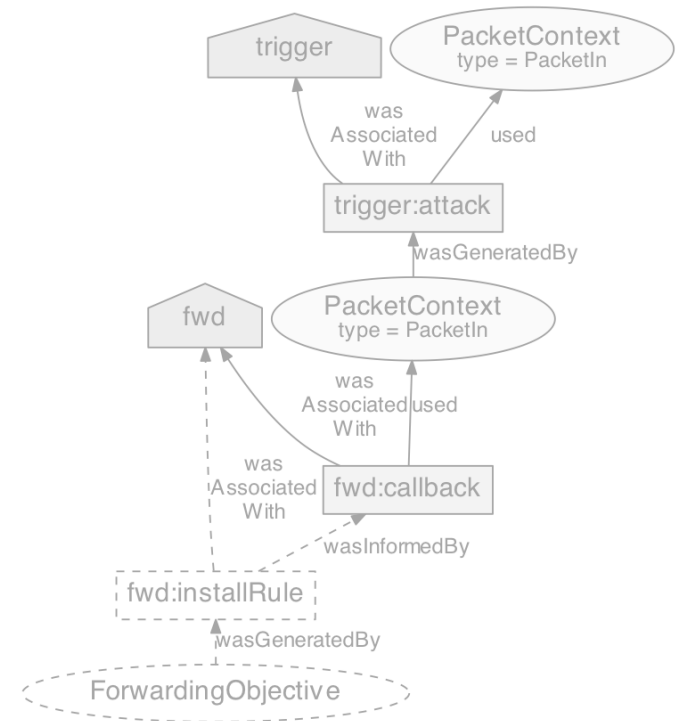


# Cross-App Poisoning in Software-Defined Networking

Benjamin E. Ujcich<sup>1</sup>, Samuel Jero<sup>2</sup>, Anne Edmundson<sup>3</sup>, Qi Wang<sup>1</sup>, Richard Skowrya<sup>2</sup>, James Landry<sup>2</sup>, Adam Bates<sup>1</sup>, William H. Sanders<sup>1</sup>, Cristina Nita-Rotaru<sup>4</sup>, and Hamed Okhravi<sup>2</sup>



1 **I ILLINOIS**

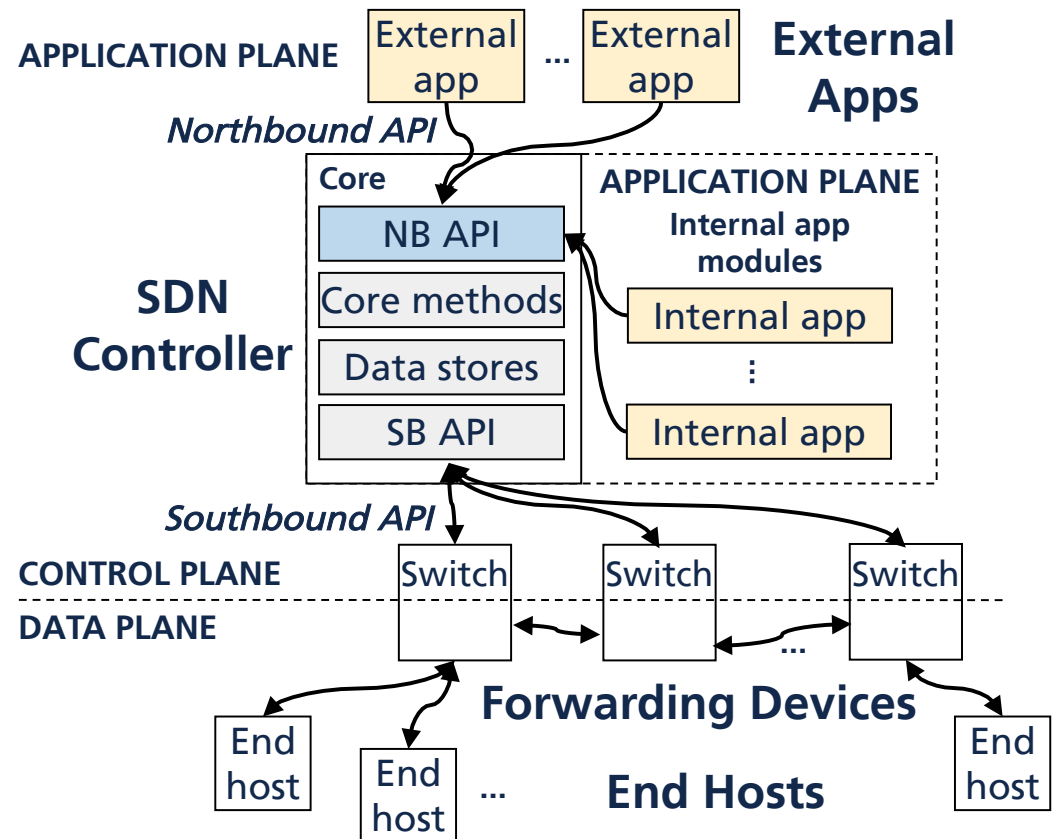
2 **LINCOLN LABORATORY**  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

3 **PRINCETON**  
UNIVERSITY

4 **Northeastern**

# SDN Overview

- SDN centralizes decisions into an **SDN controller**
- SDN controller acts as a **network operating system**
- **Network applications (apps)** extend functionality



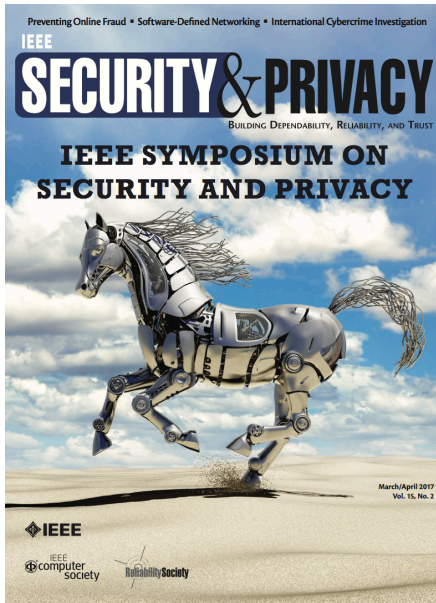
# State of SDN Security

## SYSTEMS ATTACKS AND DEFENSES

Editors: William Enck, wnenck@ncsu.edu | Thorsten Holz, thorsten.holz@rub.de | Angelos Stavrou, astavrou@gmu.edu

## Security Challenges and Opportunities of Software-Defined Networking

Marc C. Dacier | Qatar Computing Research Institute  
Hartmut König and Radosław Cwalinski | Brandenburg University of Technology Cottbus  
Frank Kargl | University of Ulm  
Sven Dietrich | City University of New York



At the beginning of the decade, software-defined networking (SDN) has attracted much attention from both industry and academia, and this trend continues in 2016, the market research firm International Data Corporation predicted that the market for network applications would reach \$3.5 billion by 2020.<sup>1</sup> In industry, the vision of “defining computer networks” has motivated many IT managers and decision makers. Consequently, attention is high regarding SDN. Leading IT companies such as Nokia, Cisco, Dell, HP, IBM, and VMware have each developed their own SDN strategy. Major switch vendors as well as promising start-ups offer programmable switches.

### Round

Since SDN provides a way to reconfigure network infrastructure—simplify it and to configure and

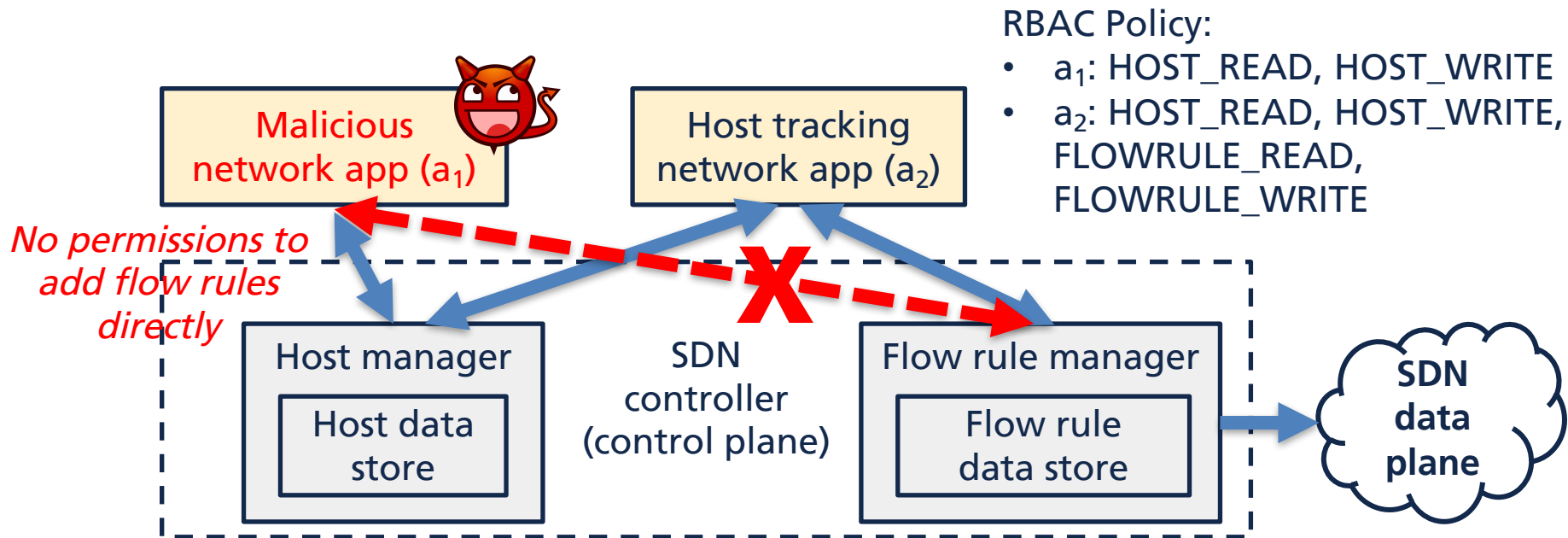
and switches become “slaves” of this application-driven controller.

SDN-enabled networks are capable of supporting user requirements from various business applications (service-level agreements, quality of service, policy management, and so on). Most SDN approaches rely on the widely used OpenFlow protocol to provide communication between controllers and networking equipment.<sup>2</sup> OpenFlow is a vendor-independent standard and thus allows for interoperability between heterogeneous devices. Besides centrally defined routing policies, another key advantage of SDN is that it allows routing choices to be defined at a much finer granularity level, that is, per flow rather than at the usual IP-prefix level. For instance, OpenFlow 1.5 supports 44 different types of header fields against which to match a packet in order to choose the flow it belongs to and, thus, determine the route it should follow.

- *IEEE Se&P* magazine, 2017
  - “Attacks against SDN controllers and ... **malicious controller apps** are probably the **most severe threats** to SDN.”
  - “**Dynamic configurations make it more difficult** for defenders to tell whether the current or past configuration is intended...”

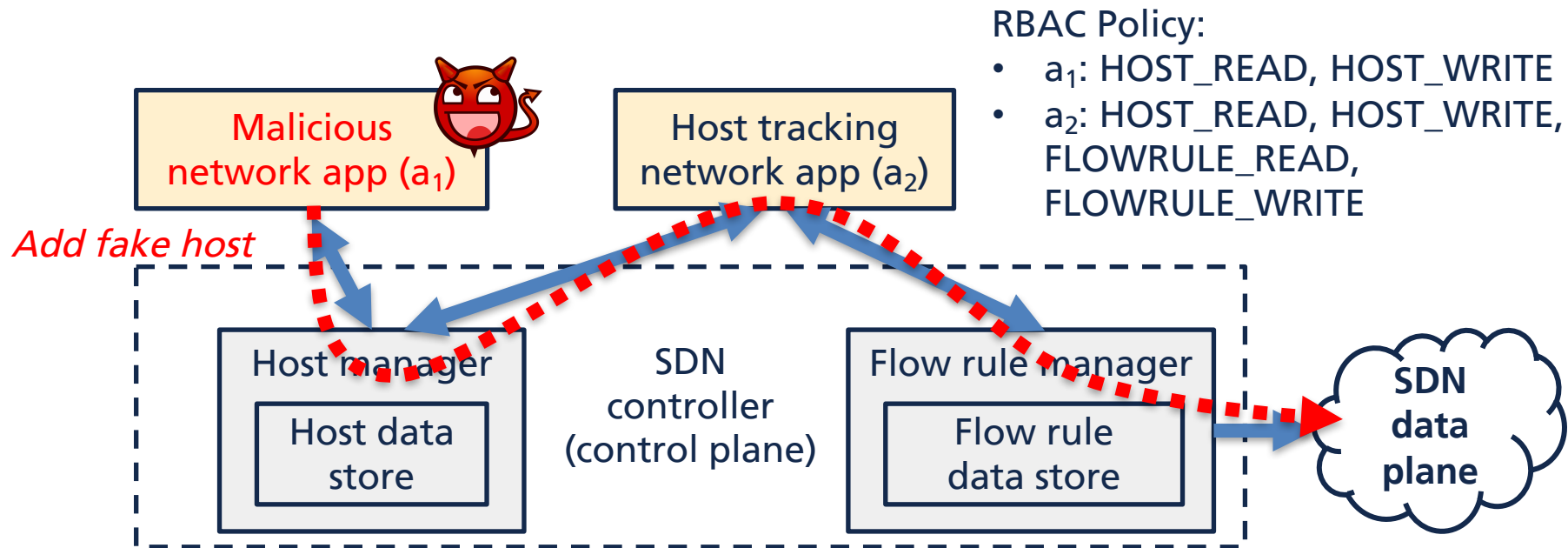
Need for greater insight into network decision-making among apps

# RBAC in Control Plane



Current solutions rely on role-based access control (RBAC)

# RBAC Limitations



RBAC is insufficient because it does not track **information flow**

# Approach

High level goal: Track information flow within the SDN control plane

- Formalize **cross-app poisoning (CAP)**
- Perform static analysis of apps to find **CAP gadgets**
- Incorporate **information flow control (IFC)** in control plane
- Apply **data provenance** techniques to track information flow and **enforce IFC** with minimal additional latency (**ProvSDN**)

# Threat Model



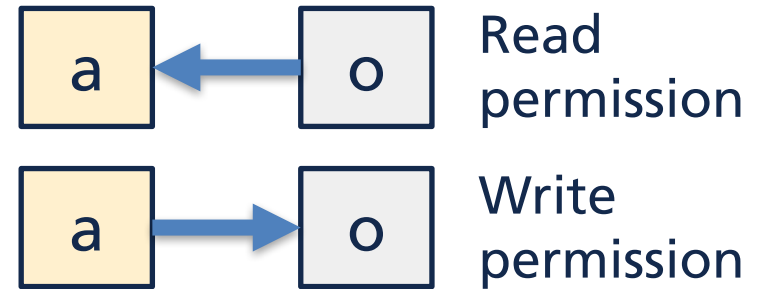
- Attacker objective: arbitrarily install flow rules to affect data plane connectivity
- Defender objective: prevent CAP attacks even after RBAC has been applied
- System assumptions:
  - SDN controller is **trusted** and adequately secured
  - Apps may originate from third parties; **untrusted**
  - Attacker controls a **least-privileges** app

# Cross-App Poisoning (CAP)

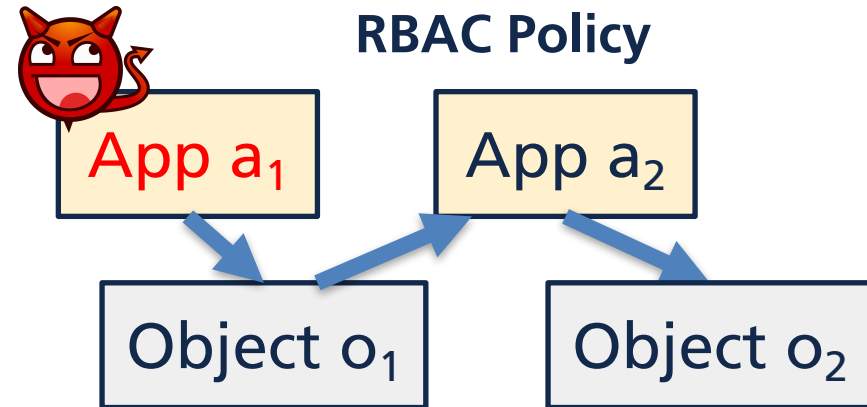
- IFC integrity problem
- Model RBAC policy with **apps**, control plane's **data structures (objects)**, and **read and write permissions (edges)**

Goal: Find paths from apps to objects that are **not directly connected**

## CAP Semantics

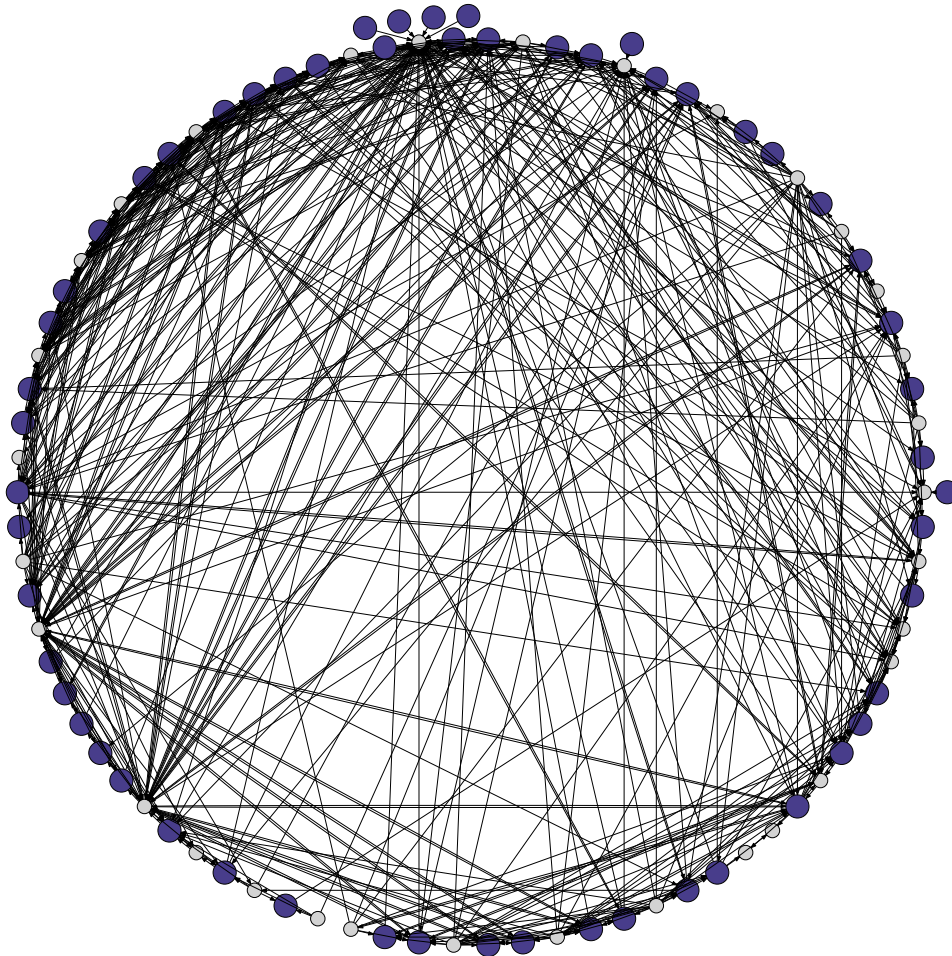


## Example with RBAC Policy

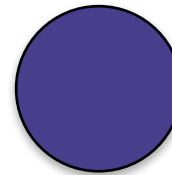




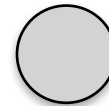
# CAP in ONOS



## CAP for (Security-Mode) ONOS with Least-Privileges RBAC Policy



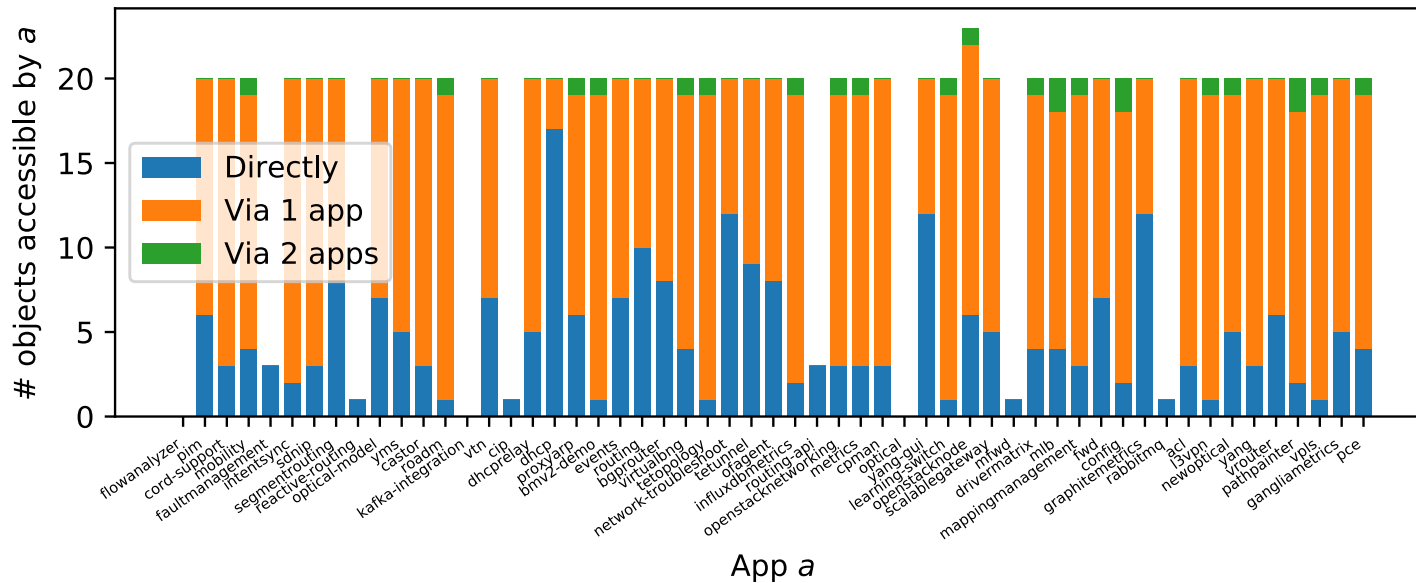
ONOS app



ONOS object  
(data structure)

Strong connectivity  
shows potential **highly  
dependent data**

# CAP in ONOS



- **63** apps (excluding test app)
- **212** protected methods in **39** manager classes

Strong connectivity shows potential highly dependent data

# CAP Gadgets

- Writes may not always causally depend on reads
- Use static analysis
- Identify CAP gadgets that allow flow from a permissioned data source to a permissioned data sink
- Assume the attacker uses a triggering app to start

## Sources and sinks in ONOS forwarding app fwd

```
1 public class ReactiveForwarding {
2     public void activate(...) {
3         ...
4         appId = coreService.registerApplication("org.onosproject.
           ↪ fwd");
5         packetService.addProcessor(processor, PacketProcessor.
           ↪ director(2));
6         ...
7     }
8     private class ReactivePacketProcessor implements Source
           ↪ PacketProcessor {
9         public void process(PacketContext context) {
10            ...
11            installRule(context,...);
12        }
13    }
14    private void installRule(PacketContext context,...) {
15        ...
16        ForwardingObjective forwardingObjective =
           ↪ DefaultForwardingObjective.builder().withSelector(
           ↪ selectorBuilder.build()).withTreatment(treatment).
           ↪ withPriority(flowPriority).withFlag(
           ↪ ForwardingObjective.Flag.VERSATILE).fromApp(appId).
           ↪ makeTemporary(flowTimeout).add(); Sink
17        flowObjectiveService.forward(context.inPacket().
           ↪ receivedFrom().deviceId(), forwardingObjective)
18    }
19 }
```

# CAP Gadgets in ONOS

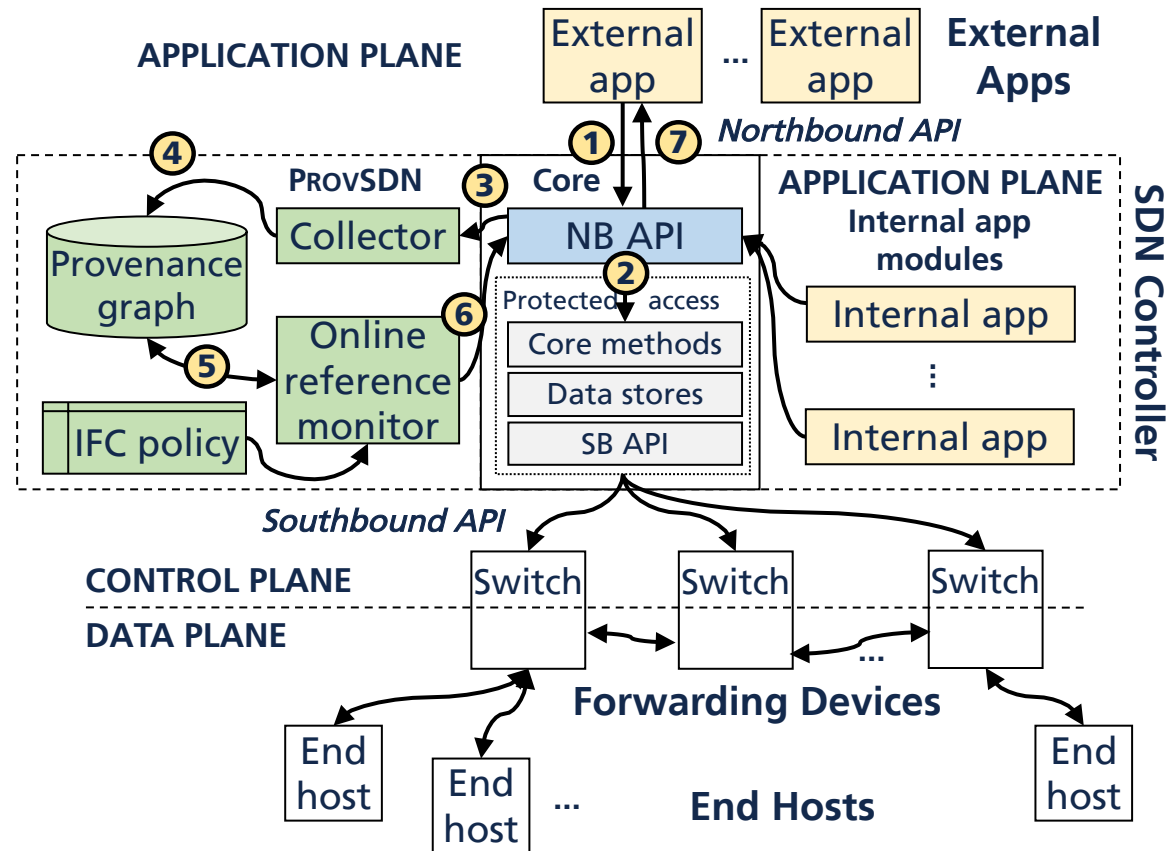
**Table 1: Static Analysis Results of CAP Gadgets for Security-Mode ONOS Apps.**

Source ( $p \in P_R$ )	App ( $a \in A$ )	Sink ( $p \in P_W$ )	Attacker's capabilities if source data have been compromised by attacker
APP_READ	openstacknetworking	FLOWRULE_WRITE	Attacker modifies the app ID to remove all flows with a given app ID
APP_READ	openstacknode	CLUSTER_WRITE	Attacker modifies the app ID to make an app run for leader election in a different ONOS topic ( <i>i.e.</i> , an app using ONOS's distributed primitives)
APP_READ	openstacknode	GROUP_WRITE	Attacker modifies the app ID to associate an app with a particular group handler
APP_READ	routing	CONFIG_WRITE	Attacker modifies the app ID to misapply a BGP configuration
APP_READ	sdnip	CONFIG_WRITE	Attacker modifies the app ID to misapply an SDN-IP encapsulation configuration
DEVICE_READ	newoptical	RESOURCE_WRITE	Attacker misallocates bandwidth resources based on a connectivity ID
DEVICE_READ	vtn	DRIVER_WRITE	Attacker misconfigures driver setup for a device ( <i>i.e.</i> , switch)
DEVICE_READ	vtn	FLOWRULE_WRITE	Attacker misconfigures flow rules based on a device ID
HOST_READ	vtn	FLOWRULE_WRITE	Attacker misconfigures flow rules based on a host with a particular MAC address
PACKET_READ	fw	FLOWRULE_WRITE	Attacker injects or modifies an incoming packet to poison a flow rule
PACKET_READ	learning-switch	FLOWRULE_WRITE	Attacker injects or modifies an incoming packet to poison a flow rule

Attackers can leverage other data structures to affect flow rules without flow rule permissions

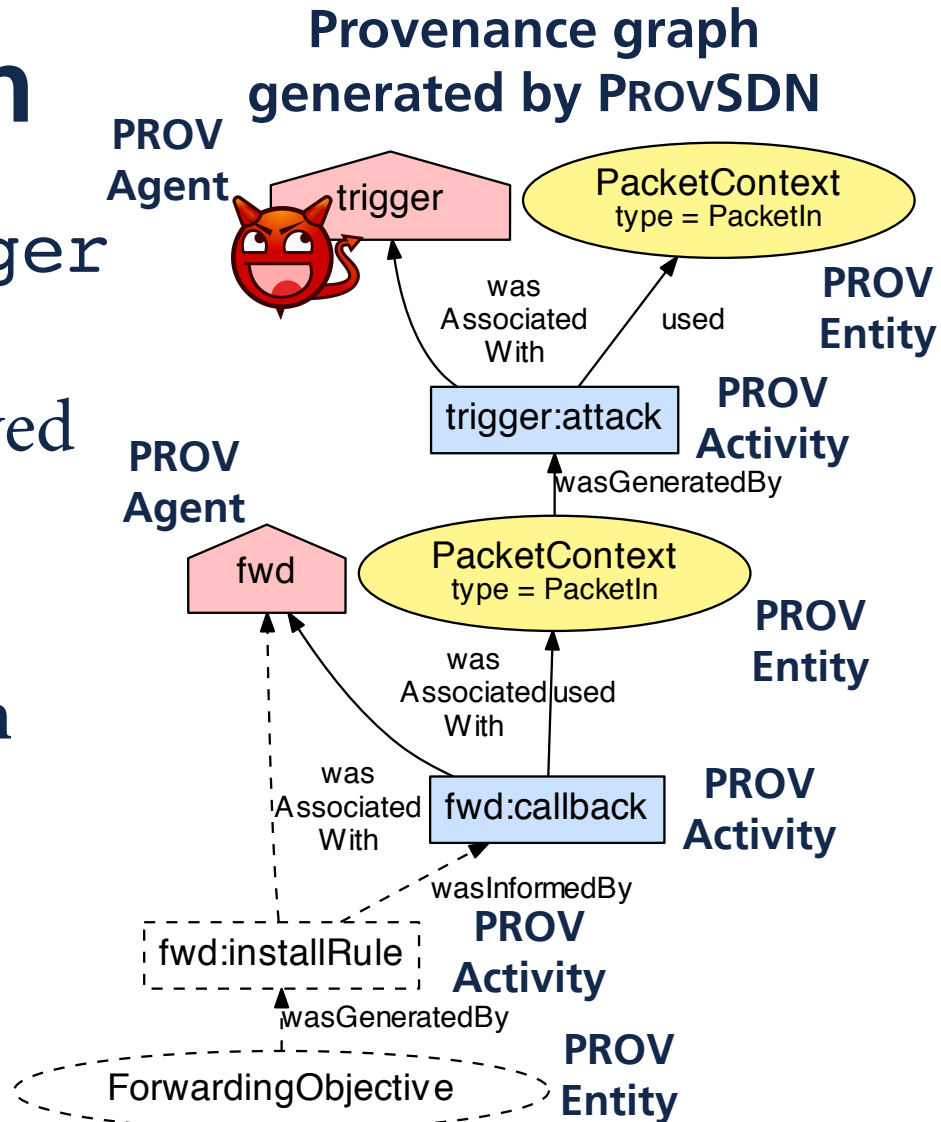
# PROVSDN

- Use **data provenance** to record control plane state
- Online reference monitor enforces IFC
- Implemented on ONOS

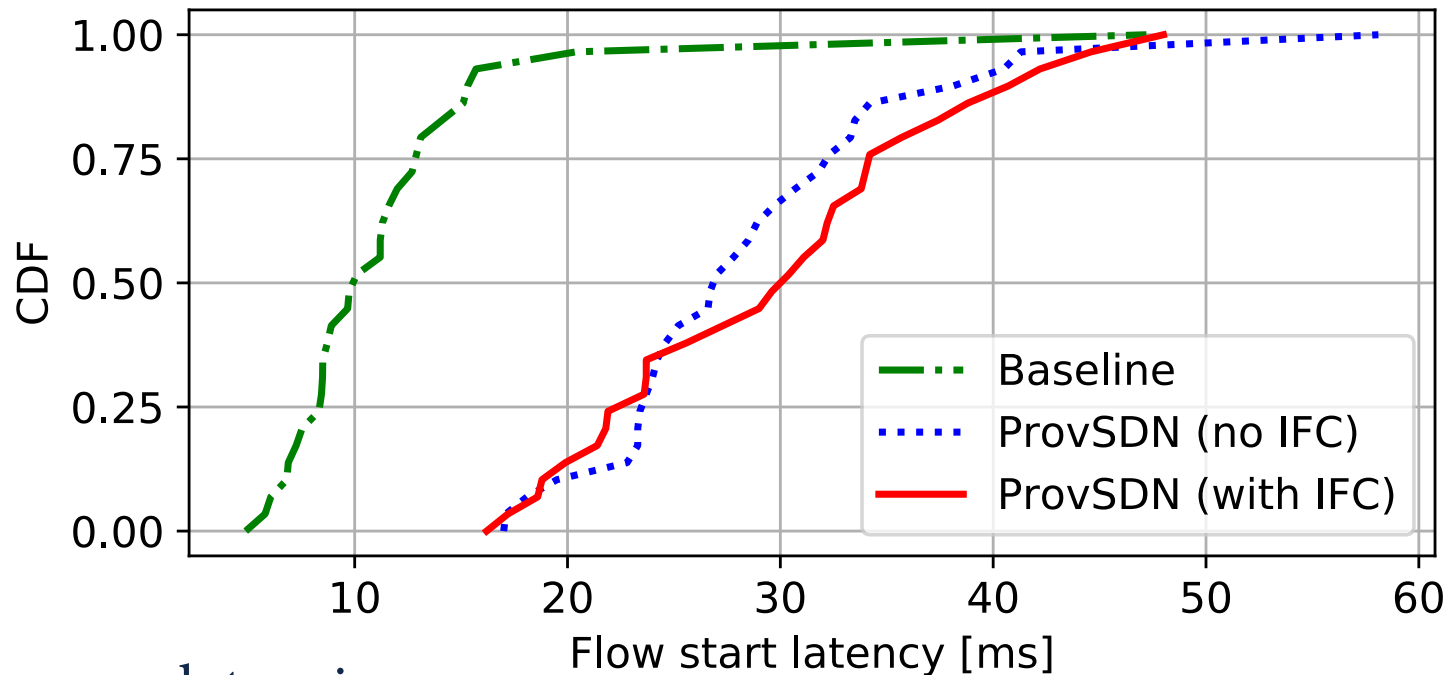


# Attack Evaluation

- Use triggering app `trigger` to modify an incoming packet before being received by forwarding app `fwd`
- Label: `trigger` as **low integrity** and `fwd` as **high integrity**
- Policy: prevent **low** from flowing to **high**



# Performance Evaluation



- Average latencies:
  - Without PROVSDN: **11.66 ms**
  - PROVSDN, no IFC: **28.51 ms**
  - PROVSDN with IFC: **29.53 ms**

Acceptable latency when  
amortized over long flows

# Summary

- We analyzed the **IFC integrity problem** in SDN control planes by investigating information flow
- We proposed a model to identify **cross-app interactions as vectors for potential attacks** and found where they existed in ONOS as a case study
- We proposed a **data provenance** approach with PROVSDN to record control plane state evolution and **enforce IFC** in an online reference monitor
- We **implemented PROVSDN** in the **ONOS controller**



# Questions?

- Thanks for listening!
- **Ben Ujcich**  
E-mail: [ujcich2@illinois.edu](mailto:ujcich2@illinois.edu)  
Web: <http://ujcich2.web.engr.illinois.edu/>

Website



CAP Paper



This material is based upon work supported in part by the National Science Foundation under Grant Nos. **CNS-1657534** and **CNS-1750024**. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

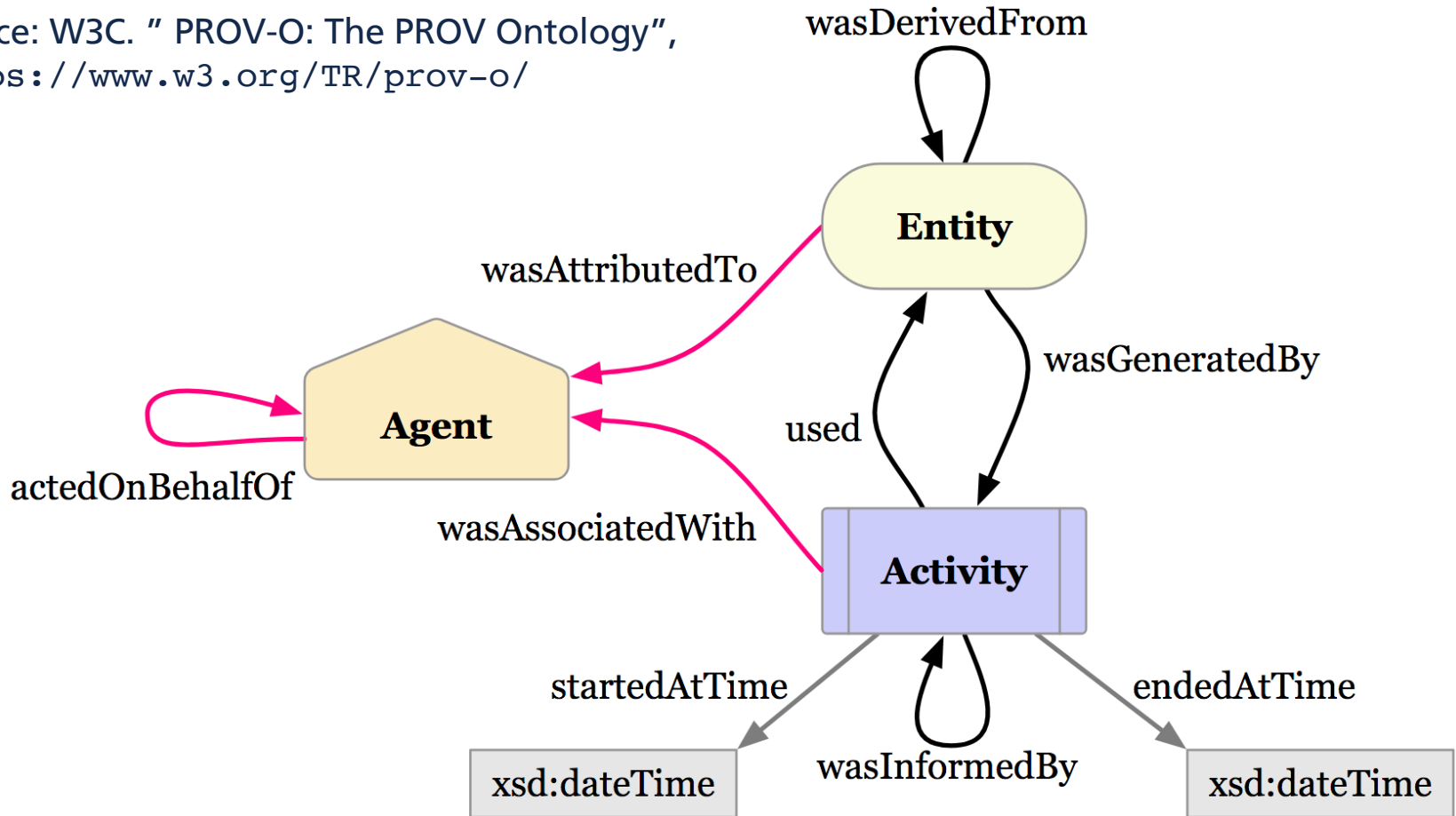
# Backup Slides

# Static Analysis for ONOS

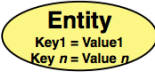
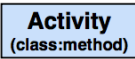

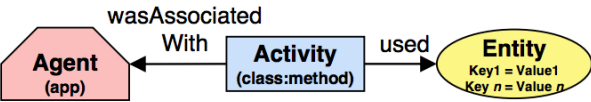
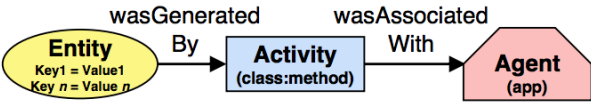
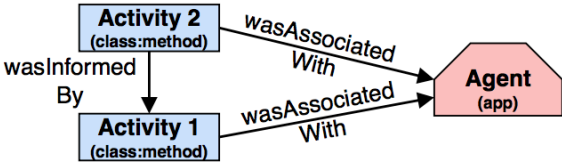
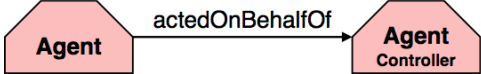
- JavaParser to build abstract syntax tree (AST)
- Sources and sinks derived from analysis of where permissioned methods were called in apps
- Field-sensitive inter-procedural data flow analysis

# W3C PROV Data Model

Source: W3C. " PROV-O: The PROV Ontology",  
<https://www.w3.org/TR/prov-o/>



# W3C PROV Semantics

Object or Event	W3C PROV-DM Representation
Control plane object with attributes	
App method or function call	
App, controller, or switch identity	
App reading object from the shared control plane	
App writing object to the shared control plane	
Intra-app method or callback method	
Internal service on behalf of controller	

# PROVSDN Microbenchmarks

Operation	Average time per operation	Number of operations	Percent of total time
Collect	155.66 $\mu\text{s}$	23 067	1.38%
Write	11.15 $\mu\text{s}$	57 948	0.25%
IFC check	98.50 $\mu\text{s}$	544	0.02%
Internal check	44.67 $\mu\text{s}$	5 692 315	98.34%

# Limitations

- Availability-based attacks → can still audit past actions to influence policy-making process
- Separation of memory enforcement → redesign controllers
- Language-based limitations
  - C/C++ controllers
  - Python controllers
  - Java controllers