Secure Forensics without Trusted Components

Wenchao Zhou*, Qiong Fei*, Arjun Narayan*, Andreas Haeberlen*, Boon Thau Loo*, Micah Sherr+

**University of Pennsylvania, +Georgetown University*

Introduction

Goal: Develop capability to answer diagnostic or forensics questions about network state

- Systems are found to be in an unexpected state
- Determine the *causes* – why did the route to foo.com change?
- Determine the *effects* – what other routes have been affected?

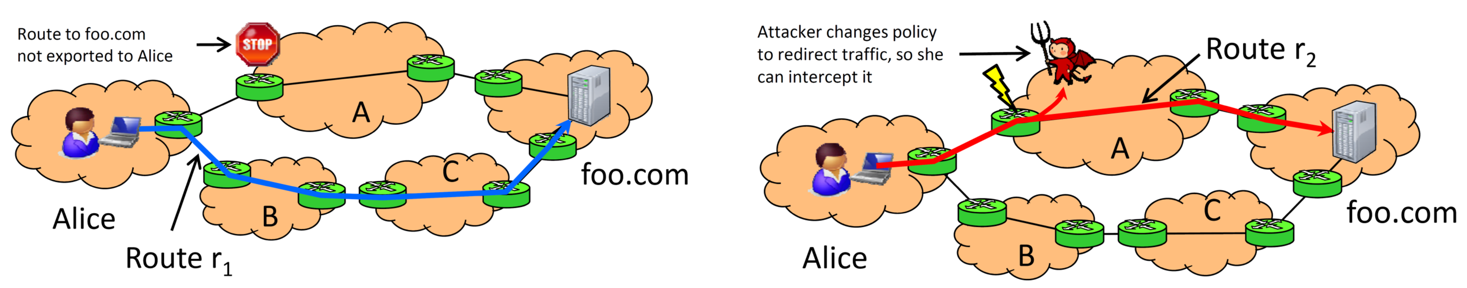


Figure 1. In the system (left), network A's policy blocks traffic to Alice, and Alice reaches foo.com through B & C. If Eve compromises A (right), she can change the policy and eavesdrop Alice's traffic.

Getting correct forensics answers is difficult

- Nodes may be compromised by the attacker
  - Fabricate plausible (yet incorrect) response
  - Misdirect accusation to innocent nodes
- Existing work relies on trusted components, e.g., OS kernel, virtual machine, monitor, hardware, etc

Tamper-evident provenance (TEP), a forensics system that can operate in a completely untrusted environment

- A novel data structure for forensics in adversarial environments
- Tamper-evident forensics query engine
- Prototype implementation and case studies on various systems

Threat Model and Guarantees

Byzantine adversaries

- May have compromised an arbitrary subset of the nodes
- May have complete control over the nodes – arbitrary behavior
- May collude with each other

Guarantees

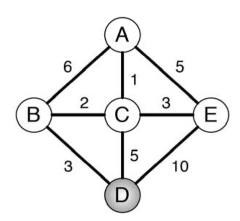
- Idealism: Always get correct forensics results (not possible!)
- Practicality: The conservative model requires compromises
  - \- TEP can only answer queries about observable network state
  - \- Responses may be incomplete, though the missing parts are always clearly identifiable
  - \+ An observable symptom of an attack can **ALWAYS** be traced to a specific misbehavior by at least one incorrect node
  - \+ Forensics results are supported by **VERIFIABLE** evidence

Provenance for Forensics

System representation: tuple and derivation rules

- System state as tuples: E.g. link(@C,D,5), bestCost(@C,D,5)
- System's algorithms as derivation rules:
  E.g. cost(@X,Z,Y,C1+C2) ← link(@X,Y,C1) ∧ bestCost(@Y,Z,C2).

Network provenance [Zhou et al. SIGMOD 2010]



- A DAG representing dependencies between state
- Explains the existence of system state

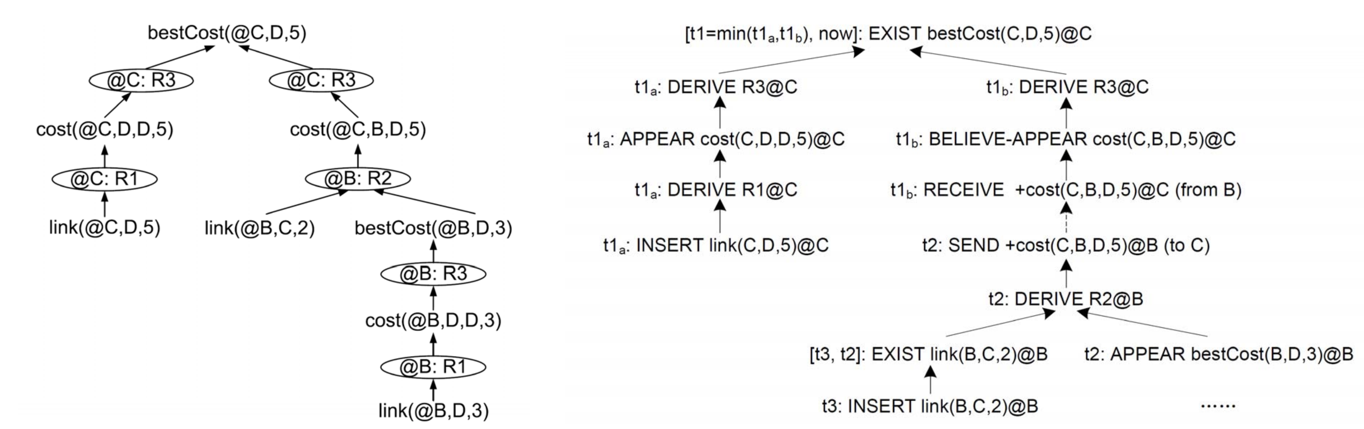


Figure 2. Example provenance graph for the bestCost(@C,D,5) tuple in the classic provenance notation (left) and the extended TEP provenance notation (right).

Strawman solution: provenance + fault detection

- Query results are not guaranteed to be correct (detection delay)
- The information on other (benign) nodes may be corrupted
  - System becomes useless when it is most needed!

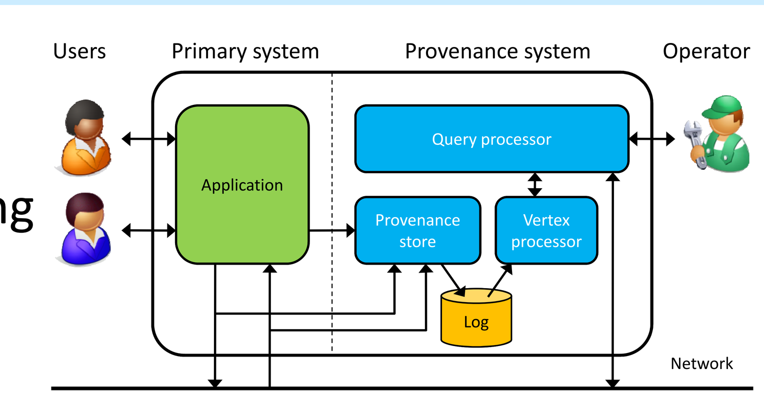
Extended TEP provenance graph (Figure 2 - right)

- An additional temporal dimension – system state in the past
- Explanation of state changes – sometimes more important
- Clean partition of the provenance graph – binding nodes' commitments to each of the partitions

Tamper-evident Query Engine

Architectural overview

- Logging at execution time
- On-demand replay for querying



Provenance store

- Use tamper-evident logging
- Record minimal system state for deterministic replay

Vertex processor

- Fetch the logs and perform deterministic replay
- Generate immediate successors and predecessors

Query processor

- Recursively expand the provenance graph
- Use vertex processor to assemble answers to higher-level queries

Implementation and Case Studies

Three techniques to extract provenance

- *M1 - Inferred Provenance:* Dependencies are explicitly captured in the implementation (e.g. via the use of declarative language)
- *M2 - Reported Provenance:* Modified code reports provenance
- *M3 - External Specification:* Dependencies are defined between observed input and output of black-box applications

Use cases

- *Chord DHT (M1):* explain finger entries / lookup results
- *Hadoop MapReduce (M2):* explain suspicious WordCount results
- *Quagga BGP (M3):* explain routing entry changes / oscillations

Evaluation – secure forensics with reasonable overheads

- *Runtime overhead*: fixed-size overhead for each message
- *Storage overhead:* easily fit into commodity hard disks
- *Query overhead*: up to 70 seconds for provenance querying

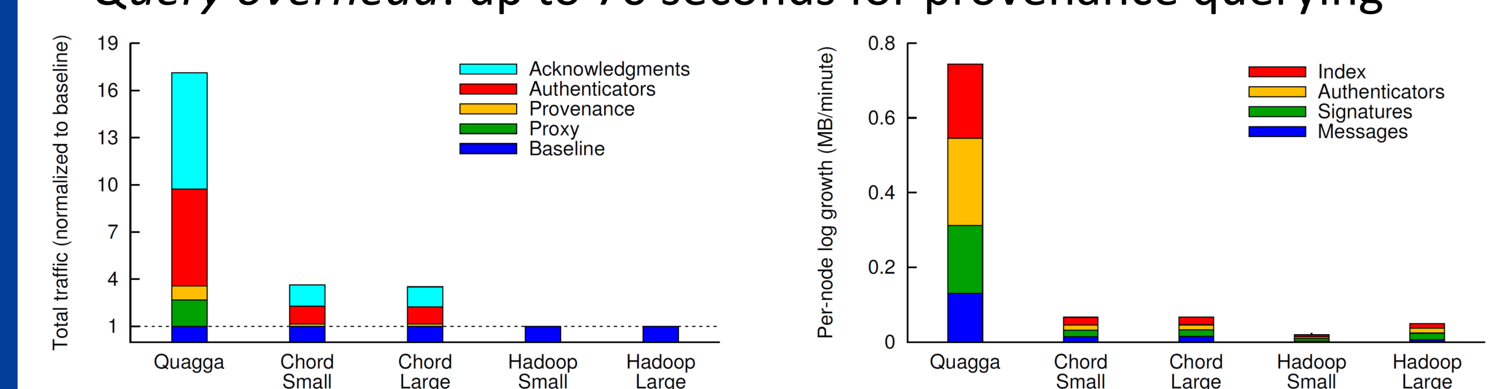
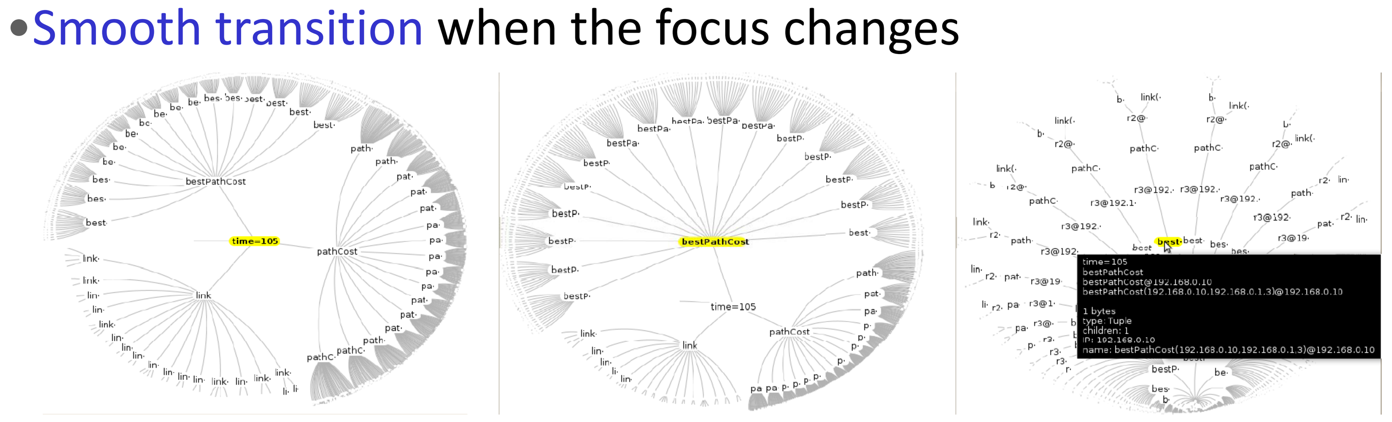


Figure 3. Normalized increase in traffic (left), and per-node log growth (right) excluding checkpoints

Demo: Interactive Visualization Tool

Provenance graph on a hyperbolic plane

- Focus on the part that users are most interested in
- Smooth transition when the focus changes



Future extensions

- Progressively expand provenance vertices
- Incorporate tamper-evident query engine

Acknowledgments

This work was supported by NSF grants IIS-0812270, CNS-0845552, CNS-1040672, CNS-1054229, AFOSR MURI grant FA9550-08-1-0352, DARPA award N66001-11-C-4020, and NPS award N00244-11-1-0008.