Cross-App Poisoning in Software-Defined Networking

Benjamin E. Ujcich\textsuperscript{1}, Samuel Jero\textsuperscript{2}, Anne Edmundson\textsuperscript{3}, Qi Wang\textsuperscript{1}, Richard Skowyra\textsuperscript{2}, James Landry\textsuperscript{2}, Adam Bates\textsuperscript{1}, William H. Sanders\textsuperscript{1}, Cristina Nita-Rotaru\textsuperscript{4}, and Hamed Okhravi\textsuperscript{2}

\textsuperscript{1} ILLINOIS
\textsuperscript{2} LINCOLN LABORATORY
\textsuperscript{3} MASSACHUSETTS INSTITUTE OF TECHNOLOGY
\textsuperscript{4} Northeastern
SDN Overview

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

SDN Overview Diagram:

- SDN Controller
  - Northbound API
    - NB API
    - Core methods
    - Data stores
    - SB API
  - Southbound API
- Forwarding Devices
  - Switches
  - End hosts
- External Apps
  - External app
  - Internal app modules
  - Internal app

SDN centralizes decisions into an SDN controller. The SDN controller acts as a network operating system. Network applications (apps) extend functionality.
IEEE S&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)

Malicious network app ($a_1$)

Host tracking network app ($a_2$)

Host manager
- Host data store

SDN controller (control plane)

Flow rule manager
- Flow rule data store

No permissions to add flow rules directly

RBAC Policy:
- $a_1$: HOST_READ, HOST_WRITE
- $a_2$: HOST_READ, HOST_WRITE, FLOWRULE_READ, FLOWRULE_WRITE
RBAC Limitations

RBAC Policy:
- $a_1$: HOST_READ, HOST_WRITE
- $a_2$: HOST_READ, HOST_WRITE, FLOWRULE_READ, FLOWRULE_WRITE

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 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

```java
public class ReactiveForwarding {
    public void activate(...) {
        appId = coreService.registerApplication("org.onosproject.fwd");
        packetService.addProcessor(processor, PacketProcessor.director(2));
        ...
    }
    private class ReactivePacketProcessor implements PacketProcessor {
        public void process(PacketContext context) {
            ...installRule(context, ...);
        }
    }
    private void installRule(PacketContext context, ...) {
        ForwardingObjective forwardingObjective =
            DefaultForwardingObjective.builder().withSelector(
                selectorBuilder.build()).withTreatment(treatment).
                withPriority(flowPriority).withFlag(
                ForwardingObjective.Flag.VERSATILE).fromApp(appId).
                makeTemporary(flowTimeout).add();
        flowObjectiveService.forward(context.inPacket(),
            receivedFrom().deviceId(), forwardingObjective)
    }
}
```
### Table 1: Static Analysis Results of CAP Gadgets for Security-Mode ONOS Apps.

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

**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 **PROV-SDN** to record control plane state evolution and enforce IFC in an online reference monitor.
- We implemented **PROV-SDN** in the **ONOS controller**.
Questions?

- Thanks for listening!
- Ben Ujcich
  E-mail: 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 Semantics

<table>
<thead>
<tr>
<th>Object or Event</th>
<th>W3C PROV-DM Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control plane object with attributes</td>
<td></td>
</tr>
<tr>
<td>App method or function call</td>
<td></td>
</tr>
<tr>
<td>App, controller, or switch identity</td>
<td></td>
</tr>
<tr>
<td>App reading object from the shared control plane</td>
<td></td>
</tr>
<tr>
<td>App writing object to the shared control plane</td>
<td></td>
</tr>
<tr>
<td>Intra-app method or callback method</td>
<td></td>
</tr>
<tr>
<td>Internal service on behalf of controller</td>
<td></td>
</tr>
</tbody>
</table>
## PROVSDN Microbenchmarks

<table>
<thead>
<tr>
<th>Operation</th>
<th>Average time per operation</th>
<th>Number of operations</th>
<th>Percent of total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>155.66 $\mu$s</td>
<td>23,067</td>
<td>1.38%</td>
</tr>
<tr>
<td>Write</td>
<td>11.15 $\mu$s</td>
<td>57,948</td>
<td>0.25%</td>
</tr>
<tr>
<td>IFC check</td>
<td>98.50 $\mu$s</td>
<td>544</td>
<td>0.02%</td>
</tr>
<tr>
<td>Internal check</td>
<td>44.67 $\mu$s</td>
<td>5,692,315</td>
<td>98.34%</td>
</tr>
</tbody>
</table>
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