Preacher: Network Policy Checker for Adversarial Environments

Kashyap Thimmaraju, Liron Schiff and Stefan Schmid
Backdoors and exploits

• Network devices are very effective attack vectors
  • Provide access to internal networks
  • Transparent to many security measures
  • Hard to detect

• Mostly used by state actors
  • Exploiting 0-day vulnerabilities
  • Compromising supply chains
Attack model

- A compromised network device can run arbitrary malicious code.
  - Modify traffic
    - To attack network hosts (including DoS)
  - Report false configuration and state
    - To evade detection

- Two attack building blocks:
  1) **Drop**: An adversary prevents a packet from being sent (from one or more ports).
  2) **Injection**: An adversary fabricates and sends a new packet or resends a packet sent earlier. This also includes sending a packet from an unintended port.
Attack model (cont.)

• Attack examples:
  a) Denial of service
  b) Port-scan
  c) Mirroring
  d) MitM
  e) Covert channel
  f) Re-route
Naïve solution: Trajectory Sampling (TS)

- Sample packets
  - Global set of hash values - **Attacker avoids them**
- Send samples to verifier
  - Attacker corrupt them on the way
- Compare trajectories to policy

- Good for traffic monitoring, but not suited adversarial settings
Split Assignment Trajectory Sampling (SATS) [Lee&Kim DSN06]

• Sample packets
  • Independent sets of hash values
• Send samples to verifier
  • Switch should use encryption
• Compare trajectories to policy

• Designed for adversarial settings

• But…
SATS Limitations

• Sample packets
  • Security guarantees?
  • Fixed-hash-crafted injection!
  • Switch compatibility

• Control plane security
  • Messages (samples and assignments)
  • Endpoints (verifier etc.)

• Compare trajectories to policy
  • Obtain policy (network compatibility)?
  • Scalability?
Preacher

• An improved trajectory sampling solution
• Harnesses programmable network technologies
• Uses robust and distributed design
• Includes a security analysis and a prototype
• Addresses all SATS limitations
Contributions

• Sample packets
  • Security guarantees
  • Fixed-hash-crafted injection
  • Switch compatibility

✓ Analysis + evaluations
✓ Dynamic assignment
✓ SDN switch

• Control plane security
  • Messages (samples and assignments)
  • Endpoints (verifier etc.)

✓ OpenFlow encryption
✓ Distributed design

• Compare trajectories to policy
  • Obtain policy (network compatibility)
  • Scalability

✓ SDN controller
✓ Parallel design
Preacher Scheme

- Cooperates with controller and routing apps
  - Sends hash assignments (switch configuration)
  - Receives samples (e.g., PacketIns)
  - Obtains a policy
- Verifies samples
  - For each sample computes other expected samples (using the policy)
  - Detects inconsistencies (with timeouts)
Preacher Scheme – Distributed and Parallel

Use redundancy to improve security and fault tolerance!

Routing app. (policy) → Preacher

Hash assignment → Verification

Controller

Topology → Incoming Samples → Switch config.

Internet

Network nodes...
Preacher Scheme – Distributed and Parallel

Use redundancy to improve security and fault tolerance!

• Hash Assignment
  • Each assigner configures a subset of switches (or pairs)
  • Compromise or malfunction of one assigner is not fatal

• Verification
  • Each verifier is responsible for a subset of hashes, and receives a subset of the samples.
  • Better performance and security (depending on subset overlaps)
Security Analysis

• An attack occurs along a directed path
  • Where the packet should have traversed

• Detection requirement
  • Attacked packet hash is assigned before and after attack
  • Same for drop and inject

• Hash assignments
  • Each switch is assigned with $p$ of hash space. $p$ is very small ($p \ll \frac{1}{n}$).
  • Independent vs. pairs assignment
Security Analysis

• Detection probability
  • For independent assignment:
    \[ P_{ia} = (1 - (1 - p)^{k_1}) \cdot (1 - (1 - p)^{k_2}) \approx p^2 k_1 k_2 \]
  • For pairs assignment:
    \[ P_{pa} > 1 - \left( 1 - \frac{p}{n-1} \right)^{k_1 k_2} \approx \frac{p k_1 k_2}{n - 1} \]
• We assume \#packets-till-detection follows geometric distribution.
• We use common packet rates to get expected detection time.
• We use common data center link capacities to derive expected total samples’ rate (pps).
Evaluation

- Prototype based on ONOS-1.4 with OpenFlow 1.3 as controller.
  - Used services: Flow objective, Flow rule, Device, Packet-in
- Clos topology with k=4
  - Open vSwitch (OvS) for switches

- Experiments goals:
  - Verifying analysis
  - Evaluating overheads
    - Switch
    - Controller
  - Evaluating throughput

1 core ≈ 1000 pps
Detection Time vs. Resources

• With pairs-assignment
  • Attacks in small network can easily be detected within minutes
  • In big networks ~10 servers (~100 cores) are needed.

• With simple independent assignment
  • Even in small networks it is very hard to detect.
  • In big networks it is infeasible.
Future work

• Implementation with more programmable network devices
  • hardware switches, P4 switches and smart NICs

• Experimenting at SDN datacenters
Summary

• Preacher harnesses programmable network technologies
• Uses distributed design to ensure robustness and security
• Provides provable security
• Open source prototype is available at:
  www.github.com/securedataplane/preacher