Secure Network Provenance


*University of Pennsylvania    +Georgetown University

http://snp.cis.upenn.edu/
Motivation

An example scenario: network routing

- System administrator observes strange behavior
- Example: the route to foo.com has suddenly changed
- *What exactly happened (innocent reason or malicious attack)?*
We Need Secure Forensics

- **For network routing ...**
  - Example: incident in March 2010
    - Traffic from Capitol Hill got redirected

- **... but also for other application scenarios**
  - Distributed hash table: Eclipse attack
  - Cloud computing: misbehaving machines
  - Online multi-player gaming: cheating

- **Goal: secure forensics in adversarial scenarios**
Q: Explain why the route to foo.com changed to r2.

A: Because someone accessed Router D and changed the configuration from X to Y.

- Not realistic: adversary can tell lies
Challenge: Adversaries Can Lie

Q: Explain why the route to foo.com changed to r2.

Problem: adversary can ...

- ... fabricate plausible (yet incorrect) response
- ... point accusation towards innocent nodes
Existing Solutions

- **Existing systems assume trusted components**
  - Trusted OS kernel, monitor, or hardware
    - E.g. Backtracker [OSDI 06], PASS [USENIX ATC 06], ReVirt [OSDI 02], A2M [SOSP 07]
  - These components may have bugs or be compromised
  - *Are there alternatives that do not require such trust?*

- **Our solution:**
  - We assume no trusted components;
  - Adversary has **full control over an arbitrary subset** of the network (Byzantine faults).
Ideal Guarantees

- Ideally: explanation is always complete and accurate
- Fundamental limitations
  - E.g. Faulty nodes secretly exchange messages
  - E.g. Faulty nodes communicate outside the system
- *What guarantees can we provide?*
Realistic Guarantees

- **No faults:** Explanation is complete and accurate
- **Byzantine fault:** Explanation identifies at least one faulty node
- **Formal definitions and proofs in the paper**

![Diagram](image)

Q: Why did my route to foo.com change to r2?

A: Because someone accessed Router D and changed its router configuration from X to Y.

Aha, at least I know which node is compromised.
Outline

- **Goal:** A secure forensics system that works in an adversarial environment
  - Explains unexpected behavior
  - No faults: explanation is complete and accurate
  - Byzantine fault: exposes at least one faulty node with evidence

- **Model:** Secure Network Provenance
- Tamper-evident Maintenance and Processing
- Evaluation
- Conclusion
Provenance as Explanations

- **Origin:** data provenance in databases
  - Explains the derivation of tuples (ExSPAN [SIGMOD 10])
  - Captures the dependencies between tuples as a graph
  - “Explanation” of a tuple is a tree rooted at the tuple
Provenance as Explanations

- **Origin: data provenance in databases**
  - Explains the derivation of tuples (ExSPAN [SIGMOD 10])
  - Captures the dependencies between tuples as a graph
  - “Explanation” of a tuple is a tree rooted at the tuple

Diagram:
- Nodes represent tuples:
  - route(A, foo.com)
  - link(A, B)
  - route(B, foo.com)
  - link(B, C)
  - route(D, foo.com)
  - link(D, E)
  - route(E, foo.com)
  - link(E, B)
  - route(C, foo.com)

- Edges represent dependencies:
  - route(A, foo.com) → link(A, B) → route(B, foo.com) → link(B, C) → route(C, foo.com)
  - route(D, foo.com) → link(D, E) → route(E, foo.com)

- Diagram shows connections between tuples.
Secure Network Provenance

- **Challenge #1. Handle past and transient behavior**
  - Traditional data provenance targets **current, stable state**
  - What if the system never converges?
  - What if the state no longer exists?
Secure Network Provenance

**Challenge #1. Handle past and transient behavior**

- Traditional data provenance targets current, stable state
- What if the system never converges?
- What if the state no longer exists?
- **Solution: Add a temporal dimension**
Secure Network Provenance

- Challenge #2. Explain changes, not just state
  - Traditional data provenance targets system state
  - Often more useful to ask why a tuple (dis)appeared
  - Solution: Include “deltas” in provenance
Secure Network Provenance

- **Challenge #3. Partition and secure provenance**
  - A trusted node would be ideal, but we don’t have one
  - Need to partition the graph among the nodes themselves
  - Prevent nodes from altering the graph
Partitioning the Provenance Graph

- Step 1: Each node keeps vertices about local actions
  - Split cross-node communications
Partitioning the Provenance Graph

- **Step 1:** Each node keeps vertices about local actions
  - Split cross-node communications
- **Step 2:** Make the graph tamper-evident
Securing Cross-Node Edges

- **Step 1:** Each node keeps vertices about local actions
  - Split cross-node communications
- **Step 2:** Make the graph tamper-evident
  - Secure cross-node edges (evidence of omissions)
Outline

- **Goal:** A secure forensics system that works in an adversarial environment
  - Explains unexpected behavior
  - No faults: explanation is complete and accurate
  - Byzantine fault: exposes at least one faulty node with evidence

- **Model:** Secure Network Provenance
- *Tamper-evident Maintenance and Processing*
- Evaluation
- Conclusion
System Overview

- **Stand-alone provenance system**
- **On-demand provenance reconstruction**
  - Provenance graph can be huge (with temporal dimension)
  - Rebuild only the parts needed to answer a query
Extracting Dependencies

- **Option 1: Inferred provenance**
  - Declarative specifications explicitly capture provenance
  - E.g. Declarative networking, SQL queries, etc.

- **Option 2: Reported provenance**
  - Modified source code reports provenance

- **Option 3: External specification**
  - Defined on observed I/Os of a black-box system
Secure Provenance Maintenance

- Maintain sufficient information for reconstruction
  - I/O and non-deterministic events are sufficient
  - Logs are maintained using tamper-evident logging
  - Based on ideas from PeerReview [SOSP 07]
Secure Provenance Querying

- Recursively construct the provenance graph
  - Retrieve secure logs from remote nodes
  - Check for tampering, omission, and equivocation
  - Replay the log to regenerate the provenance graph

Explain the route from A to foo.com.
Secure Provenance Querying

- Recursively construct the provenance graph
  - Retrieve secure logs from remote nodes
  - Check for tampering, omission, and equivocation
  - Replay the log to regenerate the provenance graph
Secure Provenance Querying

- Recursively construct the provenance graph
  - Retrieve secure logs from remote nodes
  - Check for tampering, omission, and equivocation
  - Replay the log to regenerate the provenance graph

OK. Now I know how the route was derived.
Goal: A secure forensics system that works in an adversarial environment
- Explains unexpected behavior
- No faults: explanation is complete and accurate
- Byzantine fault: exposes at least one faulty node with evidence

Model: Secure Network Provenance
Tamper-evident Maintenance and Processing
Evaluation
Conclusion
Evaluation Results

- Prototype implementation (SNooPy)
  - How useful is SNP? Is it applicable to different systems?
  - How expensive is SNP at runtime?
    - Traffic overhead, storage cost, additional CPU overhead?
    - Does SNP affect scalability?
  - What is the querying performance?
    - Per-query traffic overhead?
    - Turnaround time for each query?
Usability: Applications

- **We evaluated SNooPy with**
  - Quagga BGP: RouteView (external specification)
    - Explains oscillation caused by router misconfiguration
  - Hadoop MapReduce: (reported provenance)
    - Detects a tampered Mapper that returns inaccurate results
  - Declarative Chord DHT: (inferred provenance)
    - Detects an Eclipse attacker that always returns its own ID

- **SNooPy solves problems reported in existing work**
Runtime Overhead: Storage

Manageable storage overhead

- One week of data: E.g. Quagga – 7.3GB; Chord – 665MB

Over 50% of the overhead was due to signatures and acks. Batching messages would help.
Query Latency

- Query latency varies from application to application
- Reasonable overhead

Reasonable overhead

dominated by verifying logs and snapshots

largely due to replaying logs

Query
Latency
Summary

- **Secure network provenance in untrusted environments**
  - Requires no trusted components
  - Strong guarantees even in the presence of Byzantine faults
    - Formal proof in a technical report
  - Significantly extends traditional provenance model
    - Past and transient state, provenance of change, ...
  - Efficient storage: reconstructs provenance graph on demand
  - Application-independent (Quagga, Hadoop, and Chord)

- **Questions?**

_Project website: [http://snp.cis.upenn.edu/](http://snp.cis.upenn.edu/)_