine-grained하게 제한하기 위해서는 API 레벨의 퍼미션 기반 엑세스컨트롤이 필요하다. ONOS에 API 레벨 엑세스컨트롤을 적용하고자 할 때, 가장먼저 고려하였던 것은 어느 API까지 엑세스컨트롤을 지원해야 하는 것이었다. 현재 ONOS 애플리케이션이 쓸 수 있는 API들은 ONOS컨트롤러에서 제공하는 Northbound API들 뿐만 아니라 JAVA Native API, OSGI관련 API 들이

있다. Northbound API는 ONOS 컨트롤러에서 제공하는 API이니 당연히 액세스컨트롤을 적용시켜야하지만 JAVA Native API나 OSGI API는 액세스컨트롤은 적용여부를 생각해보아야 한다. 일반적인 ONOS 애플리케이션은 JAVA Native API나 OSGI API를 거의 사용할 일이 없다. 하지만 악성 ONOS 애플리케이션 같은 경우, JAVA Native API를 호출하여 소켓을 하나 열고, 토폴로지정보를 읽어서 만든 소켓을 통해 내보내는 행위 등을 할 수 있다. 이러한 문제를 사전에 완벽히 막기 위해서는 JAVA Native API와 OSGI API들 까지도 모두 액세스컨트롤 대상에 포함시켜야한다. 따라서 Security-Mode ONOS는 Northbound API, JAVA Native API, OSGI API들 까지를 모두 액세스컨트롤 대상으로 한다.

엑세스컨트롤을 위해서는 각 API들에 대한 퍼미션 타입을 정의해야하는데, JAVA나 OSGI같은 경우 이미 JAVA Security 플랫폼과 OSGI Security 플랫폼에서 크리티컬 API들에 대한 퍼미션 타입들이 정의되어 있으므로 새롭게 정의할 필요가 없다. 따라서 ONOS 컨트롤러의 Northbound API들에 대한 퍼미션타입을 새로 정의하여야 하는데, 퍼미션 타입은 안드로이드에서 정의한 퍼미션 타입[12]들과 비슷하게 크게 Resource + action으로 정의하였다. Resource는 API가 접근하는 리소스종류를 의미하는 것이며, action은 API가 리소스에 접근하여 어떤 행위를 하는 것인지 나타내는 것으로 써 READ, WRITE, EVENT로 분류되어져있다. READ action같은 경우, 리소스에 접근하여 리소스를 읽어 들이는 것이며 WRITE는 리소스에 접근하여 쓰는 행위, EVENT는 해당 리소스가 발생시키는 이벤트를 받아들이는 것을 의미한다. Security-Mode ONOS에서 Northbound API에 대해 현재까지 정의된 퍼미션타입들은 표 1과 같다. 각 permission type에 매칭되는 API들은 현재 Security-Mode ONOS wiki페이지[13] 에 모두 정리되어있다.

Permission Type	Permission Description
APP_READ	Permission to read various information about installed applications
APP_WRITE	Permission to register new application
APP_EVENT	Permission to receive application life cycle events
CONFIG_READ	Permission to read configuration properties
CONFIG_WRITE	Permission to write configuration properties
CODEC_READ	Permission to read codec information
CODEC_WRITE	Permission to add/remove entity class from codecs
CLOCK_WRITE	Permission to write clock properties
CLUSTER_READ	Permission to read cluster information
CLUSTER_EVENT	Permission receive cluster events
DEVICE_READ	Permission to read device information
DEVICE_EVENT	Permission receive device events
DRIVER_READ	Permission to get driver instances
DRIVER_WRITE	Permission to create a new driver handler
DEVICE_KEY_READ	Permission to read device key
EVENT_READ	Permission to read event properties
EVENT_WRITE	Permission to write event properties
FLOWRULE_READ	Permission to read flow rule information
FLOWRULE_WRITE	Permission to add/remove flow rules
FLOWRULE_EVENT	Permission receive flow rule events
GROUP_READ	Permission to read group information
GROUP_WRITE	Permission to modify groups
GROUP_EVENT	Permission to receive group events
HOST_READ	Permission to read host information
HOST_WRITE	Permission to modify host
HOST_EVENT	Permission receive host events
INTENT_READ	Permission to read intent information
INTENT_WRITE	Permission to add/remove intents
INTENT_EVENT	Permission receive intent events
LINK_READ	Permission to read link information
LINK_WRITE	Permission to modify link information
LINK_EVENT	Permission receive link events
MUTEX_WRITE	Permission to execute mutex task
PACKET_READ	Permission to read packet information
PACKET_WRITE	Permission to send/block packet
PACKET_EVENT	Permission to handle packet events
PARTITION_READ	Permission to read partition information
PARTITION_EVENT	Permission to handle partition events
PERSISTENCE_WRITE	Permission to create persistent builder
REGION_READ	Permission to read region information
RESOURCE_READ	Permission to read resource information
RESOURCE_WRITE	Permission to allocate/release resource
RESOURCE_EVENT	Permission to handle resource events
STATISTIC_READ	Permission to access flow statistic information
TOPOLOGY_READ	Permission to read path and topology information
TOPOLOGY_EVENT	Permission to handle topology events
TUNNEL_READ	Permission to read tunnel information
TUNNEL_WRITE	Permission to create tunnels
TUNNEL_EVENT	Permission to receive tunnel events
UI_READ	Permission to read UI information
UI_WRITE	Permission to create/remove UI service
STORAGE_WRITE	Permission to create stores

표 1 Security-Mode ONOS 지원 Northbound API permission Type

o Virtual network Level Access Control

Virtual network Level Access Control 이란, 데이터부에 구현되어 있는 가상 네트워크에 대한 액세스 컨트롤을 하는 것 이다. 이것을 우리는 SM ONOS-VN이라 부른다. 가상네트워크는 하나의 물리네트워크 상에서, 논리적으로 네트워크를 여러 개로 나누어, 각 관리자에 따라 논리적으로 구분된 네트워크를 부여 하는 것 이다. 본절에서 다룰 논리 네트워크는 그림 9와 같다.

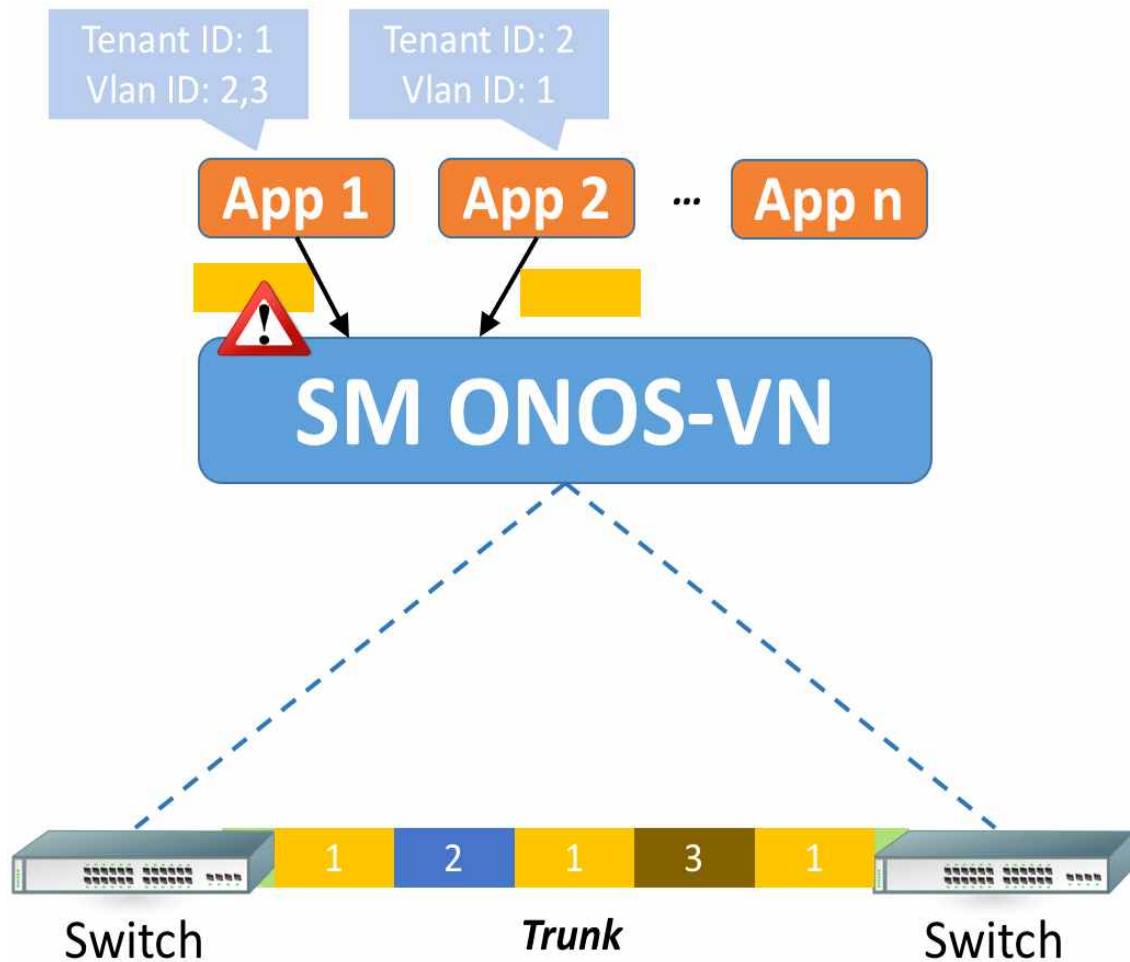


그림 9 SM ONOS-VN의 동작 시나리오

그림 9는 SM ONOS-VN의 동작 시나리오를 나타내고 있다. 각 스위치는 물리 망으로 연결이 되어 있고, 하나의 포트를 이용해 Trunk방식으로 가상네트워크를 만든 것을 확인 할 수 있다. 각각의 가상네트워크는 물리네트워크 망을 공유하고 논리적으로 분리된 가상네트워크 망을 이루게 된다. 각각의 애플리케이션은 서로 다른 가상네트워크에 접근을 시도하려고 한다. 이때 각각의 애플리케이션은 SM ONOS와 같이, 자신이 가지고 있는 애플리케이션의 아이디에 따라 각각의 가상네트워크에 접근 할 수 있는 권한을, 권한파일(Policy file)에 명시를 해야 한다. 해당 번호를 기반으로 SM ONOS-VN은 각 가상네트워크 별로 애플리케이션이 하는 행위를 제한

하게 된다. 시나리오에서 보면 애플리케이션 1번은 VN(Virtual Network) 1번 프레임에 대한 권한이 없는데, VN 1에 접근을 하는 것을 볼 수 있다. 이 경우 SM ONOS-VN은 이를 감지하고, 사전에 접근을 통제하게 된다. 이와 반대로 애플리케이션 2번은 해당 VN에 대한 적절한 권한이 있기 때문에 VN 1에 대해 SM ONOS-VN은 접근을 허가 해 준다.

이와 같이 SM ONOS-VN의 궁극적인 목적은 데이터부에서 동작하는 가상네트워크에 대하여 적절한 퍼미션을 부여하고, 해당 퍼미션을 기반으로 데이터부를 안전하게 보호하는 것이다. 포토타입에서는 호스트를 고려하지 않고, 네트워크 자체에 대한 (VN 자체에 대한) 최소권한 부여 정책을 목표로 한다.

SM ONOS-VN은 SM ONOS의 확장모듈로서 SM ONOS에서 사용되는 권한을 그대로 승계하게 된다. 표 2는 SM ONOS에 가상네트워크를 위한 권한이 추가된 전체 표를 볼 수 있다.

Permission Type	Permission Description
APP_READ	Permission to read various information about installed applications
APP_WRITE	Permission to register new application
APP_EVENT	Permission to receive application life cycle events
CONFIG_READ	Permission to read configuration properties
CONFIG_WRITE	Permission to write configuration properties
CODEC_READ	Permission to read codec information
CODEC_WRITE	Permission to add/remove entity class from codecs
CLOCK_WRITE	Permission to write clock properties
CLUSTER_READ	Permission to read cluster information
CLUSTER_EVENT	Permission receive cluster events
DEVICE_READ	Permission to read device information
DEVICE_EVENT	Permission receive device events
DRIVER_READ	Permission to get driver instances
DRIVER_WRITE	Permission to create a new driver handler
DEVICE_KEY_READ	Permission to read device key
EVENT_READ	Permission to read event properties
EVENT_WRITE	Permission to write event properties
FLOWRULE_READ	Permission to read flow rule information
FLOWRULE_WRITE	Permission to add/remove flow rules
FLOWRULE_EVENT	Permission receive flow rule events
GROUP_READ	Permission to read group information
GROUP_WRITE	Permission to modify groups
GROUP_EVENT	Permission to receive group events
HOST_READ	Permission to read host information
HOST_WRITE	Permission to modify host
HOST_EVENT	Permission receive host events
INTENT_READ	Permission to read intent information
INTENT_WRITE	Permission to add/remove intents
INTENT_EVENT	Permission receive intent events
LINK_READ	Permission to read link information
LINK_WRITE	Permission to modify link information
LINK_EVENT	Permission receive link events
MUTEX_WRITE	Permission to execute mutex task

<b>PACKET_READ</b>	Permission to read packet information
<b>PACKET_WRITE</b>	Permission to send/block packet
<b>PACKET_EVENT</b>	Permission to handle packet events
<b>PARTITION_READ</b>	Permission to read partition information
<b>PARTITION_EVENT</b>	Permission to handle partition events
<b>PERSISTENCE_WRITE</b>	Permission to create persistent builder
<b>REGION_READ</b>	Permission to read region information
<b>RESOURCE_READ</b>	Permission to read resource information
<b>RESOURCE_WRITE</b>	Permission to allocate/release resource
<b>RESOURCE_EVENT</b>	Permission to handle resource events
<b>STATISTIC_READ</b>	Permission to access flow statistic information
<b>TOPOLOGY_READ</b>	Permission to read path and topology information
<b>TOPOLOGY_EVENT</b>	Permission to handle topology events
<b>TUNNEL_READ</b>	Permission to read tunnel information
<b>TUNNEL_WRITE</b>	Permission to create tunnels
<b>TUNNEL_EVENT</b>	Permission to receive tunnel events
<b>UI_READ</b>	Permission to read UI information
<b>UI_WRITE</b>	Permission to create/remove UI service
<b>VN_CREATE</b>	<i>Permission to create a virtual network</i>
<b>VN_REMOVE</b>	<i>Permission to remove a virtual network</i>
<b>VN_READ</b>	<i>Permission to read a list of virtual networks</i>
<b>VN_EVENT</b>	<i>Permission to receive events from a certain virtual network</i>
<b>VN_WRITE</b>	<i>Permission to access a certain virtual network</i>

표 2 SM ONOS-VN의 권한이 추가된 전체 권한 목록.

표 2는 SM ONOS-VN의 권한이 추가된 전체 권한 목록으로서, SM ONOS-VN은 총 5개의 권한을 사용하게 된다. 이탤릭체로 표시된 권한은 관리자를 위한 권한으로서, 가상네트워크에 대한 수정 및 삭제를 행하는 권한이다. 해당 권한은 SM ONOS의 관리자(Admin permission)가 가지는 권한으로서, 가상네트워크를 직접 제어하는 권한을 의미한다. 나머지 세가지의 권한은 사용자(User permission)이 가지는 권한으로서, 특정 가상네트워크에 대한 접근권한이다. VN\_READ는 가상네트워크에 대한 정보를 열람할 수 있는 지에 대한 유무를 표현 한 것이다. 해당 권한은 데이터부에 직접적인 영향은 줄 수 없지만, 가상네트워크를 구성하는 다양한 정보를 열람 할 수 있기 때문에 해당 권한의 유무로, 가상네트워크 정보의 유출을 방지할 수 있다. VN\_EVENT는 특정 가상네트워크에서 발생하는 다양한 네트워크 정보들 (Packet\_IN, Flow statistics, ...)에 대한 제어메시지를 받을 수 있는지에 대한 여부이다. 예를 들어 특정 가상네트워크에 대한 권한이 있는 경우, SM ONOS-VN은 발생하는 이벤트가 어느 가상네트워크 상에서 발생되어 있는지를 확인 한 후, 적절한 권한이 있는 애플리케이션에게 해당 정보를 넘겨주게 된다. VN\_WRITE권한은 데이터부에 직접적으로 영향을 줄 수 있는 권한으로서, 특정 가상네트워크에 대하여 SM ONOS에서 정의한 데이터부에 영향을 줄 수 있는 권한을 사용 할 수 있는 권한이다. 예를 들어 특정네트워크에 대하여 해당 권한을 가진 경우에는, 애플리케이션이 대상 가상네트워크에 네트워크 룰도 내릴 수 있고, 물리네트워크를 관리할 때처럼 다양한 행위를 할 수 있다.

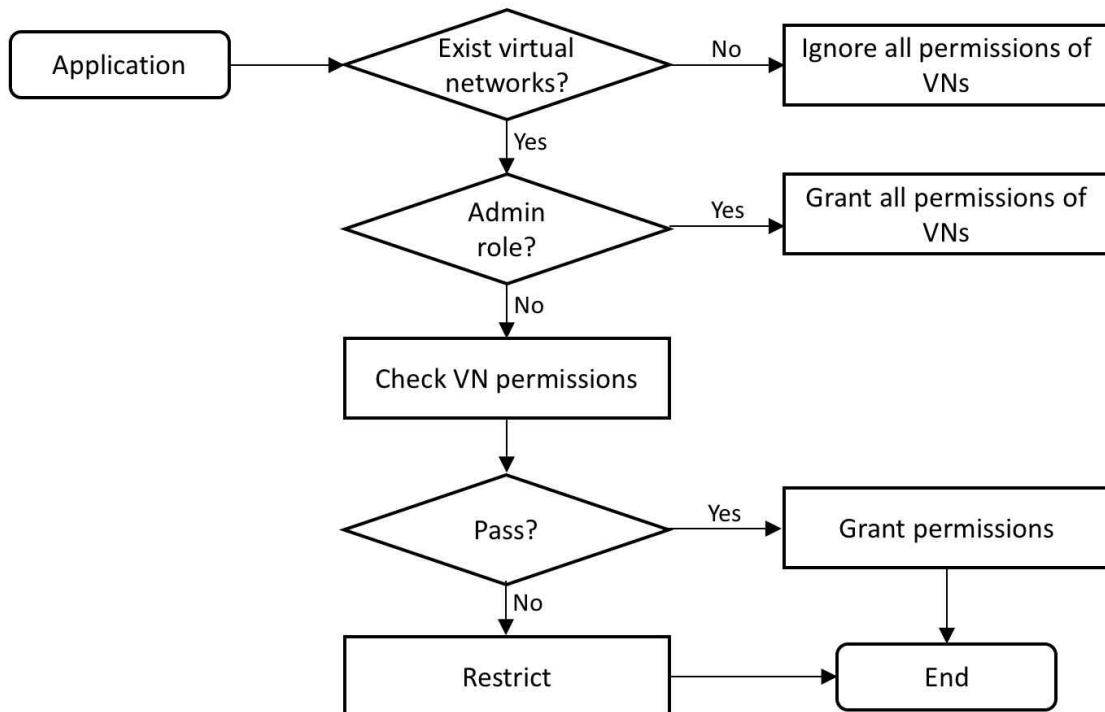


그림 10 SM ONOS-VN 퍼미션 모델 플로우 차트

그림 10은 SM ONOS-VN의 퍼미션 모델 플로우 차트를 도식화 한 그림이다. 특정 이벤트 (가상네트워크 접근, 제어, 등등)가 발생 한 경우 먼저 SM ONOS-VN은 현재 가상 네트워크가 있는지 없는지를 먼저 판단 한 후, 없는 경우에는 가상네트워크에 대한 모든 권한을 무시하게 된다. 그 다음 관리자역할인지 확인을 한 후 만약 관리자 권한을 가진다면 모든 권한을 부여하게 된다. 관리자 권한이 없는 경우에는 해당 애플리케이션은 최소 권한 전략 (Fine-grained access control mechanism)을 취하고 있는 것이므로, 권한을 상세히 확인하는 프로세스를 진행하게 된다. SM ONOS-VN의 권한파일에 정의 되어 있는 다양한 권한을 확인 한 뒤, 만약 해당 권한을 애플리케이션이 가지고 있으면, 권한을 주고 아니면 권한을 부여하지 않는다.

기본적인 SM ONOS-VN의 퍼미션 플로우 차트는 기존 SM ONOS 퍼미션모델과 비슷하다. 차이점은 대상이 제어부에 대한 최소 권한 전략에서 데이터부를 위한 최소권한 전략으로 바뀌게 가장 큰 차이점이다.

그림 11은 SM ONOS-VN의 권한이 추가된 권한 형식을 나타내고 있다. 기본적으로 SM ONOS-VN의 권한파일은 SM ONOS에서 제공하는 권한파일을 그대로 계승하고, 가상네트워크에 대한 권한이 추가된 점을 알 수 있다. SM ONOS에서 사용되는 role도 동일하게 적용이 된다. User 역할 일 경우, 가상네트워크에 대한 관리 권한 및 사용 권한은 최소 권한 부여 전략 (Fine-grained permission grant)을 취할 수 있고, 가상네트워크 관리 자체에 대한 권한과, 가상 네트워크 별 권한을 부여 할 수 있다.



```

<security>
  <role>USER</role>

  <VNs>
    <VN>
      <VNID>5</VNID>
      <VN-perm>VN_READ</VN-perm>
    </VN>
    <VN>
      <VNID>2</VNID>
      <VN-perm>VN_READ</VN-perm>
      <VN-perm>VN_EVENT</VN-perm>
      <VN-perm>VN_WRITE</VN-perm>
    </VN>
  </VNs>

  <permissions>
    <app-perm>VN_CREATE</app-perm>
    <app-perm>VN_REMOVE</app-perm>
    <app-perm>DEVICE_READ</app-perm>
    <app-perm>PACKET_WRITE</app-perm>
    <app-perm>HOST_READ</app-perm>
    <java-perm>
      <classname>org.osgi.framework.AdminPermission</classname>
      <name>*</name>
      <actions>metadata</actions>
    </java-perm>
    <java-perm>
      <classname>org.lang.RuntimePermission</classname>
      <name>modifyThread</name>
    </java-perm>
  </permissions>
</security>

```

Permissions per VN

SM ONOS permissions

그림 11 SM ONOS-VN이 추가된 권한 파일 형식

만약 Admin인 경우에는 전체 가상네트워크에 대한 모든 권한을 가질 수 있다. 예를 들어 Admin권한을 가진 애플리케이션은 모든 가상네트워크에 대한 VN\_READ, VN\_EVENT, VN\_WRITE권한을 가지게 되고, 관리자로서 모든 가상네트워크에 대한 삭제 및 생성을 할 수 있다.

VN태그는 새롭게 추가된 권한으로서, 특정 가상네트워크에 대한 최소 권한을 부여 할 수 있는 태그이다. 각 가상 네트워크에 따라 서로다른 권한을 부여 할 수 있다. 특히 VN\_WRITE권한은 SM ONOS에서 정의 한 데이터부에 영향을 줄 수 있는 권한들을 app-perm태그에 정의되어 있는 권한에 한해 수행 할 수 있도록 한다. 마찬가지로 VN\_READ는 가상네트워크에 대한 전반적인 정보와 해당 가상네트워크에 대한 정보를 열람 할 수 있다. 이 경우에는 데이터부에 직접적인 영향을 주지 않으므로 VN\_WRITE보다 덜 중요한 권한이 된다. VN\_EVENT같은 경우에는, 해당 가상네트워크에서 오는 이벤트정보를 받을 수 있는 권한이다. app-perm태그에는 가상 네트워크 관리에 대한 권한을 부여 할 수 있다. 가상네트워크 자체에 대한 권한은

가상네트워크를 생성하거나 삭제할 수 있는 권한으로서 Admin권한에 속하지만, 최소권한 정책으로 좀더 상세히 정의 할 수 있다.

이와 같이 SM ONOS-VN은 SM ONOS 권한 파일을 그대로 승계하여, 가상네트워크에 대한 부분을 추가하였고, 하위호환성을 고려하여 디자인되었다. 하지만 아직 ONOS의 virtual network가 구현완료 되지 않았기에 개발은 ONOS incubator에 구현되어 있는 부분만 고려하여 구현하였다. 현재 incubator에 있는 ONOS의 virtual network가 구현완료 되면 바로 SM ONOS를 사용할 수 있도록 구현하였으며, ONOS의 virtual network가 최종적으로 구현 완료되면 이를 바탕으로 추가적으로 개발을 진행할 예정이다.

#### o Host-level Access Control

Host-level Access Control 이란, Infrastructure Layer에서의 엔드 장치의 서비스를 대상으로 액세스 컨트롤을 하는 것이다. 앞선 여러 종류의 액세스 컨트롤과 다르게 실제 호스트가 위치한 계층에서의 액세스 컨트롤을 구현하는 것이기 때문에, 기존의 코어 서비스를 수정하는 것이 아닌 새로운 SM-ONOS 애플리케이션으로 호스트 레벨의 액세스 컨트롤을 제공한다. 이러한 호스트 단위의 액세스 컨트롤은 데이터 플레인에 연결된 각각의 호스트를 위한 서비스들 (Web 서비스, FTP 서비스 등.)을 대상을 액세스 컨트롤을 제공한다. 본 기술문서에서는 SM-ONOS 애플리케이션을 SM-ONOS HAC 애플리케이션으로 정의하였다.

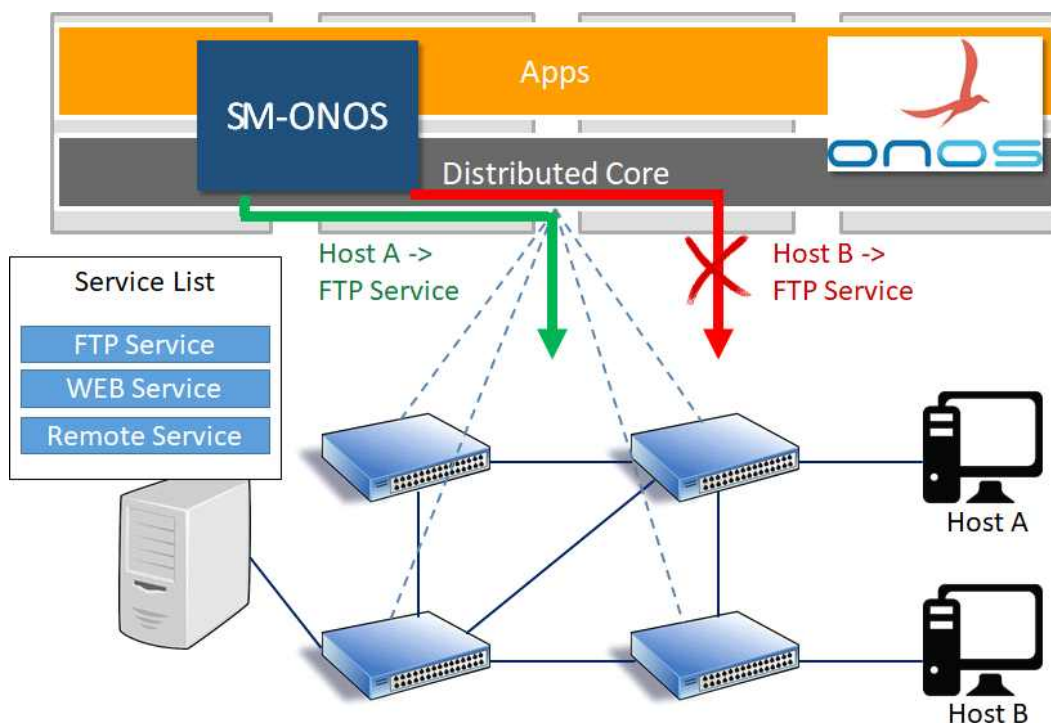


그림 12 SM-ONOS HAC 애플리케이션 액세스 컨트롤 예시

그림 12는 SM-ONOS HAC의 액세스 컨트롤 서비스 예시를 보여준다. 그림에서의 호스트 A와 호스트 B는 각각 다른 호스트의 서버가 제공하는 FTP 서비스를 이용하기를 원한다고 가정한다. 이때의 호스트 A는 네트워크 관리자의 허가를 받아 FTP 서비스를 이용할 수 있는 권한이 있는 호스트이며, 호스트 B는 네트워크 관리자의 허가를 받지 않아 FTP 서비스를 이용할 권한이 없는 호스트이다. 따라서 호스트 A가 특정 서버가 제공하는 FTP 서비스를 이용해 특정 파일을 다운로드 받고자 하는 경우, SM-ONOS HAC 애플리케이션은 권한이 있는 호스트임을 판단하고, 아무런 제제를 하지 않는다. 하지만 호스트 B가 특정 서버가 제공하는 FTP 서비스를 이용해 원하는 파일을 다운로드 받고자 하는 경우, SM-ONOS HAC 애플리케이션은 관련된 트래픽을 모두 drop하는 플로우 규칙을 생성해 스위치에 설치함으로써 이를 차단하게 된다.

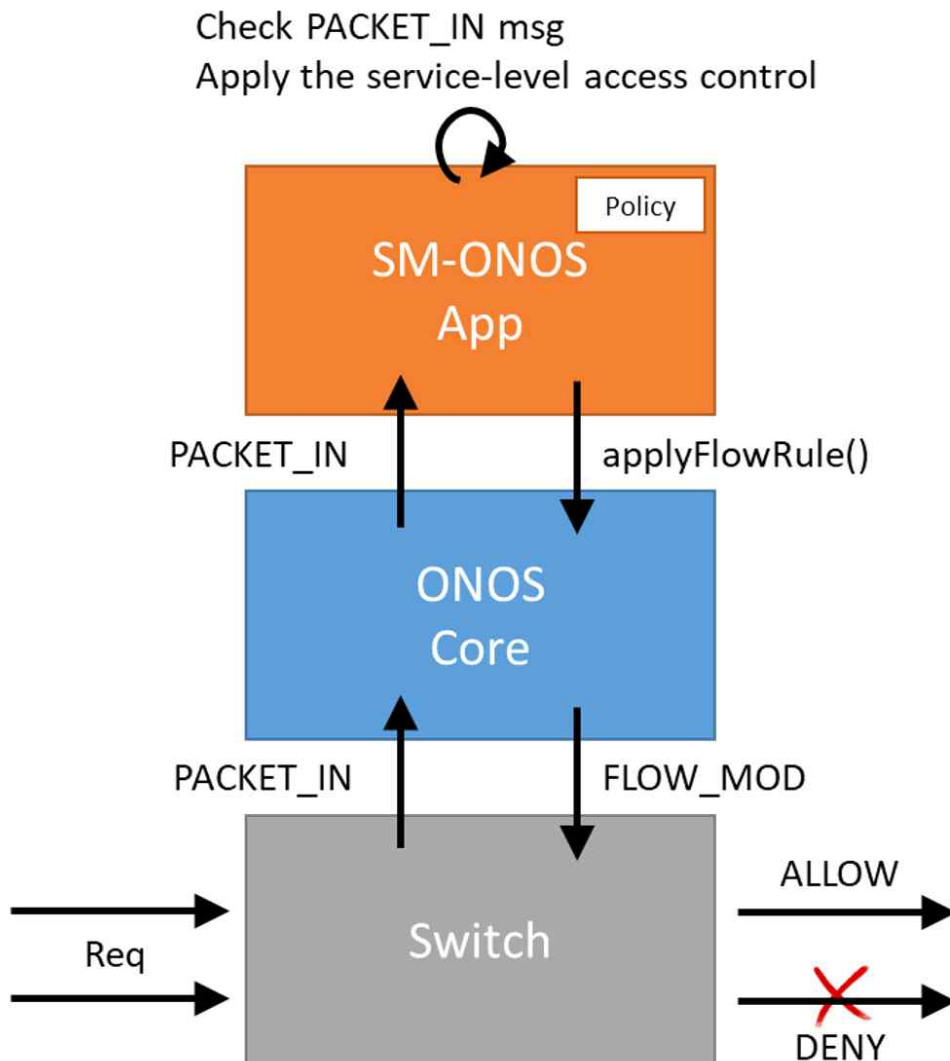


그림 13 Host-level Access Control 기본 메커니즘

SM-ONOS의 HAC 애플리케이션의 기본 메커니즘은 그림 13과 같다. 이 메커니즘에서 액세스 컨트롤을 위한 기본적인 정책들은 네트워크 관리자에게 받았다고 가정한다. 가장 먼저, 새로운 트래픽이 컨트롤러가 관리하는 스위치에 들어온 경우, 스위치는 PACKET\_IN 메시지에 맨 처음 들어온 패킷을 포함해서 컨트롤러에게 전달한다. 컨트롤러에게 PACKET\_IN 메시지가 도달하면, 컨트롤러는 해당 메시지를 받을 수 있는 SM-ONOS HAC 애플리케이션에게 전달한다. PACKET\_IN 메시지를 수신한 SM-ONOS HAC 애플리케이션은 가장 먼저 메시지 안의 패킷을 확인하고, 현재 네트워크 관리자가 설정한 정책들과 비교한다. 비교 결과 접근 권한이 없는 플로우가 스위치에 도달했다고 판별하면 해당 플로우를 강력한 우선순위로 drop하는 플로우 룰을 새롭게 컨트롤러에게 전달한다. 그리고 DROP 룰을 받은 컨트롤러는 FLOW\_MOD 메시지를 이용해 스위치에게 전달하고, 최종적으로 스위치는 해당 플로우 룰을 자신의 플로우 테이블에 저장해, 관련된 모든 플로우를 차단하게 된다. 여기서 사용하는 정책들은 네트워크 관리자가 ONOS의 CLI를 이용해 정책을 내릴 수 있으며 내린 정책들은 ONOS의 CLI나 WEB 서비스를 이용해 확인할 수 있다. 네트워크 관리자가 CLI를 이용해 정책을 추가/삭제/확인하는 방법은 2.8장에서 관련된 예시와 함께 자세하게 확인할 수 있다.

### 2.3. Security-Mode ONOS 디자인

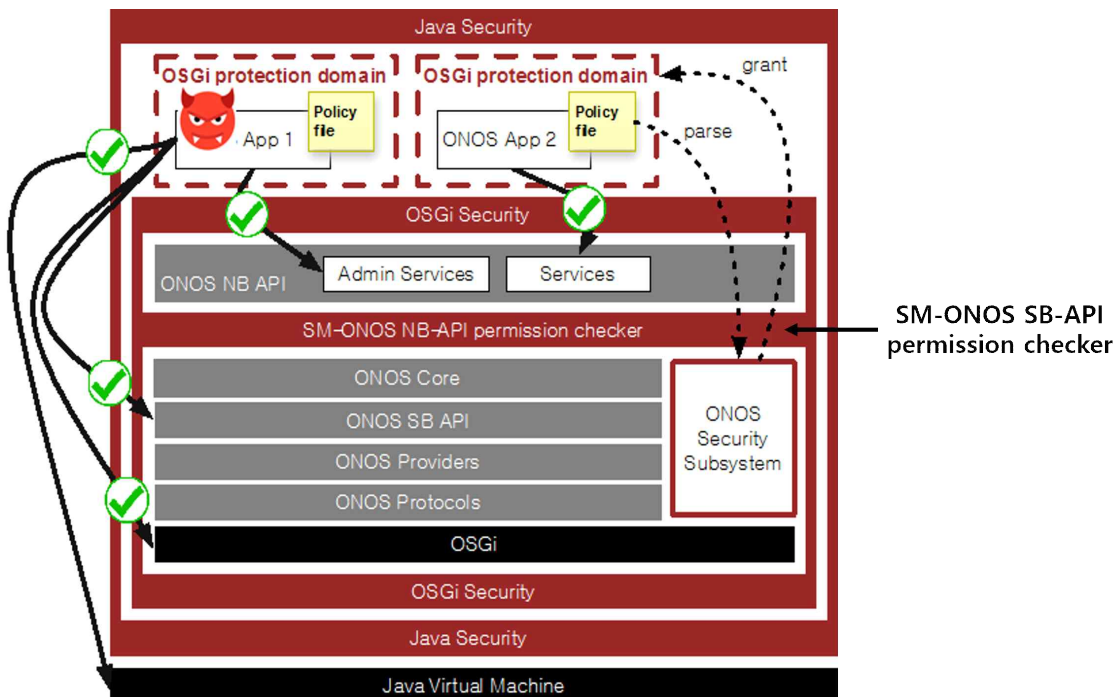


그림 14 Security-Mode ONOS 디자인

Security-Mode ONOS는 현재 그림 14와 같이 디자인 되어져 있다. 그림 14에서

볼 수 있듯이, Security-Mode ONOS의 ONOS security subsystem은 ONOS의 kernel과 애플리케이션을 논리적으로 분리시켜 각 ONOS에 설치된 애플리케이션들을 sandboxing 시킨다. 또한, ONOS 애플리케이션들이 ONOS core와 그 밑의 레이어들에 대한 접근을 감시하고 중재하기 위해, ONOS Security subsystem은 ONOS kernel 번들들과는 다른 독립된 번들로써 동작한다. 이 ONOS security subsystem은 애플리케이션이 로딩될 때, 애플리케이션 번들 내에 포함된 policy파일을 읽어와서 파싱하고, 해당 퍼미션들을 애플리케이션 번들에게 부여한다. 애플리케이션번들에게 퍼미션이 부여된 이후에는, 애플리케이션이 부여받은 퍼미션 범위를 넘어서는 API들을 호출하는지 실시간 체크를 하게 되는데 OSGI API, Northbound API, JAVA native API들에 대한 실시간 체크는 OSGI security layer, NB-API permission checker layer, JAVA security layer에서 수행하게 디자인 되어져 있다. 좀 더 자세히 설명하자면 그림 14에서, OSGI security layer는 OSGI에서 제공하는 행위들에 관련된, 예를 들어 서비스를 요청한다던지 metadata를 변경하는 행위에 대한 제어를 수행하며, SM-ONOS NB-API permission checker는 애플리케이션이 ONOS에서 제공하는 northbound api를 호출 할 때마다, 해당 애플리케이션이 사용하려는 api에 대한 퍼미션을 가지고 있는지 아닌지를 확인한다. JAVA security는 JAVA native에서 제공하는 행위들, 예를 들면 자바의 소켓퍼미션이나 파일퍼미션을 필요로 하는 행위들에 대한 접근제어를 수행한다.

이와 더불어 위에서 언급한 OSGI security layer를 확장하여 NB-API에 대한 permission check 뿐만 아니라 SM-ONOS SB-API permission checker를 NB-API permission checker와 동일한 layer에 추가하였다. SM-ONOS SB-API permission checker는 애플리케이션이 ONOS에서 제공하는 southbound api, 그 중 가상 네트워크에 대한 READ, WRITE, EVENT와 같은 기능을 제공하는 api를 호출할 때마다, 해당 애플리케이션이 사용하려는 api에 대한 퍼미션을 가지고 있는지 아닌지를 확인한다. 만약, 특정 애플리케이션이 권한이 없는 VNET의 정보를 확인 또는 수정하기 위해 READ나 WRITE를 한다면 Security-Mode ONOS는 그런 행위에 대한 접근제어를 통해 접근을 차단한다.

## 2.4. Security-Mode ONOS 구현

현재 Security-Mode ONOS의 구현에서 OSGI와 JAVA native API들에 대한 access control은 Felix OSGI security[4]를 확장, JavaSE 1.2 security를 이용하여 구현이 되어져 있다. ONOS Northbound API에 대한 access control은 2.2에서 기술한 것과 같이 새로운 ONOS Northbound API들에 따른 새로운 퍼미션 타입을 만들고 각 ONOS Northbound API들이 호출 될 때 마다, 해당 API가 실행되기 이전에 ONOS

Security Subsystem내에 있는 퍼미션체크 함수를 먼저 호출하여 API를 호출한 애플리케이션이 접근할 수 있는 퍼미션을 가지고 있는지 없는지를 검사한다. ONOS Southbound API에 대한 access control 또한 2.2에서 기술한 것과 같이 ONOS Southbound API, 그중 가상네트워크에 대한 퍼미션 타입을 만들고 이에 대한 API들이 호출 될 때 마다, 해당 API가 실행되기 이전에 ONOS Security Subsystem 내에 있는 SM-ONOS SB-API permission checker 내에 있는 퍼미션 체크 함수를 먼저 호출하여 API를 호출한 애플리케이션이 해당하는 가상 네트워크에 대해 접근할 수 있는 퍼미션을 가지고 있는지 없는지를 검사한다.

Security-Mode ONOS는 현재 계속 구현진행중이며, ONOS 컨트롤러의 Emu(1.4) version부터 컨트롤러 내에 포함되어 배포되어지고 있다.

## 2.5. Security-Mode ONOS 추가기능

### 2.5.1 Security-Mode ONOS의 보안강화기능

Security-Mode ONOS와 Android같은 퍼미션 모델과 같이, 애플리케이션 개발자가 애플리케이션이 필요한 퍼미션들을 선언하고 배포하여 end user가 그것을 다운받아 애플리케이션이 어떠한 권한이 필요한지 살펴보고 설치하는 경우, permission gap이라는 문제가 존재한다. Permission gap은 실제 애플리케이션이 사용하는 퍼미션보다 더 많은 퍼미션들이 xml파일에 선언되어있음에도 불구하고 end-user는 해당 퍼미션이 실제 애플리케이션이 사용하는 퍼미션인지 아닌지 모르기 때문에 더 많은 퍼미션들을 애플리케이션에게 부여하는 문제이다. 이러한 문제는 Android쪽에서 많은 연구가 진행되어져 왔다. Android쪽에서는 대부분 이러한 permission gap의 문제에 대해 실제 Android market에서 배포 되고있는 애플리케이션들을 다운로드받아 얼마나 많은 애플리케이션들이 더 많은 권한을 요청하고 있는지를 분석 연구하였다. 하지만, permission gap을 어떻게 해결해야 하는지에 대한 문제해결방법은 제시된 적이 없다. SM-ONOS는 permission gap의 문제를 해결하기위해, permission을 network operator가 review 할 때, application이 어떠한 권한을 실제로 사용하는지에 대해 정적분석 기법을 이용하여 분석하고, 선언된 permission과의 차이를 비교하여 어떠한 permission gap이 존재하며 이것이 차후 어떠한 문제를 발생시킬 수 있는지에 대해 알려줌으로써 보안을 강화시켰다. 그림 15는 permission gap을 이용한 보안강화기능의 결과화면이며, 이 부분에 대한 자세한 설명은 차후에 공개할 예정이다.



```

:-----:
: C - Confidentiality, I - Integrity, A - availability :
: cp - controlplane, dp - dataplane :
: (+) - More declared, (-) - Less declared :
:-----:

Permission Gap [Declared Permission - Actually Used Permission]:
  (+) [FLOWRULE_READ]
      Potential Risk : C-cp.dp
  (+) [TOPOLOGY_WRITE]
      Potential Risk : I-cp.dp, A-dp
  (+) [TOPOLOGY_EVENT]
      No Risk(Unused permission type on Security-Mode ONOS)
  (-) [APP_READ], [DEVICE_READ]
      Can potentially put your network in danger

Semantic Gap [Declared Permission - Description] :
  [APP_WRITE]
      Potential Risk : I-cp, A-cp.dp
  [FLOWRULE_READ]
  [TOPOLOGY_WRITE]
  [TOPOLOGY_EVENT]

```

그림 15 Permission gap을 활용한 보안강화기능 결과화면

## 2.5.2 Unit test

Security-Mode ONOS의 디자인에 대한 구현을 점검하기 위해 unit test를 수행할 수 있다. 즉, 구현된 Security-Mode ONOS 각 모듈의 동작을 테스트함으로써, 디자인에 따른 정확한 구현을 확인할 수 있다. unit test는 기본적으로 프로그램의 오류를 발견하여 시스템의 안정성을 제공한다. 예를 들어, 잘못된 자료형이나 논리 연산자의 사용, 수식 및 알고리즘 계산 오류, 무한 루프 등의 시스템 오류를 점검한다. 또한, ONOS의 일부분으로서의 Security-Mode ONOS의 안전한 실행도 함께 점검할 수 있다. unit test를 통해 Security-Mode ONOS를 ONOS에서 고립 시켜서 테스트를 수행함으로써, Security-Mode ONOS의 불확실성을 제거하고, 전체 ONOS 모듈들과의 통합성을 향상시킨다.

Security-Mode ONOS는 다른 ONOS 모듈들과 마찬가지로, JUnit 프레임워크 기반의 unit test를 제공한다. JUnit은 컴파일 과정에서 JAR 파일 형태로 연결되어 동작하며, 본 시스템의 동작에는 영향을 끼치지 않고, 별개의 모듈로서 구동된다. 표 3은 JUnit에서 자주 사용하는 주석(annotation)들의 예시들을 나타내며, 이러한 주석들을 사용하여 테스트 케이스를 구성할 수 있다.

Annotation	Description
<b>@Test</b> <b>public void method()</b>	해당 메소드를 테스트 메소드로 선언한다.
<b>@Test (expected = Exception.class)</b>	메소드가 정의된 예외 클래스를 발생시키지 않을 경우, 테스트 실패를 리턴한다.
<b>@Test (timeout=100)</b>	메소드의 실행이 정의된 시간 (100 milliseconds)을 초과할 경우, 테스트 실패를 리턴한다.
<b>@Before</b> <b>public void method()</b>	해당 메소드를 테스트 시작 전에 실행한다. 보통 인풋 데이터를 읽거나, 변수 초기화 등의 테스트 환경을 구성하는데 사용한다.
<b>@After</b> <b>public void method()</b>	해당 메소드를 테스트 종료 후에 실행한다. 보통 임시 데이터를 삭제하거나, 초기값 복구 등의 테스트 환경을 정리하는데 사용한다.
<b>@BeforeClass</b> <b>public static void method()</b>	모든 테스트를 시작하기 전에 해당 정적 메소드를 실행한다. 주로 데이터베이스 연결과 같은 작업을 수행한다.
<b>@AfterClass</b> <b>public static void method()</b>	모든 테스트가 종료된 후에 해당 정적 메소드를 실행한다. 주로 데이터베이스 연결 종료와 같은 환경 정리 작업을 수행한다.
<b>@Ignore</b>	테스트 메소드를 생략한다. 코드가 수정 중이거나, 테스트 케이스가 적용되지 않은 경우, 혹은 테스트 실행에 많은 시간이 소요될 때 유용하다.

표 3 JUnit 기본 주석(Annotation) 예시

o unit test 케이스 구현 예시

```
/**
 * Test of AppGuard.
 */
public class AppGuardTest {

    @Test(expected = AccessControlException.class)
    public void testCheckPermission() throws Exception {
        SecurityManager sm = new SecurityManager();
        sm.checkPermission(new AppPermission(AppPermission.Type.APP_EVENT));
    }
}
```

그림 16 Security-Mode ONOS unit test 케이스 예시

그림 16은 작성된 Security-Mode ONOS unit test 케이스의예시를보여준다. 해당 테스트 케이스는 Security-Mode ONOS의 AppGuard 클래스의 checkPermission 메소드에 대한 테스트를 수행한다. 테스트 실행 동안 AccessControlException 클래스



의 예외가 발생하지 않으면, 해당 테스트 케이스는 실패로 결과를 리턴한다. 이를 통해, checkPermission 메소드가 정상적으로 퍼미션을 검증할 수 있는지를 테스트할 수 있다.

## 2.6. 일반적인 Security-Mode ONOS 사용법

### o 개발자가 작성해야하는 policy 파일 형식

```

<security>
  <role>USER</role>
  <permissions>
    <app-perm>DEVICE_READ</app-perm>
    <app-perm>CONFIG_WRITE</app-perm>
    <app-perm>TOPOLOGY_READ</app-perm>
    <app-perm>PACKET_WRITE</app-perm>
    <app-perm>HOST_READ</app-perm>
    <app-perm>FLOWRULE_WRITE</app-perm>
    <java-perm>
      <classname>org.osgi.framework.AdminPermission</classname>
      <name>*</name>
      <actions>metadata</actions>
    </java-perm>
    <java-perm>
      <classname>java.lang.RuntimePermission</classname>
      <name>accessDeclaredMembers, modifyThread</name>
    </java-perm>
  </permissions>
</security>
  
```

The diagram illustrates the XML structure of a policy file. Annotations on the right side identify different permission types: 'ONOS Application role' points to the <role>USER element; 'ONOS Application permissions' points to the <app-perm> elements; 'OSGi permissions' points to the <java-perm> element with <classname>org.osgi.framework.AdminPermission; and 'Java native' points to the <java-perm> element with <classname>java.lang.RuntimePermission.

그림 17 Policy 파일 형식 예제

Security-Mode ONOS에서 애플리케이션이 실행될 수 있게 하기 위해서는, 개발자는 자신의 ONOS 애플리케이션을 구현한 후, 어떠한 API를 사용하는지 분석하여 그에 맞는 policy 파일을 app.xml로 만들어서 패키징하고 배포해야한다. 개발자가 작성해야 하는 policy 파일의 형식은 그림 17과 같이 Application의 role을 USER나 ADMIN으로 작성해야하며 ONOS Northbound API들에 대한 퍼미션리스트, OSGI 퍼미션, 그리고 자바 Native 퍼미션들을 차례대로 작성해주면 된다.

### o Security-Mode ONOS를 활성화시키는 방법

ONOS가 설치되어있는 로컬에서, shell에 아래와 같이 입력하면 자동으로 Security-Mode ONOS를 활성화시킨 환경이 구축된다.

```

$ onos-setup-karaf secure
$ onos-package -s -t
  
```

### o Security-Mode ONOS에서 애플리케이션을 활성화 시키는 방법

Security-Mode ONOS는 2.3절에서 기술하였듯이, 애플리케이션이 어떠한 퍼미션을 필요로 하는지 검토하고 애플리케이션에게 해당 퍼미션을 부여해야만 애플리케이션이 활성화 될 수 있게 된다. 만약 검토 과정을 거치지 않는다면 그림 18과 같이 경고가 뜨며 애플리케이션이 활성화되지 않는다.

```

Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout'

onos> app activate org.onosproject.attack

*****
SM-ONOS APP WARNING
*****
org.onosproject.attack has not been secured
Please review before activating.

This application has NOT been
reviewed and approved
by an ONOS operator

```

그림 18 퍼미션 부여되지 않은 애플리케이션을 활성화시키려는 경우

애플리케이션을 활성화시키기 위한 과정은, 먼저 그림 19와 같이 review [app-name] 명령어를 통해 애플리케이션이 요구하는 퍼미션들을 검토해야 한다.

```

onos> review org.onosproject.attack

*****
SM-ONOS APP REVIEW
*****
Application name: org.onosproject.attack
Application role: USER

Developer specified permissions:
[APP PERMISSION] HOST_EVENT
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] FLOWRULE_WRITE
[APP PERMISSION] INTENT_READ
[APP PERMISSION] INTENT_WRITE
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[Other SERVICE] org.onosproject.attack.Attack(get,register)
[SB SERVICE] org.onosproject.net.link.LinkProviderRegistry(get,register)
[CRITICAL PERMISSION] RuntimePermission exitVM.0 ()

Permissions granted:

Must review PERMISSIONS before activating an app

Network admin may decide either to
1) Accept and grant the permissions
2) Reject and uninstall the app

```

그림 19 Security-Mode ONOS review 명령어 사용법

퍼미션이 해당 애플리케이션이 가질만한 퍼미션 이라면, 그림 20과 같이 review [app-name] accept 명령어를 통해 해당 애플리케이션에게 퍼미션을 부여해준다. 이렇게 퍼미션을 부여해주고 난 이후에는, ONOS에서 애플리케이션을 활성화시키는

명령어인 `app activate [app-name]`을 통해 애플리케이션을 활성화 시킬 수 있다.

```
onos> review org.onosproject.attack accept

*****
SM-ONOS APP REVIEW
*****
Application name: org.onosproject.attack
Application role: USER

Developer specified permissions:
[APP PERMISSION] HOST_EVENT
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] FLOWRULE_WRITE
[APP PERMISSION] INTENT_READ
[APP PERMISSION] INTENT_WRITE
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[Other SERVICE] org.onosproject.attack.Attack(get,register)
[SB SERVICE] org.onosproject.net.link.LinkProviderRegistry(get,register)
[CRITICAL PERMISSION] RuntimePermission exitVM.0 ()

Permissions granted:
[APP PERMISSION] INTENT_WRITE
[APP PERMISSION] FLOWRULE_WRITE
[APP PERMISSION] HOST_EVENT
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] INTENT_READ
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[Other SERVICE] org.onosproject.attack.Attack(get,register)
[SB SERVICE] org.onosproject.net.link.LinkProviderRegistry(get,register)
[CRITICAL PERMISSION] RuntimePermission exitVM.0 ()
```

Network admin has agreed to grant the permissions to this application.

The security policy is enforced,  
The admin may activate the app!

그림 20 Security-Mode ONOS accept 명령어 사용법

## o unit test 실행 방법

ONOS가 설치된 환경에서, Shell 명령어를 통해 작성된 테스트 케이스들을 실행시킬 수 있다. Maven 빌드를 사용할 경우, 아래와 같이 maven 명령어로 테스트를 실행한다.

```
$ mvn clean test
```

최신 버전의 ONOS는 오픈 소스 빌드 도구인 Buck을 사용하는 빌드를 제공한다. Buck을 활용하여 ONOS의 unit test는 다음의 명령어를 통해 실행할 수 있다.

```
$ ONOS_ROOT/tools/build/onos-buck test
```

## 2.7. 가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS 사용법

이전 절과 다르게 가상 네트워크 접근제어 기능을 사용하기 위해서는 이전 절과 유사하지만 몇 가지 다른 명령어를 사용해야 한다. 따라서 이번 절에서는 이전 절과는 다르게 ONOS에서 기본적으로 제공하는 ReactiveForwarding 애플리케이션을 약간 수정해 이 애플리케이션을 대상으로 예를 들어 가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS의 사용 방법에 대해 살펴보겠다.

```
<role>USER</role>
<permissions>
  <app-perm>DRIVER_READ</app-perm>
  <app-perm>DEVICE_READ</app-perm>
  <app-perm>CONFIG_READ</app-perm>
  <app-perm>CLUSTER_READ</app-perm>
  <app-perm>DEVICE_READ</app-perm>
  <app-perm>TOPOLOGY_READ</app-perm>
  <app-perm>PACKET_READ</app-perm>
  <app-perm>HOST_READ</app-perm>
  <app-perm>FLOWRULE_READ</app-perm>
  <vnet-perm>
    <permission>VNET_READ</permission>
    <vnetId>1</vnetId>
  </vnet-perm>
  <vnet-perm>
    <permission>VNET_WRITE</permission>
    <vnetId>1</vnetId>
  </vnet-perm>
  <java-perm>
    <classname>org.osgi.framework.AdminPermission</classname>
    <name>*</name>
    <actions>*</actions>
  </java-perm>
  <java-perm>
    <classname>org.osgi.framework.ServicePermission</classname>
    <name>org.apache.karaf.shell.console.CompletableFunction</name>
    <actions>register</actions>
  </java-perm>
</permissions>
```

ONOS Virtual Network Permission

그림 21 Policy 파일 형식 예제

o 개발자가 작성해야하는 policy 파일 형식

가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS에서 애플리케이션이 실행될 수 있게 하기 위해서는, 개발자는 자신의 ONOS 애플리케이션을 구현한 후, 어떠한 API를 사용하는지 분석하여 그에 맞는 policy를 기존과 다른 BUCK 파일에 정의해야 한다. 개발자가 작성해야 하는 policy의 형식은 그림 21과 같이 Application의 role을 USER나 ADMIN으로 작성해야하며 ONOS Northbound API들에 대한 퍼미션리스트, ONOS Southbound API인 가상 네트워크에 대한 퍼미션, OSGI 퍼미션, 그리고 자바 Native 퍼미션들을 차례대로 작성해주면 된다.

o 가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS에서 애플리케이션을 활성화 시키는 방법

```
onos> app activate org.onosproject.fwd

*****
SM-ONOS APP WARNING
*****
org.onosproject.fwd has not been secured.
Please review before activating.
Activated org.onosproject.fwd
```

그림 22 퍼미션을 부여하지 않은 상태에서의 활성화 경고 메시지

가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS의 경우, 일반적인 경우와 동일하게, 애플리케이션이 어떠한 퍼미션을 필요로 하는지 검토하고 애플리케이션에게 해당 퍼미션을 부여해야만 애플리케이션이 활성화 될 수 있게 된다. 그림 22는 퍼미션을 부여하지 않은 상태에서 ONOS에서 기본적으로 제공하는 fwd 애플리케이션을 활성화 하였을 때 나타나는 경고 화면이다.

애플리케이션을 활성화시키기 위한 과정은, review [app-name] 명령어를 통해 애플리케이션이 요구하는 퍼미션들을 검토해야 한다.

퍼미션이 해당 애플리케이션이 가질만한 퍼미션 이라면, review [app-name] accept 명령어를 통해 해당 애플리케이션에게 퍼미션을 부여해준다. 이렇게 퍼미션을 부여해주고 난 이후에는, ONOS에서 애플리케이션을 활성화시키는 명령어인 app activate [app-name]을 통해 애플리케이션을 활성화 시킬 수 있다. 그림 23은 ONOS에서 기본적으로 제공하는 fwd 애플리케이션에 대한 퍼미션을 검토한 뒤 퍼미션을 부여하여 최종적으로 허가된 퍼미션의 목록이다. 이전에 언급한 일반적인 경우와는 다르게 가상 네트워크에 대한 퍼미션이 추가된 것을 확인할 수 있다.



```
Permissions granted:
[APP PERMISSION] DRIVER_READ
[APP PERMISSION] STORAGE_WRITE
[APP PERMISSION] FLOWRULE_WRITE
[APP PERMISSION] CLUSTER_WRITE
[APP PERMISSION] PACKET_EVENT
[APP PERMISSION] PACKET_READ
[APP PERMISSION] APP_WRITE
[APP PERMISSION] TOPOLOGY_READ
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] CONFIG_WRITE
[APP PERMISSION] FLOWRULE_READ
[APP PERMISSION] CONFIG_READ
[APP PERMISSION] HOST_READ
[APP PERMISSION] VNET_WRITE_ALL
[APP PERMISSION] PACKET_WRITE
[APP PERMISSION] CLUSTER_READ
[APP PERMISSION] VNET_READ_1
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[Other SERVICE] org.onosproject.incubator.net.virtual.VirtualNetworkService(get)
[Other SERVICE] org.onosproject.fwd.ReactiveForwarding(register)
[Other] AdaptPermission (adaptClass=org.osgi.framework.wiring.BundleRevision) (adapt)
[Other] AdminPermission (name=org.onosproject.onos-core-net) (metadata)
[Other] AdminPermission * (class,execute,extensionLifecycle,lifecycle,listener,metadata)
[Other] ConfigurationPermission * (configure)
[Other] PropertyPermission java.version (read)
[CRITICAL PERMISSION] RuntimePermission accessDeclaredMembers ()
[CRITICAL PERMISSION] ReflectPermission suppressAccessChecks ()

Additional permissions requested on runtime (POLICY VIOLATIONS):
```

### ONOS Virtual Network Permission

그림 23 최종적으로 허가된 fwd 애플리케이션의 퍼미션 목록

#### o Security-Mode ONOS의 가상 네트워크 접근제어 기능 활성화 방법

```
onos>
onos> grantvnetperm org.onosproject.fwd READ 2
Error executing command: No enum constant org.onosproject.security.AppPermission.Type.VNET_READ_2
```

그림 24 가상 네트워크 접근제어 기능 실행 방법

가상 네트워크 접근제어 기능을 추가한 Security-Mode ONOS에서 특정 VLAN(가상네트워크)에 대한 접근제어 기능을 실행하기 위해서는 `grantvnetperm [app-name] [READ/WRITE/EVENT] [vnet-no]` 명령어를 통해 실행할 수 있다. 하지만 이 때, ONOS에서 Tenant와 가상 네트워크를 생성하여 생성한 VNET 번호에 한해서 [vnet-no]에 입력하여야 하며 그렇지 않은 경우 오류메시지를 출력한다. 또한 해당하는 VNET 번호와 그에 대한 입력한 Action의 퍼미션이 없는 경우, 그림 24와 같이 오류 메시지를 출력하는 것을 확인할 수 있다.

#### o Security-Mode ONOS의 가상 네트워크 접근제어 예시 및 테스트

구현한 가상 네트워크에서의 접근 제어 메커니즘이 정상적으로 접근 제어를 수행하는지 확인하기 위해, 기존의 ONOS의 주요 애플리케이션 중 하나인 Reactive Forwarding 애플리케이션에 특정 가상 네트워크의 정보를 불러오도록 수정하여 테스트를 진행한다.

첫 번째 테스트는 해당 애플리케이션의 가상 네트워크 서비스와 관련된 권한을 전혀 주지 않고 임의의 가상 네트워크의 정보를 불러오도록 수정하여 테스트를 진행하였다. 그림 25는 이에 대한 테스트 결과로 access denied 메시지와 함께 접근이 불가하다는 것을 확인할 수 있다.

```
2018-02-11 17:53:16,671 | WARN | -127.0.0.1:40532 | PacketManager
| 131 - org.onosproject.onos-core-net - 1.13.0.SNAPSHOT | Packet processor o
rg.onosproject.fwd.ReactiveForwarding$ReactivePacketProcessor@100ba06a threw an e
xception
com.google.common.util.concurrent.UncheckedExecutionException: java.security.Acce
ssControlException: access denied ("org.onosproject.security.AppPermission" "VNET
_READ_1")
```

그림 25 Security-Mode ONOS 가상네트워크 접근 제어 테스트 결과 1

두 번째 테스트는 해당 애플리케이션의 권한이 없는 가상 네트워크의 아이디를 통해 해당하는 가상 네트워크의 정보를 불러오도록 수정하여 테스트를 진행하였다. 그림 26은 이에 대한 테스트 결과로 예외처리(IllegalArgumentException)와 함께 해당 가상 네트워크 아이디의 정보에 대한 접근이 불가하다는 것을 확인할 수 있다.

```
2018-02-11 18:12:36,656 | WARN | -127.0.0.1:40814 | PacketManager
| 131 - org.onosproject.onos-core-net - 1.13.0.SNAPSHOT | Packet processor o
rg.onosproject.fwd.ReactiveForwarding$ReactivePacketProcessor@3939c4ab threw an e
xception
java.lang.IllegalArgumentException: No enum constant org.onosproject.security.App
Permission.Type.VNET_READ_2
```

그림 26 Security-Mode ONOS 가상네트워크 접근 제어 테스트 결과 2

현재 ONOS에서 가상 네트워크 서비스에 대한 구현이 완료되지 않았기 때문에 현재까지 구현된 부분만을 대상으로 Security-Mode ONOS를 구현하였다. 현재 ONOS에서 지속적으로 개발이 진행 중이므로 추후에 ONOS에서 완벽한 가상 네트워크 서비스를 제공하면 개발자는 Security-Mode ONOS를 통해 쉽게 네트워크 슬라이싱을 통한 네트워크 레벨의 접근 제어가 가능할 것으로 기대된다.

## 2.8. 다중 레이어에서의 접근 제어를 통합한 Multi-layered Security-Mode ONOS

앞서 소개한 여러 가지 접근 제어들인, Bundle-level rule based 접근제어, App-level rule based 접근 제어, App-level permission-based 접근제어, Virtual Network-level 접근제어, Host-level 접근제어를 모두 한 코어에서 제어 가능하도록 하여, Control Layer, Application Layer, Infrastructure layer 등, 다중 레이어에 대하여 접근제어를 가능케 하는 Multi-layered Security-Mode ONOS를 구현하였다.

따라서 그림 27에서는 이러한 접근 제어들이 하나의 코어 서비스에서 제공된다는

것을 표현하고 있다. 그림 27에서 A는 Bundle-level rule 기반 접근 제어, B 및 C는 App-level 접근제어, D는 Virtual Network-level 접근 제어, E는 Host-level 접근 제어가 가능하다는 것을 보여준다.

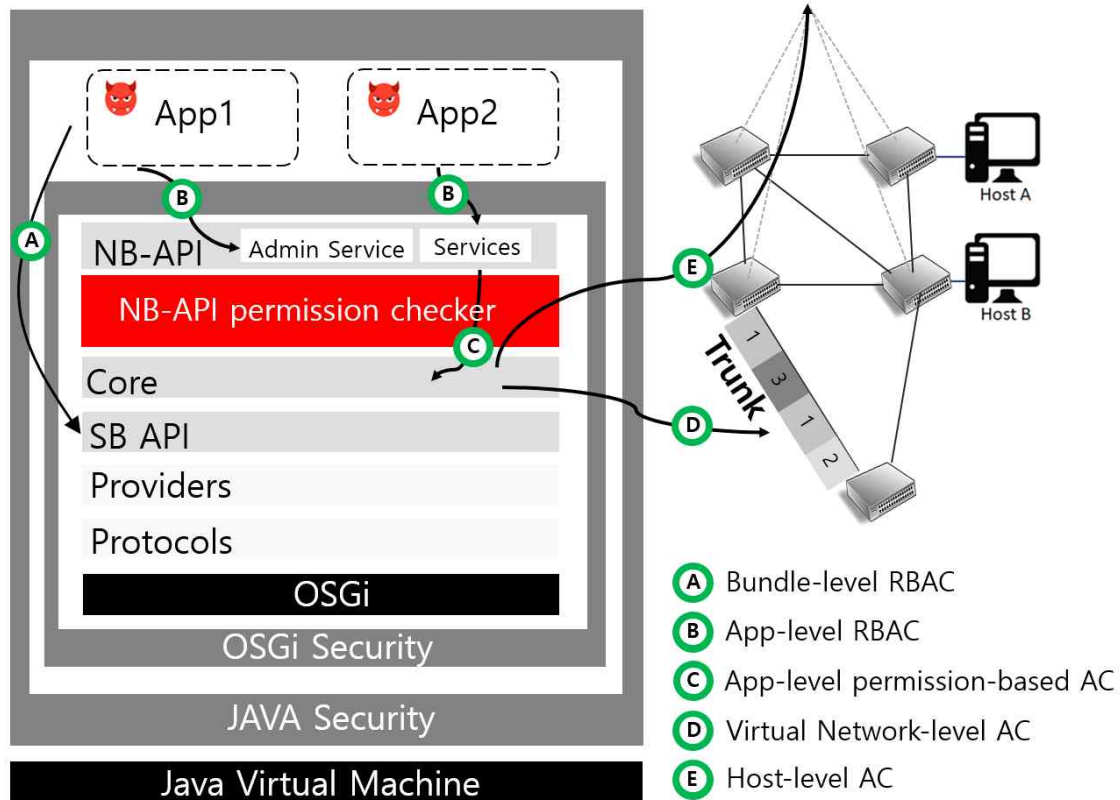


그림 27 Multi-layered Security Mode ONOS 접근제어 개요

따라서 MLS-ONOS의 다양한 요소들에 대한 통합 프레임워크 및 각 계층의 레이어에서 구현한 기능들에 대응하는 Access Control Language를 통합 프레임워크에 맞게 구현되었다. 통합 레이어 구현은 Control Plane specific AC(3 layer), Data Plane specific AC(2 layer)로 나누어진 MLS-ONOS의 다중레이어를 통합하여, 통합 AC Language를 통하여 네트워크 운영체제에 대한 방어를 전담하는 Control Plane 접근 제어와 호스트에 대한 영향을 줄 수 있는 Data Plane 접근 제어를 모두 제어할 수 있는 Role 기반의 접근 제어를 사용할 수 있고, 통합 Access Control Language의 구현은 본 연구에서 제안한 Access Control Language를 통하여 쉽게 각각의 응용에 대한 전 방위적 접근 제어를 구현할 수 있었다. 그 결과, MLS-ONOS를 구성하는 모든 컴포넌트와 기능을 이용할 수 있는 Access Control Language를 제공할 수 있다.

## 2.9. Multi-layered Security-Mode ONOS를 통한 실제 애플리케이션 공격 방어

이렇게 구현된 Multi-layered Security-Mode ONOS가 정상적으로 multi-layer에



대한 접근 제어를 수행하는지 확인하기 위해, DDos attack을 방어하는 애플리케이션을 실제로 만들고, 이 애플리케이션이 본래의 표명한 목적에서 벗어나는 공격을 하도록 하였다. 따라서 애플리케이션의 목적과는 전혀 상관없는 함수를 통해 SDN 네트워크를 공격하는 시나리오를 통하여 APP-level 접근제어와 Virtual Network-level 접근제어가 성공적으로 이루어지는 것을 보이려고 한다. 뿐만 아니라, Host-level 접근 제어도 성공적이라는 것을 확인하여, application layer, control layer, infrastructre layer 모두에서 다중 레이어에 대한 접근제어가 하나의 코어 서비스에서 성공적으로 제어 가능하다는 것을 이 절에서 확인하고자 한다.

#### o DDos 방어 애플리케이션의 세부사항

이 절에서 사용하는 애플리케이션은 DDos를 방어하는 기능을 가지는 직접 제작된 애플리케이션이다. 이 애플리케이션은 활성화 되면서 packet event가 발생할 때 packet의 정보를 받는 Listener를 background에 실행시키게 된다. 그 후 이 Listener를 통하여 packet event가 발생할 때, 이 packet에 대한 정보를 받아 packet이 DDos 공격을 위하여 사용 되어지는 것인지 판단한다. 만약 DDos 공격에 이용되고 있다면 방어기제를 실행한다. 그림 28의 코드와 같이 packet이 보내진 device의 port를 disable하는 방식으로 DDos 공격을 방어하게 된다.

```
private void DisableAttackPort(DeviceId deviceid, PortNumber portNumber){
    changePortState(deviceId, portNumber, disable)
}
```

그림 28 디바이스의 포트를 disable하는 코드

어떤 애플리케이션이 Security-Mode ONOS에서 실행되기 위해서는, 해당 애플리케이션의 policy-파일에 이 애플리케이션이 어떠한 퍼미션들을 요구하는지 명시해야 한다. 따라서 이 절에서의 DDos 방어 애플리케이션에도 policy 파일에 퍼미션들을 설정해주었다. 해당 애플리케이션이 위에서 설명한 것과 같은 기능을 실행하기 위해서는 기본적으로 PACKET, DEVICE와 관련된 퍼미션들을 필요로 한다. 실제로 부여한 퍼미션들은 아래 그림 29에서 확인할 수 있다. 그림 29는 SM-ONOS의 review 명령어를 통하여 해당 애플리케이션에 부여된 퍼미션들을 보여주고 있다. 이 퍼미션을 확인한 유저는 애플리케이션의 목적에 부합하는 퍼미션 set이라는 것을 확인한다면 SM-ONOS의 review [app-name] accept 명령어를 통하여 퍼미션들을 accept 해주게 된다. 그림 29에 나와 있듯이, PACKET을 읽는 것과 관련된 api들을 사용하는 것을 허락해주는 PACKET\_READ 퍼미션, PACKET EVENT에 관한 퍼미션인 PACKET\_EVENT 및 PACKET\_WRITE 퍼미션을 부여해줌으로써 해당 애플리케이션이 PACKET에 관한 API 들을 사용할 수 있도록 하였다. 이 밖에도 DEVICE\_READ 퍼미션을 통하여 DEVICE에 관한 정보를 읽어올 수 있도록 퍼미션

```
onos> review org.onosproject.ddos12

*****
SM-ONOS APP REVIEW
*****
Application name: org.onosproject.ddos12
Application role: USER

Developer specified permissions:
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] HOST_READ
[APP PERMISSION] VNET_WRITE_1
[APP PERMISSION] PACKET_EVENT
[APP PERMISSION] PACKET_READ
[APP PERMISSION] PACKET_WRITE
[APP PERMISSION] VNET_READ_1
[APP PERMISSION] APP_WRITE
[APP PERMISSION] TOPOLOGY_READ
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[Other SERVICE] org.onosproject.incubator.net.virtual.VirtualNetworkService(get)
[Other SERVICE] org.onosproject.ddos12.h2hintent(register)
[Other] AdaptPermission (adaptClass=org.osgi.framework.wiring.BundleRevision) (adapt)
[Other] PropertyPermission java.version (read)
[Other] AdminPermission * (class,execute,extensionLifecycle,lifecycle,listener,metadata)
[CRITICAL PERMISSION] RuntimePermission accessDeclaredMembers ()
[CRITICAL PERMISSION] ReflectPermission suppressAccessChecks ()
[CRITICAL PERMISSION] RuntimePermission modifyThread ()

Additional permissions requested on runtime (POLICY VIOLATIONS):
```

그림 29 DDos 방어 애플리케이션의 초기 퍼미션을 설정하였다.

### 2.9.1 DDos 방어 애플리케이션을 통한 공격 시나리오

이 애플리케이션이 본래의 자신의 목적에 벗어나 악성의 함수를 통하여 SDN 네트워크를 공격하는 하나의 시나리오으로써 Flow Rule Flooding Attack 이 존재할 수 있다. Flow Rule Flooding Attack은 악성의 애플리케이션이 수많은 flow rule을 설치하여 target switch의 flow table을 overflow 시키는 공격이다. Flow Rule Flooding Attack을 받은 target switch는 예측 불가능한 state에 놓이게 된다. 이 절에서의 시나리오는 직접 flow rule을 설치 하는 것이 아닌 호스트와 호스트간의 intent를 설치함으로써 공격을 하게 된다.

### 2.9.2 INTENT : ONOS에서 가장 추상적인 프로그래밍 모델

ONOS에서 네트워크의 제어에 대한 추상화로 Flow Rule, Flow Objective, Intent 를 제공한다. 이 중, Intent가 그림 30과 같이 가장 높은 수준의 추상화 인터페이스를 제공한다. Intent의 경우에는 정책 기반 프로그래밍 모델로, 기능을 자동화된 프레임워크를 통하여 Flow Rule로 변환되어 각 경로에 위치하고 있는 스위치들에 설치된다. 따라서 애플리케이션에서 HostToHostIntent를 이용하여 호스와 호스트 간의 Intent를 설치하면, 두 호스트 사이의 네트워크 연결 설정을 위한 Flow Rule을 만들어 스위치에 설치하게 된다. 애플리케이션에서 정의한 호스트 간의 Intent를

Intent service API를 거쳐 Intent 프레임워크에 전달된다. Intent 프레임워크는 먼저 전달받은 Intent를 토대로 패킷의 송신, 수신을 가능하게 하기 위한 Path Intent들을 생성하고, 호스트와 호스트간의 경로에 위치한 각 스위치들에 이 Path Intent들이 Flow Rule로 변환되어 설치된다.

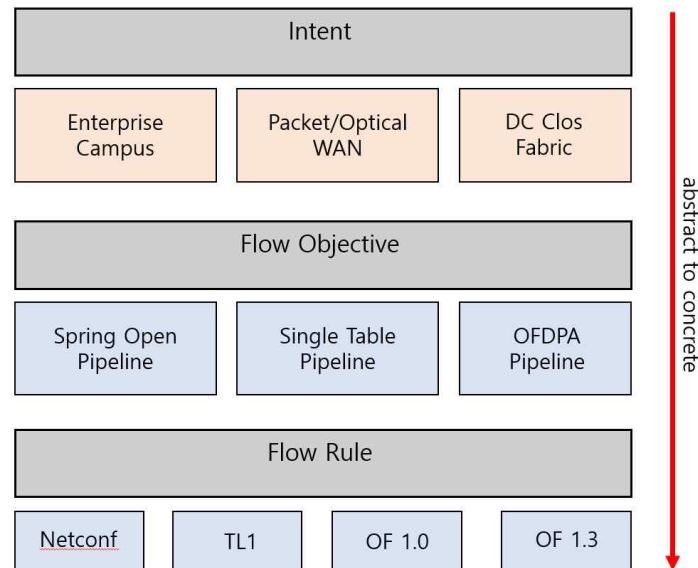


그림 30 INTENT : 가장 추상적인 프로그래밍 모델

### 2.9.3 DDos 방어 애플리케이션을 통한 App-level 접근 제어 평가

```
private class InternalHostListener implements HostListener {
    @Override
    public void event(HostEvent hostEvent) {
        switch (hostEvent.type()) {
            case HOST_ADDED:
                addConnectivity(hostEvent.subject());
                hosts.add(hostEvent.subject());
                break;
            case HOST_REMOVED:
                break;
            case HOST_UPDATED:
                break;
            case HOST_MOVED:
                break;
        }
    }

    private void addConnectivity(Host host) {
        for (Host dst : hosts) {
            HostToHostIntent intent = HostToHostIntent.builder().appId(appId).one(host.id()).
            two(dst.id()).build();
            intentService.submit(intent);
        }
    }
}
```

그림 31 호스트간의 Intent를 설치하는 코드

먼저, App-level 접근 제어가 정상적으로 기능하는지 알아보기 위하여 이 절에서

사용되는 DDos 방어 애플리케이션에 그림 31과 같은 코드를 추가하여 호스트와 호스트 간의 Intent를 설치하게 하는 기능을 추가하였다. 해당 애플리케이션을 실행시키기 위하여 먼저 review [app-name] accept 명령어를 통하여 그림 32와 같이 퍼미션을 accept 하였다.

```
Permissions granted:
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] HOST_READ
[APP PERMISSION] VNET_WRITE_1
[APP PERMISSION] PACKET_EVENT
[APP PERMISSION] PACKET_READ
[APP PERMISSION] PACKET_WRITE
[APP PERMISSION] VNET_READ_1
[APP PERMISSION] TOPOLOGY_READ
[APP PERMISSION] APP_WRITE
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[Other SERVICE] org.onosproject.incubator.net.virtual.VirtualNetworkService(get)
[Other SERVICE] org.onosproject.ddos12.h2hintent(register)
[Other] AdaptPermission (adaptClass=org.osgi.framework.wiring.BundleRevision) (adapt)
[Other] PropertyPermission java.version (read)
[Other] AdminPermission * (class,execute,extensionLifecycle,lifecycle,listener,metadata)
[CRITICAL PERMISSION] RuntimePermission accessDeclaredMembers ()
[CRITICAL PERMISSION] ReflectPermission suppressAccessChecks ()
[CRITICAL PERMISSION] RuntimePermission modifyThread ()

Additional permissions requested on runtime (POLICY VIOLATIONS):
```

그림 32 DDos 방어 애플리케이션의 퍼미션 accept

그 후, 그림 33과 같이 app activate [app-name] 명령어를 통하여 해당 애플리케이션을 활성화시켰다.

```
onos> app activate org.onosproject.ddos12
Activated org.onosproject.ddos12
```

그림 33 DDos 방어 애플리케이션의 활성화

해당 애플리케이션을 활성화시키는 과정에서는 어떠한 access deny 오류가 발생하지도 않고 정상적으로 실행되었다. 하지만, mininet을 통해 생성된 host들 사이에 pingall 명령을 통하여 ping을 보내 ONOS controller가 호스트들 인식하고, 이렇게 인식된 호스트들 사이에 그림 31의 코드를 통하여 Intent를 설치하려는 순간, 그림 34와 같이 AccessControlException : access denied("~.AppPermission" "STORAGE\_WRITE") 오류가 발생하게 된다. 즉, Intent를 설치하게 되는 과정에서 ONOS 컨트롤러의 내부 STORAGE에 저장하는 부분이 필요하여 STORAGE\_WRITE 퍼미션을 요구하는 api를 호출 함에도 그림 29에서 보여지는 것과 같이 해당 애플리케이션은 STORAGE\_WRITE 퍼미션을 가지고 있지 않기 때문이다.



```
com.google.common.util.concurrent.UncheckedExecutionException: java.security.AccessControlException: access denied
object.security.AppPermission" "STORAGE_WRITE")
    at com.google.common.cache.LocalCache$Segment.get(LocalCache.java:2218)
    at com.google.common.cache.LocalCache.get(LocalCache.java:4147)
    at com.google.common.cache.LocalCache$LocalManualCache.get(LocalCache.java:5053)
    at org.onosproject.security.AppGuard$PermissionCheckCache.checkCache(AppGuard.java:74)
```

그림 34 STORAGE\_WRITE 접근 거부 오류

이번에는 그림 35에서 나타난 것과 같이 STORAGE\_WRITE 권한을 애플리케이션에 추가적으로 부여하였다. review 명령어를 통하여 퍼미션 목록에 STORAGE\_WRITE 권한이 추가된 것을 확인할 수 있다.

```
onos> review org.onosproject.ddos12

*****
SM-ONOS APP REVIEW
*****
Application name: org.onosproject.ddos12
Application role: USER

Developer specified permissions:
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] HOST_READ
[APP PERMISSION] STORAGE_WRITE Added Permission
[APP PERMISSION] VNET_WRITE_1
[APP PERMISSION] PACKET_EVENT
[APP PERMISSION] PACKET_READ
[APP PERMISSION] PACKET_WRITE
[APP PERMISSION] VNET_READ_1
[APP PERMISSION] APP_WRITE
[APP PERMISSION] TOPOLOGY_READ
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[Other SERVICE] org.onosproject.incubator.net.virtual.VirtualNetworkService(get)
[Other SERVICE] org.onosproject.ddos12.h2hintent(register)
[Other] AdaptPermission (adaptClass=org.osgi.framework.wiring.BundleRevision) (adapt)
[Other] PropertyPermission java.version (read)
[Other] AdminPermission * (class,execute,extensionLifecycle,lifecycle,listener,metadata
text,weave)
[CRITICAL PERMISSION] RuntimePermission accessDeclaredMembers ()
[CRITICAL PERMISSION] ReflectPermission suppressAccessChecks ()
[CRITICAL PERMISSION] RuntimePermission modifyThread ()

Additional permissions requested on runtime (POLICY VIOLATIONS):
```

그림 35 DDos 방어 애플리케이션에 STORAGE\_WRITE 권한 추가

다시 한번, mininet에서 pingall을 통하여 host event를 발생시켜 그림 31의 코드를 통하여 Intent를 설치하는 명령을 보내면 그림 36와 같이 AccessControlException : access denied("~.AppPermission" "INTENT\_WRITE") 오류가 발생하게 된다. STORAGE\_WRITE 퍼미션과 관련된 api는 호출 할 수 있게 되었지만, INTENT\_WRITE 퍼미션은 부여되어 있지 않기 때문에 이와 관련된 API는 호출할 수 없기 때문이다. 그림 37에서 볼 수 있듯, IntentService.submit() 함수는 INTENT\_WRITE 퍼미션이 존재해야 호출 할 수 있는데 그림 31의 코드에서 intentService.submit(intent)를 호출하기 때문에 INTENT\_WRITE access deny 오류가 발생하게 된다.

```
com.google.common.util.concurrent.UncheckedExecutionException: java.security.AccessControlException: access denied
("org.onosproject.security.AppPermission" "INTENT_WRITE")
    at com.google.common.cache.LocalCache$Segment.get(LocalCache.java:2218) [44:com.google.guava:22.0.0]
    at com.google.common.cache.LocalCache.get(LocalCache.java:4147) [44:com.google.guava:22.0.0]
    at com.google.common.cache.LocalCache$LocalManualCache.get(LocalCache.java:5053) [44:com.google.guava:22.0.0]
```

그림 36 INTENT\_WRITE 접근 거부 오류

INTENT_WRITE	Permission to add/remove intents	Intent Service	submit withdraw purge
		Intent Extension Service	registerCompiler unregisterCompiler

그림 37 INTENT\_WRITE에 포함되어 있는 API들

따라서 access가 거부 되었으므로 onos에서 intents 명령어를 통하여 설치된 intent의 리스트를 보려 해도, 그림 38과 같이 intents의 설치가 거부되어 어떠한 목록도 나오지 않는다는 것을 확인 할 수 있다.

```
onos> intents
onos> █
```

그림 38 INTENT 설치 실패

```
onos> review org.onosproject.ddos12
*****
SM-ONOS APP REVIEW
*****
Application name: org.onosproject.ddos12
Application role: USER

Developer specified permissions:
[APP PERMISSION] DEVICE_READ
[APP PERMISSION] INTENT_WRITE Added Permission
[APP PERMISSION] HOST_READ
[APP PERMISSION] STORAGE_WRITE Added Permission
[APP PERMISSION] VNET_WRITE_1
[APP PERMISSION] PACKET_EVENT
[APP PERMISSION] PACKET_READ
[APP PERMISSION] PACKET_WRITE
[APP PERMISSION] CLUSTER_READ Added Permission
[APP PERMISSION] VNET_READ_1
[APP PERMISSION] APP_WRITE
[APP PERMISSION] TOPOLOGY_READ
[CLI SERVICE] org.apache.karaf.shell.commands.CommandWithAction(register)
[CLI SERVICE] org.apache.felix.service.command.Function(register)
[CLI SERVICE] org.osgi.service.blueprint.container.BlueprintContainer(register)
[CLI SERVICE] org.apache.karaf.shell.console.CompletableFunction(register)
[Other SERVICE] org.onosproject.incubator.net.virtual.VirtualNetworkService(get)
[Other SERVICE] org.onosproject.ddos12.h2hintent(register)
[Other] AdaptPermission (adaptClass=org.osgi.framework.wiring.BundleRevision) (adapt)
[Other] PropertyPermission java.version (read)
[Other] AdminPermission * (class,execute,extensionLifecycle,lifecycle,listener,metadata,e,resource,startlevel,context,weave)
[CRITICAL PERMISSION] RuntimePermission accessDeclaredMembers ()
[CRITICAL PERMISSION] ReflectPermission suppressAccessChecks ()
[CRITICAL PERMISSION] RuntimePermission modifyThread ()

Additional permissions requested on runtime (POLICY VIOLATIONS):
```

그림 39 DDos 방어 애플리케이션에 필요한 권한 모두 추가

이번에는 그림 39와 같이 이 애플리케이션의 악성 함수 부분을 실행하는데 필요한 퍼미션을 모두 부여하고 다시 같은 명령을 반복하였다. 그림 40과 같이 Intents가 정상적으로 설치되는 것을 볼 수 있다.

```
onos> intents
Id: 0x0
State: INSTALLED
Key: 0x0
Intent type: HostToHostIntent
Application Id: org.onosproject.ddos12
Resources: [1E:2C:2E:9A:83:50/1, A6:8D:77:E2:2B:83/1]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]]]
Source host: 1E:2C:2E:9A:83:50/1
Destination host: A6:8D:77:E2:2B:83/1

Id: 0x1
State: INSTALLED
Key: 0x1
Intent type: HostToHostIntent
Application Id: org.onosproject.ddos12
Resources: [B6:25:80:DB:70:38/1, A6:8D:77:E2:2B:83/1]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]]]
Source host: B6:25:80:DB:70:38/1
Destination host: A6:8D:77:E2:2B:83/1

Id: 0x2
State: INSTALLED
Key: 0x2
Intent type: HostToHostIntent
Application Id: org.onosproject.ddos12
Resources: [B6:25:80:DB:70:38/1, 1E:2C:2E:9A:83:50/1]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]]]
```

그림 40 INTENT 설치 성공

결론적으로, 애플리케이션이 악성 기능을 실행시키려 할 때, 이에 필요한 권한 중 어느 하나만 부여되어 있지 않다면, 악성 기능은 원천적으로 차단되기 때문에 SDN 네트워크의 보안을 유지할 수 있고, Multi-layered Security Mode ONOS를 통하여 성공적으로 이러한 컨트롤을 제어할 수 있다는 것을 이 절을 통하여 확인 할 수 있었다.

#### 2.9.4 DDos 방어 애플리케이션을 통한 Virtual Network-level 접근 제어 평가

현재, DDos를 방어하는 이 어플리케이션에는 VNET\_READ\_1 과 VNET\_WRITE\_1을 통하여 Virtual Network 1번에 대하여만 읽고 쓸 권한을 부여해 준 상태이다. 따라서 vnet-no가 1이 아닌 다른 Virtual Network에 대해서는 권한을 가질 수 없는지 평가할 필요가 있다. 특정 어플리케이션이 특정 virtual network에 대한 접근 제어 기능을 가능하게 하기 위해서는 grantvnetperm [app-name] [READ/WRITE/EVENT] [vnet-no]를 통하여 권한을 줄 필요가 있다. 따라서 먼저 org.onosproject.ddos12 어플리케이션에 4번 Virtual Network에 대하여 그림 41과 같이 READ 권한을 부여하게 되면 그림 42와 같이 java.lang.IllegalArgumentException 오류가 뜨게 된다. 이번에는 3번 Virtual

Network에 대하여 WRITE 권한을 부여하게 되면 같은 java.lang.IllegalArgumentException 오류가 발생한다. 이번에는 1번 Virtual Network에 대하여 READ 권한을 부여해주게 되면, VNET\_READ\_1은 이미 부여된 권한이므로 단순히 update 된다.

```
onos> app activate org.onosproject.ddos12
Activated org.onosproject.ddos12
onos> grantvnetperm org.onosproject.ddos12 READ 4
Error executing command: No enum constant org.onosproject.security.AppPermission.Type.VNET_READ_4
```

그림 41 VNET 4번 id에대한 READ 권한 부여 실패

```
java.lang.IllegalArgumentException: No enum constant org.onosproject.security.AppPermission.Type.VNET_READ_4
    at java.lang.Enum.valueOf(Enum.java:238)[:1.8.0_191]
    at org.onosproject.security.AppPermission$Type.valueOf(AppPermission.java:29)
    at org.onosproject.incubator.net.virtual.impl.VirtualNetworkManager.getVirtualNetwork(VirtualNetworkManager.java:327)
```

그림 42 VNET\_READ\_4 access deny오류

따라서 Multi-layered Security Mode ONOS가 virtual network에 대한 접근 제어도 성공적으로 기능한다는 것을 알 수 있었다.

## 2.10. 호스트 레벨 접근제어 기능을 추가한 Security-Mode ONOS 사용법

### o Host-level Access Control 기본 사용 방법

호스트 레벨 접근제어 기능을 구현한 hostAC 애플리케이션은 ONOS CLI를 통해 Policy를 추가/삭제/확인할 수 있는 인터페이스를 제공한다. 기본적인 CLI 명령어는 아래와 같다.

\$ hostac [command-type]

command-type에는 policy를 추가할 수 있는 ADD, policy를 삭제할 수 있는 REMOVE, 모든 policy를 확인할 수 있는 GET으로 구성되어 있다.

### o Host-level Access Control ADD 명령어

hostAC 애플리케이션의 ADD 명령어는 아래와 같은 구조를 가진다. 각각의 파라미터에 대한 설명은 다음과 같다. source-ip는 접근이 필요한 IP주소, destination-ip는 목적지 IP주소, protocol-type은 ICMP, TCP와 같은 프로토콜 타입, 마지막으로 destination-port는 목적지 포트번호를 의미한다. destination-port를 제외한 나머지는 ADD 명령을 수행할 때 필수적으로 적어야 한다.

\$ hostac ADD [source-ip] [destination-ip] [protocol-type] [destination-port]

그림 43은 ICMP 서비스를 이용하기 위해 실제 네트워크 환경에서 정책들을 추가한 결과이다. IP주소 172.30.94.4와 172.30.95.4를 가진 각각의 호스트에서 ICMP 프로토콜과 관련된 네트워크 트래픽을 주고받을 수 있도록 policy를 넣었음을 알 수 있다. 그 결과로 현재 hostAC 애플리케이션이 가지고 있는 총 policy의 개수를 받



환하는 것을 확인할 수 있다.

```
onos> hostac ADD 172.30.95.4/32 172.30.94.4/32 ICMP  
Rule Count: 1  
  
onos> hostac ADD 172.30.94.4/32 172.30.95.4/32 ICMP  
Rule Count: 2
```

그림 43 Host-level Access Control 애플리케이션 ADD 명령어 예시

o Host-level Access Control REMOVE 명령어

hostAC 애플리케이션의 REMOVE는 아래와 같은 구조를 가진다. 앞서 설명한 ADD와 동일한 구조를 가지므로 각 파라미터에 대한 설명은 생략한다.

```
$ hostac REMOVE [source-ip] [destination-ip] [protocol-type] [destination-port]
```

그림 44는 앞서 추가한 정책을 삭제한 결과이다. IP주소 172.30.94.4와 172.30.95.4를 가진 각각의 호스트에서 ICMP 서비스를 이용하기 위한 두 개의 policy를 REMOVE 명령으로 삭제하여 권한을 제거하였으며, 각 명령어의 실행 결과로 현재 hostAC 애플리케이션이 가지고 있는 총 policy의 개수를 반환하는 것을 확인할 수 있다.

```
onos> hostac REMOVE 172.30.94.4/32 172.30.95.4/32 ICMP  
Rule Count: 1  
onos> hostac REMOVE 172.30.95.4/32 172.30.94.4/32 ICMP  
Rule Count: 0
```

그림 44 Host-level Access Control 애플리케이션 REMOVE 명령어 예시

o Host-level Access Control GET 명령어

hostAC 애플리케이션의 GET은 아래와 같은 구조를 가진다. 앞서 설명한 ADD와 REMOVE 명령과 달리 GET 명령은 특별한 파라미터를 요구하지 않는다.

```
$ hostAC GET
```

그림 45은 앞서 ADD 명령으로 두 개의 policy를 추가한 상태에서 GET 명령어를 실행한 결과이다. GET 명령어를 입력한 결과, 저장한 모든 policy가 주요 파라미터 값과 함께 반환되는 것을 확인할 수 있다.

```
onos> hostac GET
1. srcIP: 172.30.95.4/32, dstIP: 172.30.94.4/32, protocolType: ICMP
2. srcIP: 172.30.94.4/32, dstIP: 172.30.95.4/32, protocolType: ICMP
```

Rule Count: 2

그림 45 Host-level Access Control 애플리케이션 GET 명령어 예시 1

그림 46은 앞서 REMOVE 명령으로 ADD 명령으로 추가한 두 개의 policy를 삭제한 상태에서 GET 명령어를 실행한 결과이다. GET 명령어를 입력한 결과, 저장한 policy가 존재하지 않으므로 아무런 policy도 반환하지 않는 것을 확인할 수 있다.


```
onos> hostac GET
```

Rule Count: 0

그림 46 Host-level Access Control 애플리케이션 GET 명령어 예시 2

GET 명령어는 ADD 명령과 REMOVE와 다르게, 그림 47과 같이 ONOS가 제공하는 Web 서비스를 통해 현재 저장된 모든 policy를 확인할 수 있다. 접근할 수 있는 URI는 아래와 같다.

[http://\[controller-ip:port\]/onos/v1/hostAC/rules](http://[controller-ip:port]/onos/v1/hostAC/rules)



```
srcIP: 172.30.94.4/32, dstIP: 172.30.95.4/32, type: ICMP / srcIP:
172.20.95.6/32, dstIP: 172.20.95.10/32, type: TCP, dstPort: 5566 /
srcIP: 172.20.95.10/32, dstIP: 172.20.95.6/32, type: UDP, dstPort: 6655
/ srcIP: 172.20.95.4/32, dstIP: 172.20.95.8/32, type: TCP, dstPort: 80 /
```

그림 47 Host-level Access Control 애플리케이션 Web 서비스

#### o Security-Mode ONOS의 호스트 레벨 접근제어 예시 및 테스트

구현한 호스트 레벨 접근 제어 메커니즘이 정상적으로 접근 제어를 수행하는지 확인하기 위해, 실제 네트워크 환경에서 하나의 호스트에서 다른 호스트에 접근할 수 있는 권한 여부에 따른 접근제어 결과를 테스트하였다. 테스트 시나리오는 다음과 같다. IP주소 172.30.94.4를 가진 호스트 A에서 172.30.95.4를 가진 호스트 B로 ICMP 프로토콜을 사용하는 PING 서비스를 통해 권한유무에 따른 각각 통신의 가능 여부를 확인하여 호스트 레벨 접근제어 서비스를 검증한다.

그림 48은 앞서 설명한 ADD 명령을 통해 관련된 policy (첫 번째, 두 번째)를 넣어 마지막 GET 명령을 통해 가지고 있는 총 policy들을 반환하여 안전하게 저장되

었다는 것을 확인할 수 있다.

```
onos> hostac ADD 172.30.95.4/32 172.30.94.4/32 ICMP
Rule Count: 1
onos> hostac ADD 172.30.94.4/32 172.30.95.4/32 ICMP
Rule Count: 2
onos>
onos> hostac ADD 172.20.95.6/32 172.20.95.10/32 TCP 5566
Rule Count: 3
onos> hostac ADD 172.20.95.10/32 172.20.95.6/32 UDP 6655
Rule Count: 4
onos>
onos> hostac ADD 172.20.95.4/32 172.20.95.8/32 TCP 80
Rule Count: 5
onos>
onos> hostac GET
1. srcIP: 172.30.95.4/32, dstIP: 172.30.94.4/32, protocolType: ICMP
2. srcIP: 172.30.94.4/32, dstIP: 172.30.95.4/32, protocolType: ICMP
3. srcIP: 172.20.95.6/32, dstIP: 172.20.95.10/32, protocolType: TCP, dstPort: 5566
4. srcIP: 172.20.95.10/32, dstIP: 172.20.95.6/32, protocolType: UDP, dstPort: 6655
5. srcIP: 172.20.95.4/32, dstIP: 172.20.95.8/32, protocolType: TCP, dstPort: 80
Rule Count: 5
```

그림 48 Host-level Access Control 애플리케이션 권한 부여 테스트

그림 49는 그림 47의 결과에 따라 각각의 호스트에 ICMP 서비스에 대한 권한이 부여된 상태에서의 PING 테스트 결과 서로 통신이 가능함을 확인할 수 있다.

```
PING 172.30.94.4 (172.30.94.4) 56(84) bytes of data.
64 bytes from 172.30.94.4: icmp_seq=1 ttl=64 time=14.1 ms
64 bytes from 172.30.94.4: icmp_seq=2 ttl=64 time=0.573 ms
64 bytes from 172.30.94.4: icmp_seq=3 ttl=64 time=0.068 ms
64 bytes from 172.30.94.4: icmp_seq=4 ttl=64 time=0.031 ms
64 bytes from 172.30.94.4: icmp_seq=5 ttl=64 time=0.047 ms
64 bytes from 172.30.94.4: icmp_seq=6 ttl=64 time=0.040 ms
64 bytes from 172.30.94.4: icmp_seq=7 ttl=64 time=0.087 ms
64 bytes from 172.30.94.4: icmp_seq=8 ttl=64 time=0.082 ms
64 bytes from 172.30.94.4: icmp_seq=9 ttl=64 time=0.058 ms
64 bytes from 172.30.94.4: icmp_seq=10 ttl=64 time=0.064 ms

--- 172.30.94.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9002ms
rtt min/avg/max/mdev = 0.031/1.522/14.175/4.220 ms
```

그림 49 Host-level Access Control 애플리케이션 권한 부여 테스트 결과 1

그림 50은 통신이 가능할 때의 ONOS 컨트롤러의 GUI를 캡처한 화면이다. 앞서 PING 테스트 결과와 동일하게 전달이 가능하도록 하는 우선순위 10의 2개의 플로우룰이 설치된 것을 확인할 수 있다.

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT
Added	105	930	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true
Added	301	930	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true
Added	301	930	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true
Added	2	930	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true
Added	1	1	10	0	IN_PORT:1, ETH_DST:00:00:00:00:00:02, ETH_SRC:00:00:00:00:00:01	imm[OUTPUT:2], cleared:false
Added	1	1	10	0	IN_PORT:2, ETH_DST:00:00:00:00:00:01, ETH_SRC:00:00:00:00:00:02	imm[OUTPUT:1], cleared:false

그림 50 Host-level Access Control 애플리케이션 권한 부여 테스트 결과 2

그림 51은 REMOVE 명령을 통해 기존에 부여된 권한을 제거한 것을 확인할 수 있다. 따라서 호스트 레벨 접근제어 서비스가 올바르게 작동된다면 호스트 A와 호스트 B 사이의 ICMP 서비스 사용이 불가할 것이다. GET 명령의 결과 관련된 policy가 삭제된 것을 확인할 수 있다.

```
onos> hostac REMOVE 172.30.95.4/32 172.30.94.4/32 ICMP
Rule Count: 4
onos> hostac GET
1. srcIP: 172.30.94.4/32, dstIP: 172.30.95.4/32, protocolType: ICMP
2. srcIP: 172.20.95.6/32, dstIP: 172.20.95.10/32, protocolType: TCP, dstPort: 5566
3. srcIP: 172.20.95.10/32, dstIP: 172.20.95.6/32, protocolType: UDP, dstPort: 6655
4. srcIP: 172.20.95.4/32, dstIP: 172.20.95.8/32, protocolType: TCP, dstPort: 80
```

그림 51 Host-level Access Control 애플리케이션 권한 제거 테스트

그림 52는 그림 50의 결과에 따라 호스트 A에서의 ICMP 서비스에 대한 권한이 제거된 상태에서의 PING 테스트 결과, Reactive 방식으로 접근을 차단하는 것을 확인할 수 있다.

```
PING 172.30.94.4 (172.30.94.4) 56(84) bytes of data.
64 bytes from 172.30.94.4: icmp_seq=1 ttl=64 time=11.6 ms

--- 172.30.94.4 ping statistics ---
10 packets transmitted, 1 received, 90% packet loss, time 9001ms
rtt min/avg/max/mdev = 11.640/11.640/11.640/0.000 ms
```

그림 52 Host-level Access Control 애플리케이션 권한 제거 테스트 결과 1

그림 53은 통신이 불가능할 때의 ONOS 컨트롤러의 GUI를 캡처한 화면이다. 앞서 권한이 있을 때의 결과와 동일하게 전달이 가능하도록 하는 우선순위 10의 2개

의 플로우룰이 설치되었으나 이보다 더 높은 우선순위를 갖는 플로우룰에 의해 통신을 차단하는 것을 확인할 수 있다.

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT
Added	0	5	30000	0	ETH_TYPE:ipv4, IP_PROTO:1, IPV4_SRC:172.30.94.4/32, IPV4_DST:172.30.95.4/32	imm[NOACTION], cleared:false
Added	56	635	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true
Added	206	635	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true
Added	206	635	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true
Added	2	635	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true
Added	5	5	10	0	IN_PORT:1, ETH_DST:00:00:00:00:00:02, ETH_SRC:00:00:00:00:00:01	imm[OUTPUT:2], cleared:false
Added	0	5	10	0	IN_PORT:2, ETH_DST:00:00:00:00:00:01, ETH_SRC:00:00:00:00:00:02	imm[OUTPUT:1], cleared:false

그림 53 Host-level Access Control 애플리케이션 권한 제거 테스트 결과 2



### 3. 결론

현재 SDN 컨트롤러들은 오픈소스 기반으로 활발하게 개발되어지고 있지만, ONOS를 포함한 대부분의 JAVA기반의 오픈소스 SDN 컨트롤러들이 보안을 고려하지 않고 디자인되었다. 이에 따라 하나의 JVM위에 컨트롤러와 모든 애플리케이션이 동작하고, 하나의 애플리케이션이 SDN컨트롤러가 제공하는 모든 서비스들에 접근가능하며 JVM에도 직접적으로 접근이 가능한, 굉장히 큰 권한을 가지는 취약한 구조를 가지고 있다. 이러한 취약한 구조를 보완하기 위해서는, SDN 애플리케이션의 안전성을 테스트하는 효율적인 SDN application security-instrumentation 이나, 액세스 컨트롤 메카니즘이 필요하다.

본 기술문서에서 설명한 Security-Mode ONOS는 보다 안전한 SDN 환경을 만들기 위해 ONOS 컨트롤러위에 액세스컨트롤 메카니즘을 구현한 것으로, 네트워크 관리자가 애플리케이션을 컨트롤러위에 활성화시키기 전에 애플리케이션의 행위를 퍼미션 기반으로 이해하고 활성화 시킬지 말지 결정하는 1차 필터링 메카니즘, 그리고 런타임에 애플리케이션이 부여받은 퍼미션 이외의 행위를 하는지 검사하고 이외의 행위에 대해서는 제한하는 메카니즘을 제공한다. 이러한 Northbound쪽의 접근제어 뿐만 아니라, data-plane쪽에 관련된 virtual network의 접근제어 및 호스트 레벨의 접근제어 기능도 제공한다. 또한, Security-Mode ONOS는 추가기능으로 unit test와 permission gap을 활용한 보안강화기능도 제공한다. Security-Mode ONOS는 안전한 SDN 환경을 만들기 위한 첫걸음으로, 앞으로는 이 기능을 확장하여 ONOS 컨트롤러 위에 더 다양한 보안기능 뿐만 아니라 통합된 보안 프레임워크를 만들 계획이다.

## 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] THE APACHE SOFTWARE FOUNDATION. Apache Felix.  
<http://felix.apache.org>.
- [3] THE APACHE SOFTWARE FOUNDATION. Apache Karaf.  
<http://karaf.apache.org>.
- [4] THE APACHE SOFTWARE FOUNDATION. Apache felix framework security.  
<http://felix.apache.org/documentation/subprojects/apache-felix-framework-security.html>.
- [5] A LINUX FOUNDATION COLLABORATIVE PROJECT. Open-Daylight SDN Controller. <http://www.opendaylight.org>.
- [6] BERDE, P., GEROLA, M., HART, J., HIGUCHI, Y., KOBAYASHI, M., KOIDE, T., LANTZ, B., O'CONNOR, B., RADOSLAVOV, P., SNOW, W., ET AL. ONOS: towards an open, distributed SDN OS. In Proceedings of the third workshop on Hot topics in software defined networking (2014), ACM, pp. 1-6.
- [7] ANDROID OPEN SOURCE PROJECT. App Manifest.  
<http://developer.android.com/guide/topics/manifest/manifest-intro.html>.
- [8] K.Hong, L. Xu, H. Wang, and G. Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In Proceedings of the 22nd Annual Network and Distributed System Security Symposium(NDSS15), Feburary 2015.
- [9] C. Ropke and T. Holz. Sdn rootkits: Subverting network operating systems of software-defined networks. In Research in Attacks, Intrusions, and Defenses, pages 339-356. Springer, 2015.
- [10] S. Lee, C. Yoon, and S. Shin. The smaller, the shrewder: A simple malicious application can kill an entire sdn environment. In Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, pages 23-28. ACM, 2016.

- [11] FloodLight. Open sdn controller. <http://floodlight.openflowhub.org>
- [12] Android permission. <http://developer.android.com/reference/android/Manifest.permission.html>.
- [13] Security-Mode ONOS permission.  
<https://wiki.onosproject.org/display/ONOS/ONOS+Application+Permissions>
- [14] Symbolic execution.  
[https://en.wikipedia.org/wiki/Symbolic\\_execution](https://en.wikipedia.org/wiki/Symbolic_execution)



## *K-ONE* 기술 문서

- K-ONE 컨소시엄의 확인과 허가 없이 이 문서를 무단 수정하여 배포하는 것을 금지합니다.
- 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](mailto:k1@opennetworking.kr))

작성기관: K-ONE Consortium  
작성년월: 2019/11