Project Members

- Ben Ujcich (UIUC)
- Sam Jero (Purdue)
- Anne Edmundson (Princeton)
- Richard Skowyra (MITLL)
- James Landry (MITLL)
- Adam Bates (UIUC)
- Bill Sanders (UIUC)
- Cristina Nita-Rotaru (Northeastern)
- Hamed Okhravi (MITLL)
Motivation

Security Challenges and Opportunities of Software-Defined Networking

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

Attacks against SDN controllers and the introduction of malicious controller apps are probably the most severe threats to SDN.3,7 Dynamic configurations make it more difficult for defenders to tell whether the current or past configuration is intended and correct. The underlying technology and protocols. In addition, flexibility makes it hard to define meaningful SDN network policies, such as which flows are affected by a specific network application and modified in a specific way. The flexibility SDNs offer, however, defines...
Challenges

- Network applications can modify:
  - OpenFlow control protocol messages (e.g., PACKET_IN)
  - Shared data structures (e.g., topology data store)
- Northbound API boundary between apps and controller is complicated
- Apps bundled with controller have risks depending on language
Prior Solutions

- **Permission-based access control (e.g., Security Mode ONOS)**
  - **Pros**: easy to implement hierarchical permissions
  - **Cons**: does not track data once permission has been granted; not expressive for contextual-based systems

- **Taint tracking**
  - **Pros**: traces how data is used from “sources” to “sinks” for information flow control; minimal additional storage constraints
  - **Cons**: does not capture which system principal/agent was responsible (i.e., no attribution)
Solution: ProvSDN

- Add data provenance collection to controller activities to create a provenance-aware control plane
- Implemented as extension to ONOS SDN controller
- No modifications needed to apps
- Acceptable latency overheads for provenance capture (~100 ms) and online detection/prevention (~300 ms)
Components

- Cross-app poisoning attacks
- Northbound API semantics
- ProvSDN provenance model
- ProvSDN architecture design
- Implementation
- Evaluation
- Results
Cross-App Poisoning Attacks

**Method 1:** Shared data structure access via controller API

**Method 2:** `PacketIn` processing via callbacks.
Northbound API Semantics

- Unlike traditional operating systems, SDNs do not (yet) have well-defined semantics
- Prerequisite for defining provenance model
- Approach: static analysis of controller functions/methods
  - Class with high number of references in other classes (3 or more) is considered public-facing and thus part of the northbound API
  - ONOS “Public”: 63 classes, 721 methods
  - ONOS “Internal”: 194 classes, 1,405 methods
## ProvSDN Provenance Model (W3C PROV-DM)

<table>
<thead>
<tr>
<th>W3C PROV type</th>
<th>SDN objects of interest</th>
<th>Additional attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>● Switches&lt;br&gt;● Hosts&lt;br&gt;● Network Links&lt;br&gt;● Flow Rules&lt;br&gt;● OpenFlow messages</td>
<td>● RUUID&lt;br&gt;● DUUID&lt;br&gt;● Creation time&lt;br&gt;● Class name</td>
</tr>
<tr>
<td>Activity</td>
<td>● OpenFlow message processing&lt;br&gt;● Flow rule management&lt;br&gt;● Host tracking&lt;br&gt;● Link and topology management&lt;br&gt;● Storage management</td>
<td>● UUID&lt;br&gt;● Creation time&lt;br&gt;● Method name&lt;br&gt;● Class name</td>
</tr>
<tr>
<td>Agent</td>
<td>● Apps&lt;br&gt;● Controller&lt;br&gt;● Switches</td>
<td>● UUID&lt;br&gt;● App name</td>
</tr>
</tbody>
</table>

ProvSDN Architecture Design

- Security goals
  - Non-bypassable
  - Complete
- Threat model
- Northbound API enforcement
- Optimizations
Implementation

- Controller: ONOS (Java-based)
  - ProvSDN provenance collector: 1,100 LOC
- Provenance graph database: Neo4j
  - Separate Neo4j server instance
- Provenance query language: Neo4j Cypher
Evaluation

- **Policy:** only allow apps to use data that was
  1. generated from previous activity by app,
  2. generated by controller, or
  3. generated by switches

- Enforcing application isolation

**Neo4j Cypher query for policy**

```cypher
MATCH p=(b:AGENT)<-(a:ACTIVITY),
     q=(a:ACTIVITY)<-[*]- (f:ENTITY)<-[ :USED ]- ()<-[*]- (c: ACTIVITY) -> (d:AGENT),
     r=(c:ACTIVITY)<-[ :WAS_GENERATED_BY ]- (e:ENTITY)
WHERE e.time_create > currentTime() - 2 seconds
  AND b.name <> d.name AND e.name <> f.name
  AND b.name <> "openflow" AND d.name <> "openflow"
  AND b.name <> "controller" AND d.name <> "controller"
RETURN p, q, r LIMIT 1;
```

**Subgraph pattern represented by query**
Evaluation: Host Location Change Attack

- Prevent forwarding app from using HostLocation data that was previously tampered with by malicious app mal
Evaluation: ARP Spoofing Attack

- Prevent forwarding app from using an OpenFlow PacketIn message that was tampered with by malicious app mal.
Results: End Host Latency

- Provenance generation adds one order of magnitude to latency
- Average 140 ms without checks and 330 ms with checks
- (Future work: other graph databases)
Results: Microbenchmarking

- Online querying was most expensive
- API boundary check was most frequent (and least expensive)
Results: Storage

- Spikes correspond to flow modifications; depends on topology
- (Future work: pruning provenance graph)
Summary

- Provenance-based solution to information flow control for securing SDN controllers and network applications
- Real-time checking for online enforcement of information flow control policies
- Implemented in production-quality ONOS SDN controller
- Future work: exploring other ways we can use provenance (e.g., compliance, forensics)
- Paper submitted to NDSS ‘18
Questions?

- Thanks for listening!