Causal Analysis for Software-Defined Networking Attacks

Benjamin E. Ujcich¹, Samuel Jero², Richard Skowyra², Adam Bates³, William H. Sanders⁴, and Hamed Okhravi²

¹ GEORGETOWN UNIVERSITY  ² LINCOLN LABORATORY
3 ILLINOIS  ⁴ Carnegie Mellon University

30th USENIX Security Symposium
August 11–13, 2021
Virtual Event
SDN: A Primer

Decoupling of traffic decision-making from traffic being forwarded

[Diagram showing SDN architecture with control plane and data plane]

- Control Plane
- Switches
- Data Plane Hosts
- Data Plane
SDN: A Primer

Logically centralized control plane that determines behavior

CONTROL PLANE

SB API protocol (e.g., OpenFlow)

DATA PLANE

SDN Controller

Southbound API

Logically centralized (but perhaps physically distributed)
SDN: A Primer

Network services API for extensible network applications

Data Plane

CONTROL PLANE

APPLICATION PLANE

SDN Controller

Southbound API

Firewall App

QoS App

Routing App

Northbound API

SB API protocol (e.g., OpenFlow)

Switches

Data Plane Hosts

Application plane

Logically centralized (but perhaps physically distributed)
SDN: A Security Target

Cross-plane vulnerabilities
(Ujcich et al., NDSS ‘20)
(Xiao et al., S&P ‘20)

Cross-app poisoning
(Ujcich et al., CCS ‘18)
Help, My SDN Has Been Attacked!

What events happened in my network?
How do I know I have complete oversight?
Can I accurately understand the attack?
What are the root causes of the attack?
What else did the attack affect?
Data Provenance to the Rescue 😊

- Shows how data were generated and used
- Captures system principals, processes, and data objects in DAG
- Useful for attack investigation and root cause analysis
What Makes This Challenging for SDN?

Challenge 1: Dependency explosion

Challenge 2: Incomplete dependencies

Challenge 3: Attribution and responsibility

Challenge 4: Interpretation

PicoSDN
Provenance-Informed Causal Observation for Software-Defined Networking
PicoSDN Challenges and Solutions

Challenge 1: Dependency explosion
Discovery: Long-running apps produce false data and process dependencies

- Challenge: Long-running apps produce false data and process dependencies.
- Diagram illustrating the dependency explosion with packets and flow rules:
  - Packet 1 wasGeneratedBy Flow rule 1
  - Packet 2 wasGeneratedBy Flow rule 1
  - Packet 2 used Flow rule 2
  - Packet 3 used Flow rule 2
  - Packet 4 used Flow rule 2
  - Flow rule 1 used
  - Flow rule 2 used
  - Flow rule 3 used
  - Flow rule 4 used

Diagram shows the complex interdependencies created by long-running SDN applications, leading to false data and dependencies.
Solution: Mitigate with events as short processes (*execution partitioning*) and control plane objects as data (*data partitioning*)
Challenge 2: Incomplete dependencies
Discovery: Control plane can trigger other control plane activities via the data plane
PicoSDN Challenges and Solutions

Solution: Mitigate by modeling the data plane

Data plane model based on:
- happens-before relations, packet timestamps (within threshold), match fields, and network topology
Challenge 3: Attribution and responsibility
Discovery: Default flow rules create a data dependency explosion

Default Flow Rule (flow ID = 1)
Match: all traffic
Action: send to controller

wasDerived From
Packet In
Time: 1
Flow ID: 1

Packet In
Time: 2
Flow ID: 1

Packet In
Time: 10
Flow ID: 1

Packet In
Time: 100
Flow ID: 1

High fan-out potential
Solution: Mitigate by assigning agency to a switch port
Challenge 3: Attribution and responsibility

Discovery: Host identifiers can be easily spoofed

Victim sends packets with its own network identifier (MAC address)

Attacker spoofs victim’s network identifier to trick SDN controller into redirecting victim traffic to the attacker
**PicoSDN Challenges and Solutions**

Solution: Track how hosts’ identifiers change over time

- **Host X**
  - MAC address: X
  - Location: Switch 1, Port 1
  - \texttt{wasRevisionOf} Host X
  - MAC address: X
  - Location: Switch 1, Port 2
  - \texttt{wasRevisionOf} Host X

- Host Provider event listener
- Packet In
- Switch 1 Port 1
- Switch 1 Port 2

**Host identifier evolution**
Challenge 4: Interpretation

Discovery: Provenance is not useful unless we can understand it
PicoSDN Challenges and Solutions

Solution: Provide practical tools to summarize, analyze, and trace network activities

1. **Common ancestry:** Given several nodes, what nodes are the common ancestors?
2. **Backward-forward:** Given a path between evidence (node) and root, what does the ancestry look like at each stage?
3. **Activity summary:** How do data plane packets impact flow rules?
4. **Identifier evolution:** How do hosts change identity?
PicoSDN Architecture

**SDN Controller**
- **APPLICATION PLANE**
  - Event Listeners & Packet Processors
  - App 1
  - App n
- **CONTROL PLANE**
- **DATA PLANE**
- **ONLINE OPERATION**
  - Controller core
    - NB API
      - Hooked Methods
      - Data Store
    - SB API
      - Hooked Methods
  - Provenance Collector
  - Provenance Serializer
  - Internal State
  - Hooked Methods
- **OFFLINE OPERATION**
  - **PicoSDN INVESTIGATION PHASE**
    - Ingester
    - Cleaner
    - Topology Augmenter
      - Tracer
        - Common Ancestry
        - Backward-Forward
        - Activity Summary
        - Identifier Evolution
  - **PicoSDN RUNTIME PHASE**
    - Southbound API
      - Data Plane Hosts
      - Forwarding Devices
  - **Data Plane**
- **PicoSDN Inputs and Outputs**
  - (from/to network administrator)
PicoSDN Security Evaluation

Example: Cross-Plane Event-Based Vulnerabilities

1. Switch ports as agents
2. Host identifier evolution (i.e., spoofing)
3. Data plane model based on reactive forwarding

PM = packet manager
HP = host provider
f = flow rule, p = packet
h = host, s = switch
fwd = forwarding app
acl = access control app

1. Cross-Plane Event-Based Vulnerabilities
2. P1
3. P2
4. P3
5. P4
6. S1
7. S2
8. f1
9. f2
10. f3
11. h1(v1)
12. h1(v2)
13. acl
14. PM
15. HP
16. fwd
17. Agency
18. Data Plane Model
19. Identifier Evolution
Conclusions

- Considered causal analysis challenges in SDN attacks
- Design takeaways
  - Dependency explosion mitigated by control plane control plane objects (data) and events (execution)
  - Incomplete dependencies mitigated by data plane model
  - Attribution and responsibility are challenging
- Designed PicoSDN and implemented on ONOS SDN controller
Thanks!

Thank you for your time!

Benjamin E. Ujcich
E-mail: bu31@georgetown.edu
Web: https://benujcich.georgetown.domains/

This work was supported in part by NSF Grant No. 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.