Sphinx: Detecting Security Attacks in Software-Defined Networks

Mohan Dhawan    Rishabh Poddar    Kshiteej Mahajan    Vijay Mann

IBM Research, India
Software-Defined Network (SDN)

Logically-centralized control

SDN Controller

Smart, slow

Dumb, fast switches

Data plane
Software-Defined Network (SDN)

Logically-centralized control

Smart, slow

SDN Controller

Control plane

Dumb, fast

switches
Software-Defined Network (SDN)

Logically-centralized control

Smart, slow

SDN Controller

Dumb, fast switches

OpenFlow
Software-Defined Network (SDN)
Software-Defined Network (SDN)
Software-Defined Network (SDN)
Software-Defined Network (SDN)
Software-Defined Network (SDN)
Software-Defined Network (SDN)

Correct functioning requires preservation of:

- Network topology
- Data plane forwarding
Outline

- SDN Overview
- **Motivation**
- Sphinx
- Implementation
- Evaluation
- Conclusion
Vulnerable SDNs

- OpenFlow operational semantics
  - All unmatched packets are forwarded to the controller
Vulnerable SDNs

- OpenFlow operational semantics
  - All unmatched packets are forwarded to the controller
- Attacks afflicting traditional networks affect SDNs too
  - Traditional defenses do not work in SDNs
Vulnerable SDNs

- OpenFlow operational semantics
  - All unmatched packets are forwarded to the controller
- Attacks afflicting traditional networks affect SDNs too
  - Traditional defenses do not work in SDNs
- Attacks possible from compromised switches and end hosts
  - Soft switches on end host servers attractive targets for attackers
Several Attacks Possible

- Network topology
  - Corrupt routing table (ARP)
  - Fake topology (LLDP)
  - Multicast (IGMP)

- Data plane forwarding
  - Switch TCAM exhaustion
  - Switch blackhole
Controller Vulnerability

- Security analysis of four popular available SDN controllers

<table>
<thead>
<tr>
<th>Attack</th>
<th>OpenDaylight</th>
<th>Floodlight</th>
<th>POX</th>
<th>Maestro</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARP poisoning</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Fake topology</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Controller DoS</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Network DoS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TCAM exhaustion</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Switch blackhole</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Fake Network Topology Attack

SDN Controller

LLDP

A

B

C

D
Fake Network Topology Attack
Fake Network Topology Attack

SDN Controller

A
B
C
D

LLDP

D
D
Fake Network Topology Attack
Fake Network Topology Attack

SDN Controller

LLDP

PACKET_IN

A

B

C

D
Fake Network Topology Attack
Fake Network Topology Attack

SDN Controller

PACKET_IN

LLDP

AD

A

B

C

D
Fake Network Topology Attack
Fake Network Topology Attack

Video demo: [http://goo.gl/zRG8bz](http://goo.gl/zRG8bz)
Outline

- SDN Overview
- Motivation
- **Sphinx**
- Implementation
- Evaluation
- Conclusion
Detecting Security Threats in Real Time

- Verify network actions using OpenFlow metadata
  - All controller communication mediated by a shim
  - Learn network behaviour and automatically generate network invariants
Key Idea: FlowGraphs

Exploit predictability and pattern in topological and data plane forwarding to detect violation
Key Idea: FlowGraphs

Exploit predictability and pattern in topological and data plane forwarding to detect violation
Workflow (I)

- Intercept relevant OpenFlow messages to extract topological and forwarding metadata

![Diagram of workflow](image-url)
Workflow (I)

- Intercept relevant OpenFlow messages to extract topological and forwarding metadata

Assumption: Honest majority of switches along flow path
Workflow (II)

- Generate flowgraph constraints from the extracted metadata

1. Intercept OpenFlow packets
2. Build Flow Graph
3. Validate network behavior
Accurate Characterization of Flows

- Maintain mapping of entities and allowed metadata
  - Hosts (Src MAC/IP/port, Dst MAC/IP/port)
  - Switches (Switch and in/out-port)
  - Flows (Flow match and statistics)

- Incrementally augment the flowgraph with such constraints
Workflow (III)

- Use custom algorithms to detect constraint violations on flowgraphs

1. Intercept OpenFlow packets
2. Build Flow Graph
3. Validate network behavior
# Administrator Policies

- Specified in constraint language

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>((\text{SrcID, DstID}), \text{where } \forall \text{SrcID and DstID } \in {\text{Controller}</td>
</tr>
<tr>
<td>Object</td>
<td>{\text{Counters}</td>
</tr>
<tr>
<td>Operation</td>
<td>\text{In}</td>
</tr>
<tr>
<td>Trigger</td>
<td>\text{Packet_in}</td>
</tr>
</tbody>
</table>
Administrator Policies

• Specified in constraint language

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>(SrcID, DstID), where ∀ SrcID and DstID ∈ {controller</td>
</tr>
<tr>
<td>Object</td>
<td>{Counters</td>
</tr>
<tr>
<td>Operation</td>
<td>In</td>
</tr>
<tr>
<td>Trigger</td>
<td>Packet_in</td>
</tr>
</tbody>
</table>

• Example policy to check if all flows from host H3 pass through specified waypoints S2 and S3

```xml
<Policy PolicyId="Waypoints">
  <Subjects><Subject value="H3, *" /></Subjects>
  <Objects>
    <Object><Waypoint value="S2" /></Object>
    <Object><Waypoint value="S3" /></Object>
  </Objects>
  <Operation value="IN" />
  <Trigger value="Periodic" />
</Policy>
```
Constraint Validation

- Topological state
  - Packet spoofing, controller DoS
  - Fast and deterministic
Constraint Validation

• Topological state
  – Packet spoofing, controller DoS
  – Fast and deterministic

• Forwarding state
  – Flow graph consistency, switch DoS, flow statistics
  – Both deterministic and probabilistic
  – Similarity Index (SI) categorizes nature of flow using statistics observed at switches along the flow path
    • Identify malicious switches along flow path
Outline

- SDN Overview
- Motivation
- Sphinx
- **Implementation**
- Evaluation
- Conclusion
Implementation

- Controller-agnostic proxy between the controller and the switches
  - Prototype compatible with OpenFlow (v1.1.0)
  - Works with OpenDaylight (v0.1.0) and Floodlight (v.0.90)
  - Written in ~2100 Java LOC
  - Uses the fast Netty I/O framework with separate queues for communication in either direction
Outline

- SDN Overview
- Motivation
- Sphinx
- Implementation
- **Evaluation**
- Conclusion
Experimental Setup

- Physical setup of three tiered datacenter topology with 14 switches
- Emulated Mininet network of up to 10K hosts
- Measure
  - Accuracy of deterministic and probabilistic verification
  - Performance impact on end user latency, throughput and policy verification
Accuracy (I)

- Attack detection times under different settings

<table>
<thead>
<tr>
<th>Attack</th>
<th>Detection time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Physical testbed</td>
</tr>
<tr>
<td>ARP poisoning</td>
<td>44</td>
</tr>
<tr>
<td>Fake topology</td>
<td>66</td>
</tr>
<tr>
<td>Controller DoS</td>
<td>75</td>
</tr>
<tr>
<td>Network DoS</td>
<td>75</td>
</tr>
<tr>
<td>TCAM exhaustion</td>
<td>n/a</td>
</tr>
<tr>
<td>Switch blackhole</td>
<td>75</td>
</tr>
</tbody>
</table>

- Measure false alarms generated in three diverse benign traffic traces (14min, 65min and 2hr)
  - Execution raised no alarms
Accuracy (II)

- Probabilistic verification – probability of false alarms and lack of genuine alarms at different margins of similarity ($\tau$)
  - $\tau = x$ implies that SI observed at each switch in the flow path must lie between SI$/x$ and SI*$x$
  - $\tau = 1$ implies that all switches along the flow path must report the same flow statistics
  - $\tau = 1.045$ corresponds to link loss rate of 1%
Accuracy (II)

- Probabilistic verification – probability of false alarms and lack of genuine alarms at different margins of similarity ($\tau$)
Performance (I)

- End user latency

Only 300µs at 50% for 1K hosts
Performance (II)

• Throughput

![Graph showing throughput vs. TCAM misses/sec with and without Sphinx, indicating just 2% overhead.](image)
Performance (III)

- Policy verification

Only 869μs for 10K policies
Outline

- SDN Overview
- Motivation
- Sphinx
- Implementation
- Evaluation
- Conclusion
Conclusion

• Existing controllers are vulnerable to a wide array of attacks
• Sphinx is a controller agnostic tool that detects security threats originating within SDNs in real time
• Sphinx builds succinct metadata for each network flow and uses both deterministic and probabilistic checks to identify deviant behavior
• Our evaluation shows that Sphinx is practical and imposes minimal overheads
Thank You.

Contact: mohan.dhawan@in.ibm.com