Assured Cloud Computing

Roy Campbell
University of Illinois at Urbana-Champaign

Information Trust Institute

Assured Cloud Computing
University Center of Excellence
Sponsored By: AFRL/AFOSR

Motivation

Missions

- Assigned Tasks in Accordance with an Intended Purpose to Accomplish an Assured Mission.

Critical Clouds

- Hybrid (public, private, heterogeneous) clouds that require the realization of “end-to-end” and “cross-layered” security, dependability, and timeliness.

Middleware

- Control, monitoring, assessment of policies, and response

Management

- Configuration and management of dynamic systems of systems with both trusted and partially trusted resources

Multi-tenancy

- Services sourced from multiple organizations

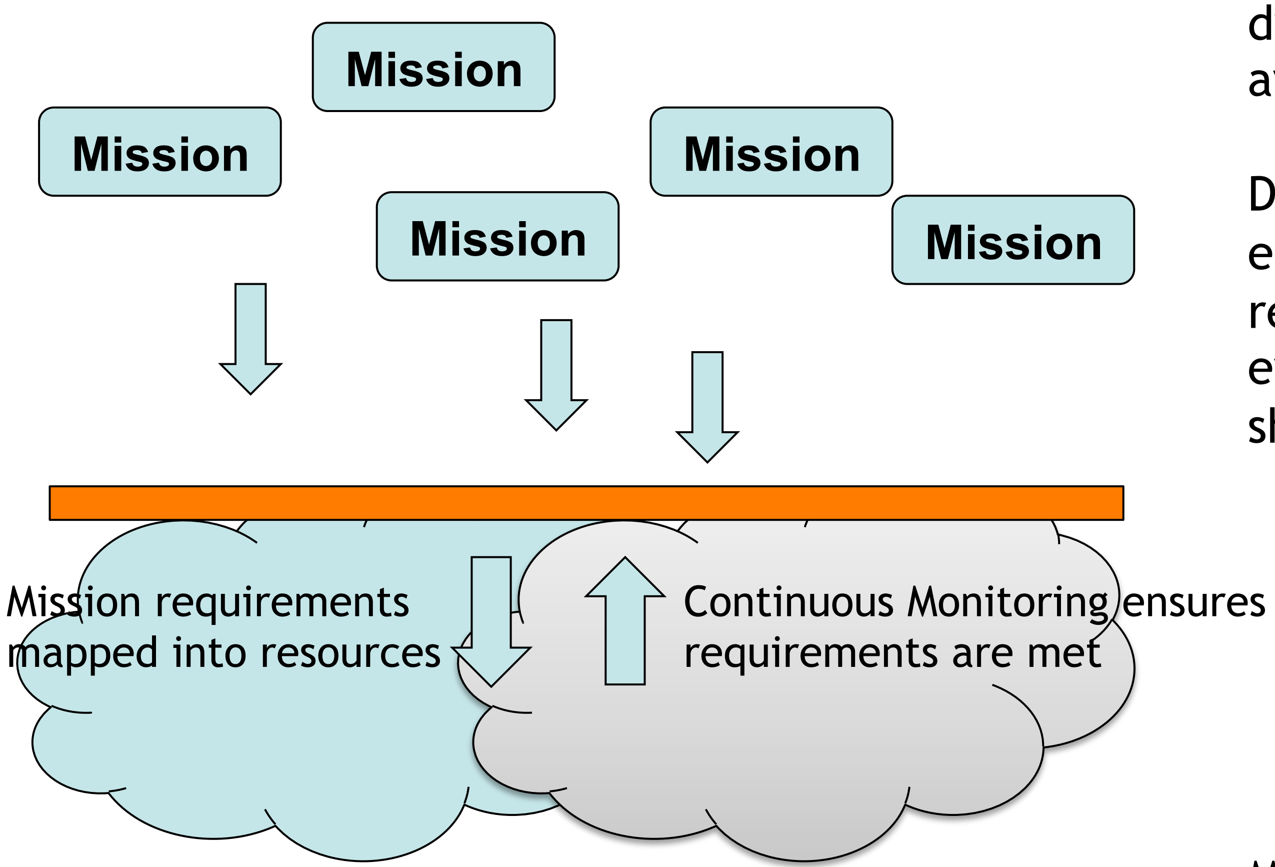
NIST Definition

July 5, 2011:

The NIST Definition of Cloud Computing identified cloud computing as:

„ [...] a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“

Mission: Dependability


Mission
Mission
Mission
Mission
Mission
Mission requirements mapped into resources
Continuous Monitoring ensures requirements are met

Missions are tasks mapped to dynamic resources made available by cloud providers

Dependability: we need to ensure that the mission requirements will be met even when resources are shared

Middleware Layer aware of the security and of the performance of the underlying resources

Mission Requirements

Timeliness of computation

- Fast, parallel, and guaranteed to finish before deadlines
  - *"analysis of the map needs to finish within 2 minutes"*

Security of computation

- Systems performing computation respect security policies
  - *"computation can be performed only on DoD hosts"*
  - *"hosts running the computation cannot run other clients' computation at the same time"*
  - *"host providing authentication should not be accessible from outside the network"*

Monitoring for Policy Compliance

Outline

- Introduction to Assured Cloud Computing
- Monitoring for Security Policy Compliance
  - Local Processing of Policies
  - Distributed Event Processing
  - Security of the Monitoring System
- Experimental evaluation
- Conclusions

Security Policy Compliance

- Security is at the base of "Assured Cloud Computing"

- Security requirements expressed through "policies" that indicate minimal security requirements
  - Approach used in the US by FISMA, PCI-DSS, NERC CIP

Examples:

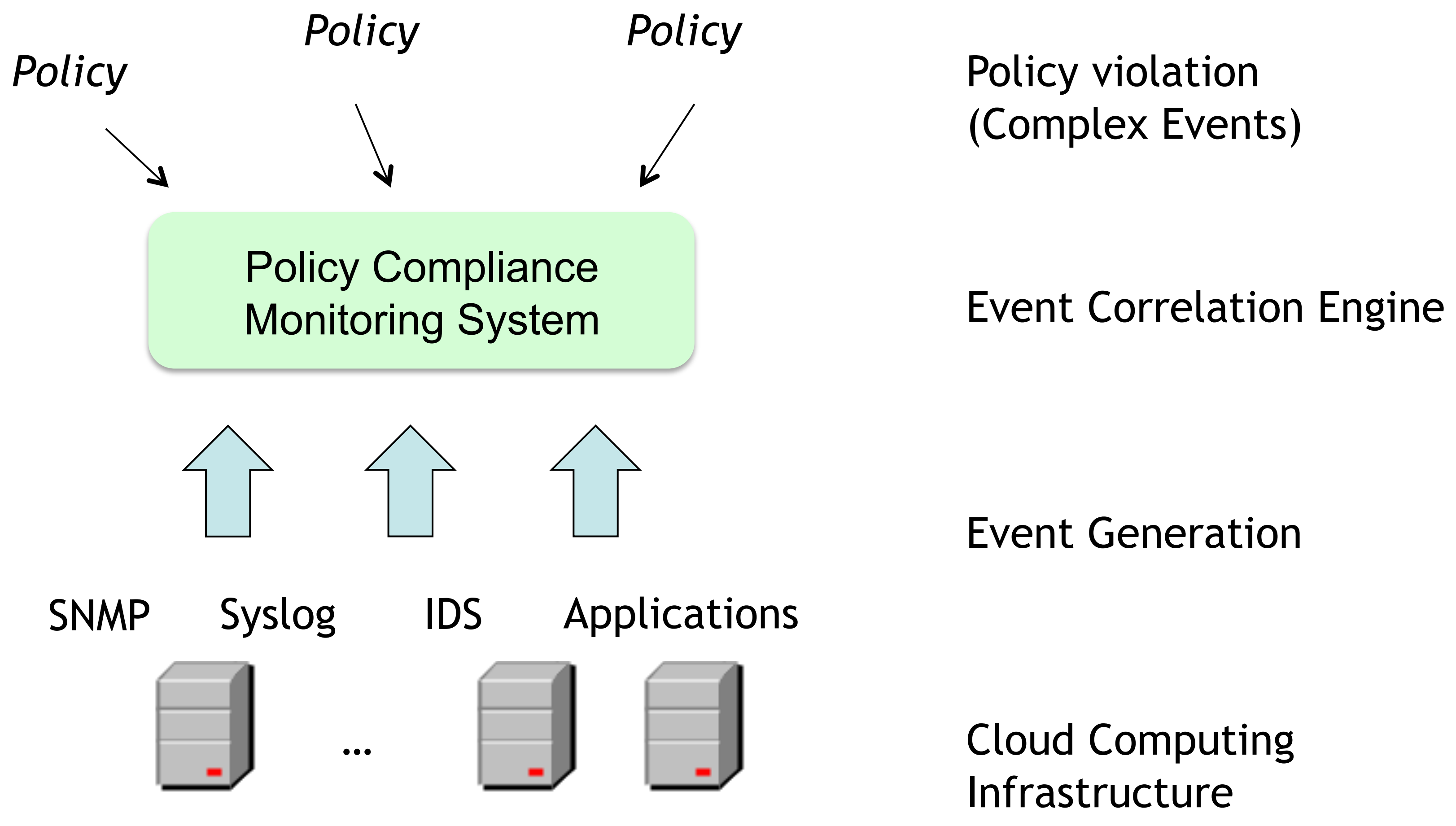
- *A critical device should be placed within a security perimeter*
- *Unprotected devices should not communicate with machines running critical services*
- *Computation on confidential data must performed on hosts under the control of DoD*

Related Work

- Policy-driven approaches to security
  - FISMA, NERC, PCI-DSS all provide documents specifying security requirements on the infrastructure
  - Compliance measured through periodic assessments
- Cloud monitoring
  - Monitoring of continuous variables
    - [Meng et al. – K&DE 2011], [Laguna et al. – Middleware 2009] and others
  - Open-source architecture for monitoring [Chaves et al. – IEEECOMM 2011]
  - We look at discrete events and we show how knowledge of policies can optimize the process
- Access control policy compliance monitoring
  - Focus on complex policies and small number of event generator
    - e.g, [Garg et al. - CCS 2011], [Lam el al. – TRUSTBUS 2009]
- Discrete Event Systems [PADRES, AMIT]
  - We focus on system monitoring and we exploit the fact that events describe resources for optimizing processing

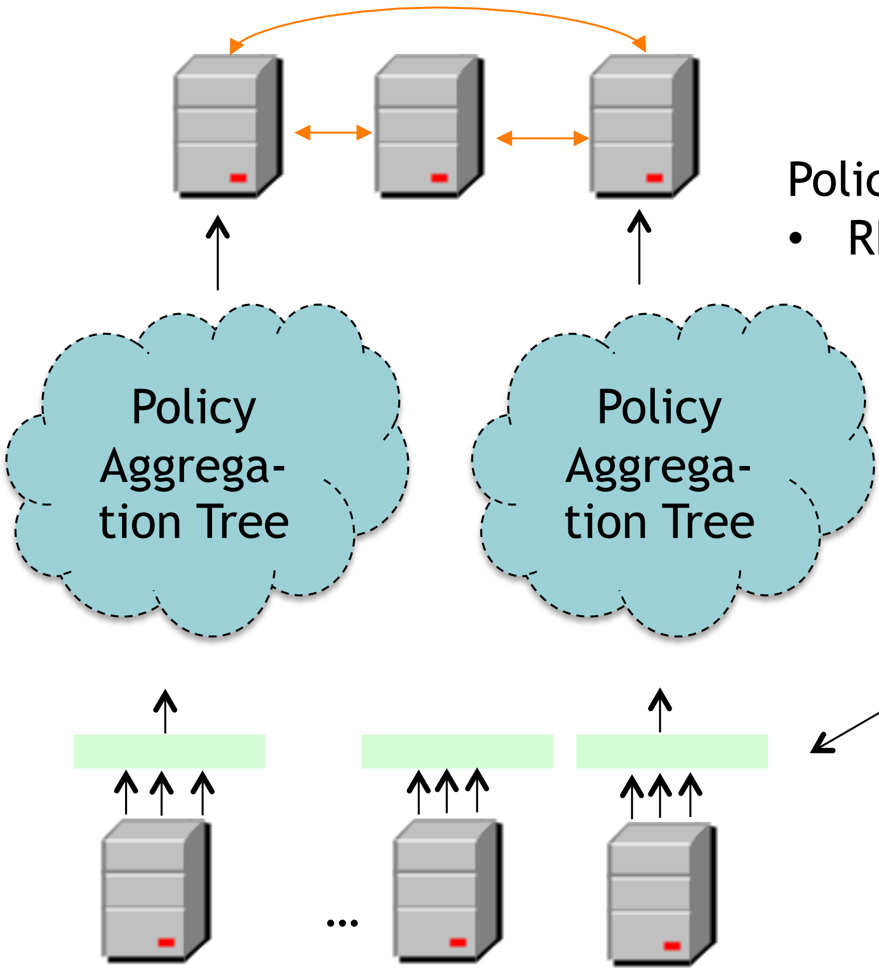
Events as Monitoring Data

- Monitoring compliance requires information about the state of the system – **Discrete Event Processing**



Discrete Event Processing for Policy Monitoring

- Distribution of processing enables scalability and security
- Policies define how events are processed across hosts



Integrity

Policies expressed in Datalog / RDF

- RDF statement: *subject*, ***predicate***, *object*

Violation ← Event Correlation

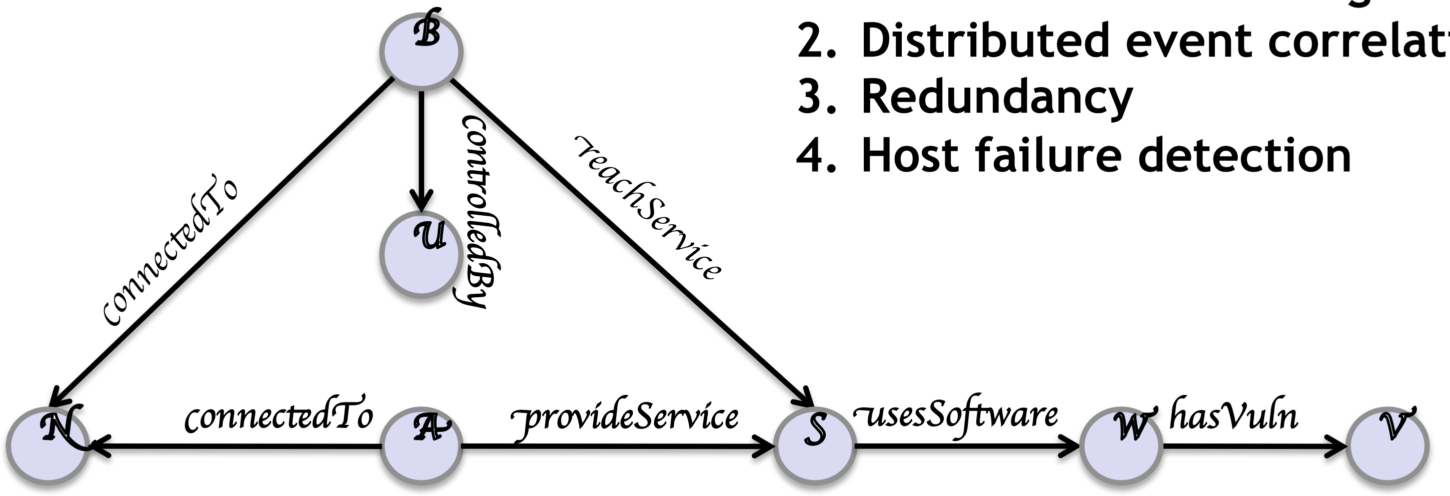
B reachService S,
B connectedTo N,
B controlledBy U,
W hasVulnerability V,
S usesSoftware W,
A provideService S,
A connectedTo N,

Local Event Processing

Policy Analysis

- Analyze policies to distribute the event correlation process
- RDF Policies represented as graphs
    - Variables are nodes, edges are predicates

Violation ← B reachService S, B connectedTo N, B controlledBy U,
W hasVulnerability V, S usesSoftware W,
A provideService S, A connectedTo N



Graph-based analyses enable:

1. Local Event Processing
2. Distributed event correlation
3. Redundancy
4. Host failure detection

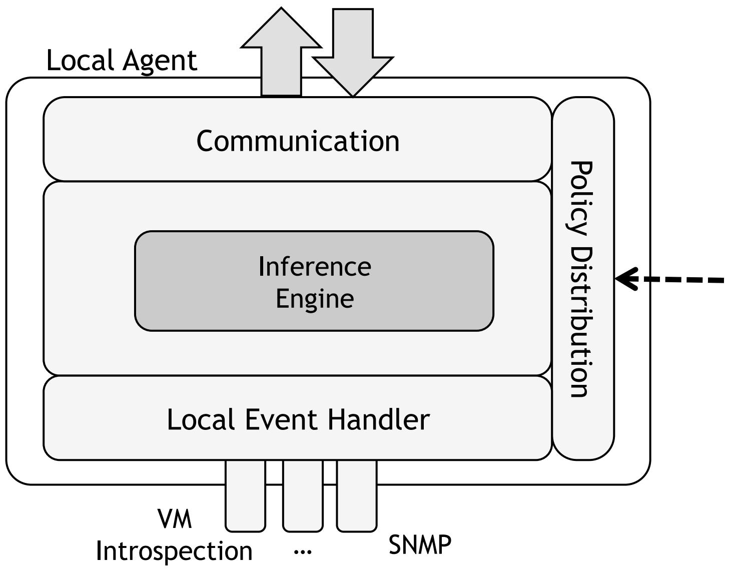
1) Local Event Middleware

Partial processing of policies is delegated to local nodes to reduce the overall event exchange

Policies are distributed to a monitoring middleware present on local hosts

Local event processing detects violations based on complex events generated locally

Distributed reasoning is used for correlating events across multiple hosts

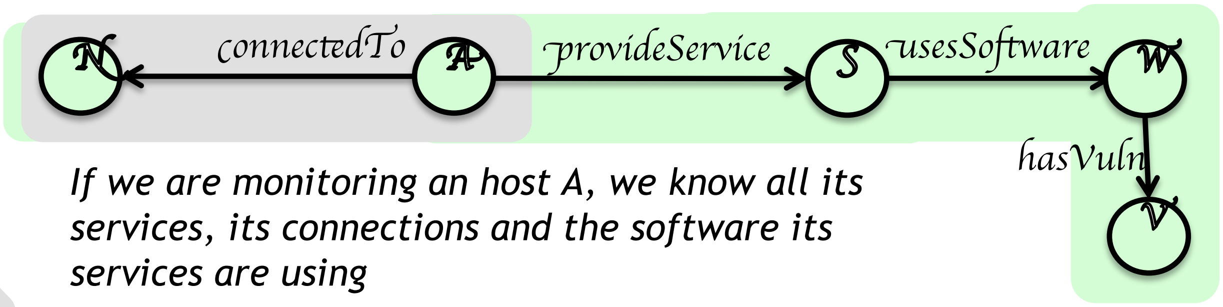


Montanari M., Chan E., Larson K., Yoo W., Campbell R.H. , **"Distributed Security Policy Conformance,"** IFIP SEC 2011

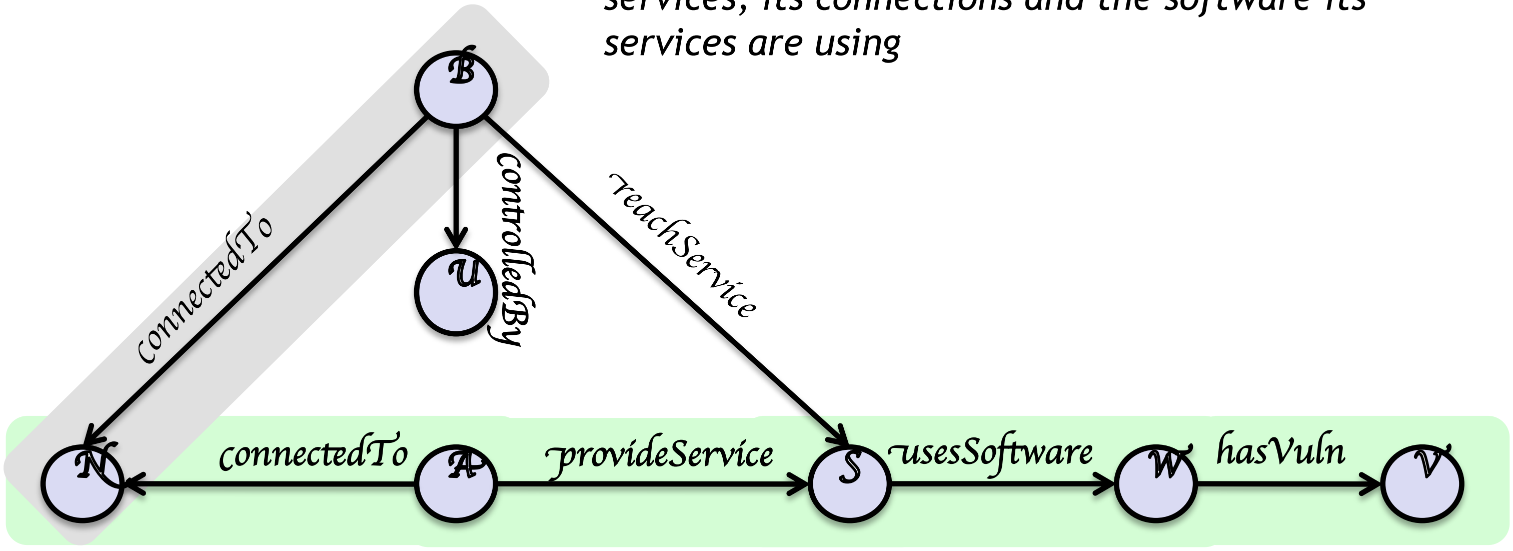
1) Local Policy Identification

- **Definition of Local Portion of the policy**
  - Exploits knowledge about what is being monitored
  - Only events satisfying all local conditions are forwarded outside

Monitoring system meta-info used for identifying local "complete" events



If we are monitoring an host A, we know all its services, its connections and the software its services are using

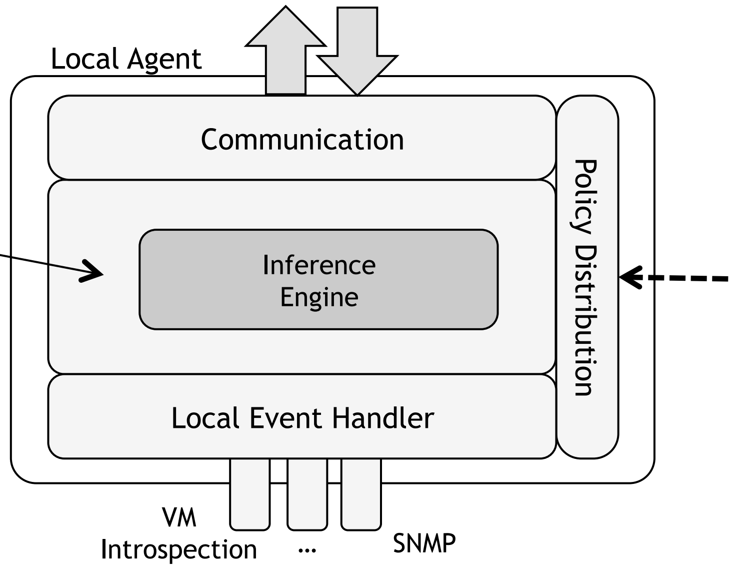


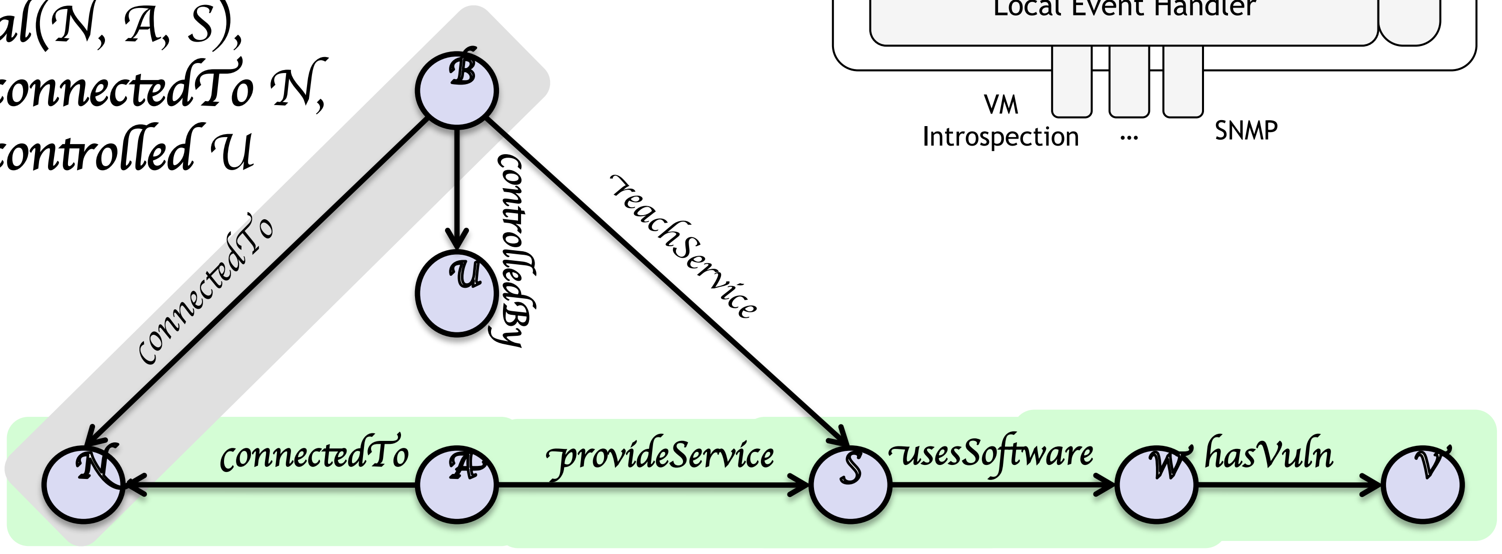
1) Local Policy Equivalent Rewrite

- Local portion of the policy is processed in the inference engine

local(N, A, S) ←
  A connectedTo N
  A provideService S
  S useSoftware W
  W hasVuln V

Violation ←
  local(N, A, S),
  B connectedTo N,
  B controlled U

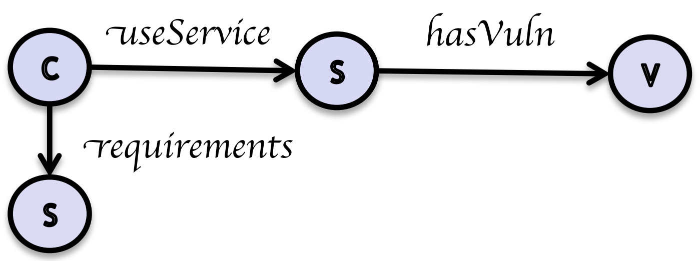




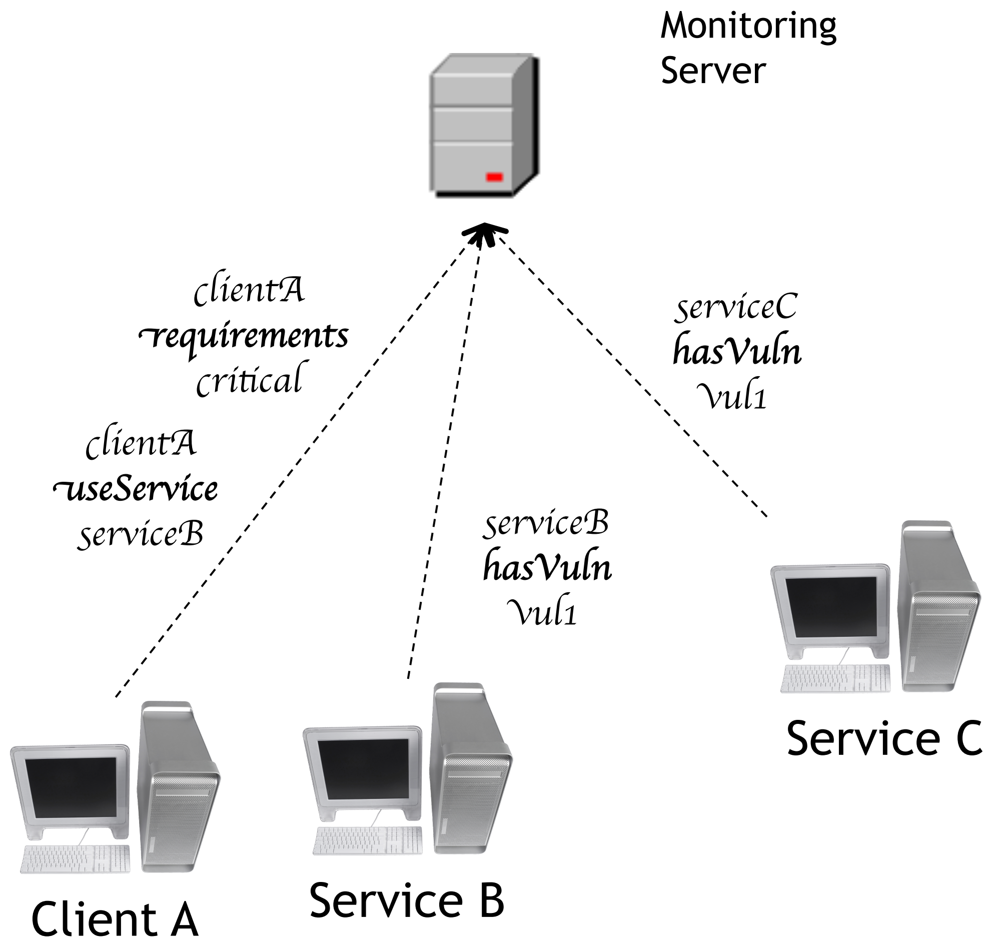
2) Distributed Correlation – Basic Architecture

A client with "critical" security requirements should not use a service on a vulnerable machine

Violation ← *C* **requirements** *critical*,
C **useService** *S*,
S **hasVuln** *V*



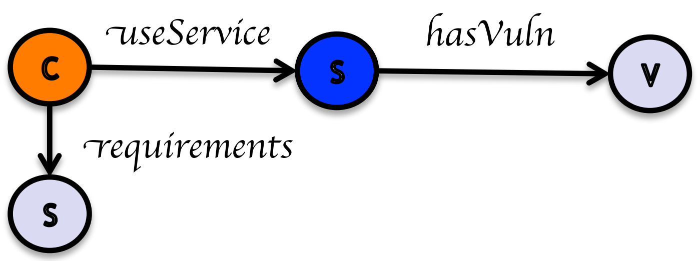
Intuition: two events connected in the graph share the value for one of the variables



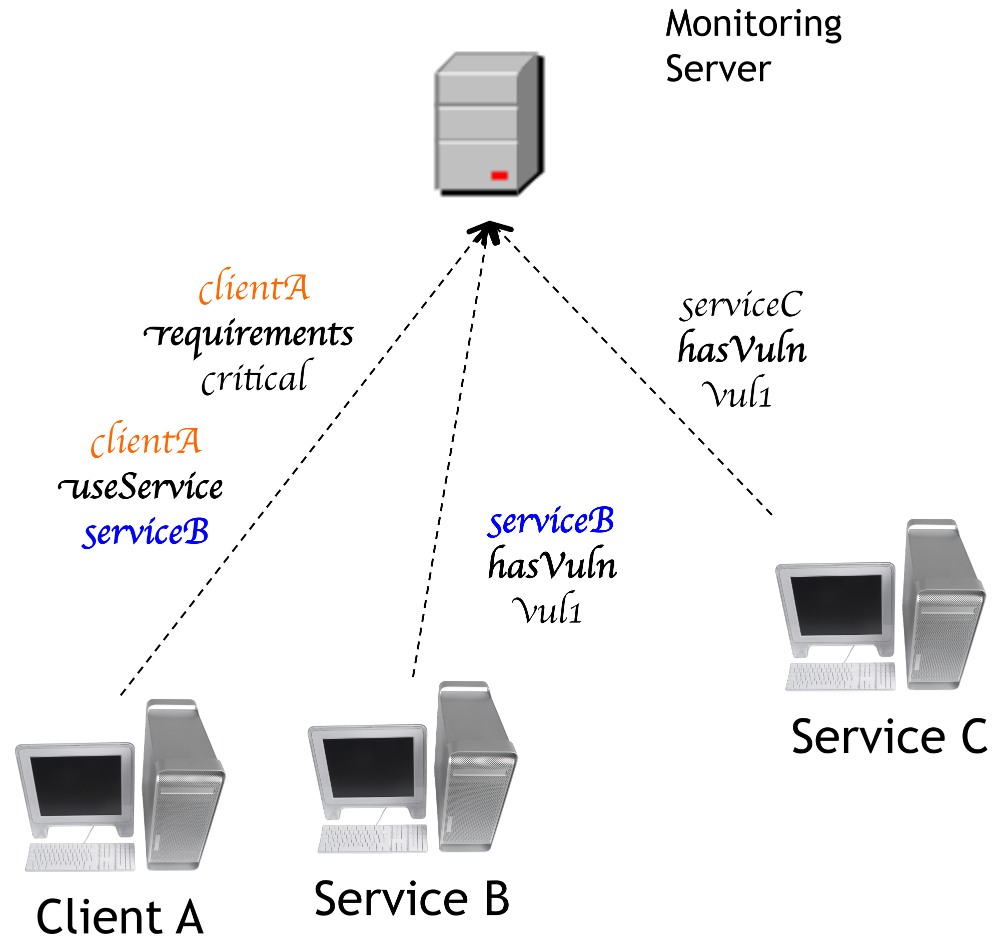
2) Distributed Correlation – Basic Architecture

A client with "critical" security requirements should not use a service on a vulnerable machine

Violation ← *C* **requirements** *critical*,
C **useService** *S*,
S **hasVuln** *V*

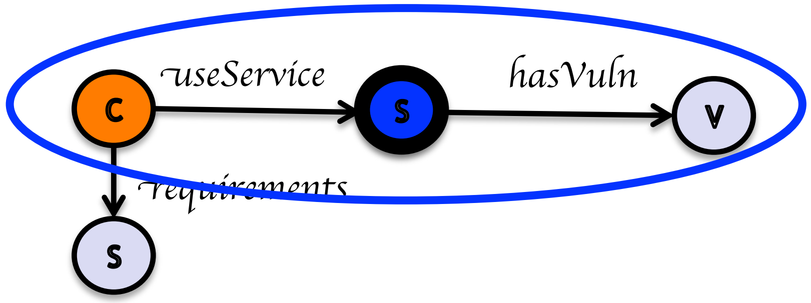


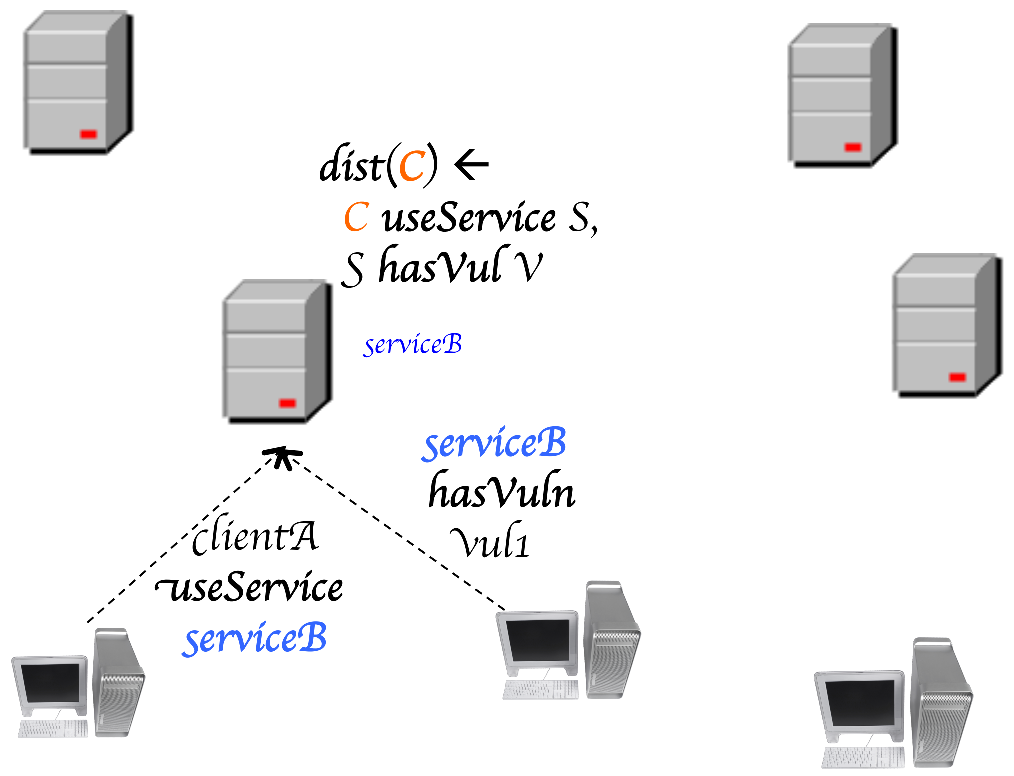
Intuition: two events connected in the graph share the value for one of the variables



2) Distributed Correlation

We use the policy to define **Aggregation Policy Trees** that correlate subsets of the policy at each level

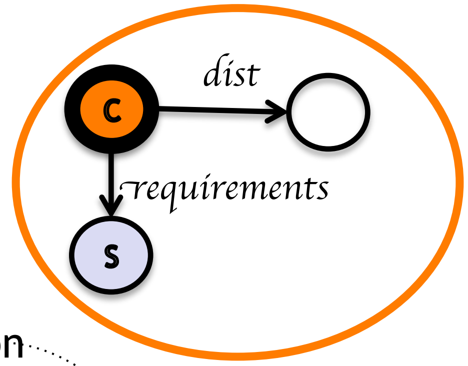


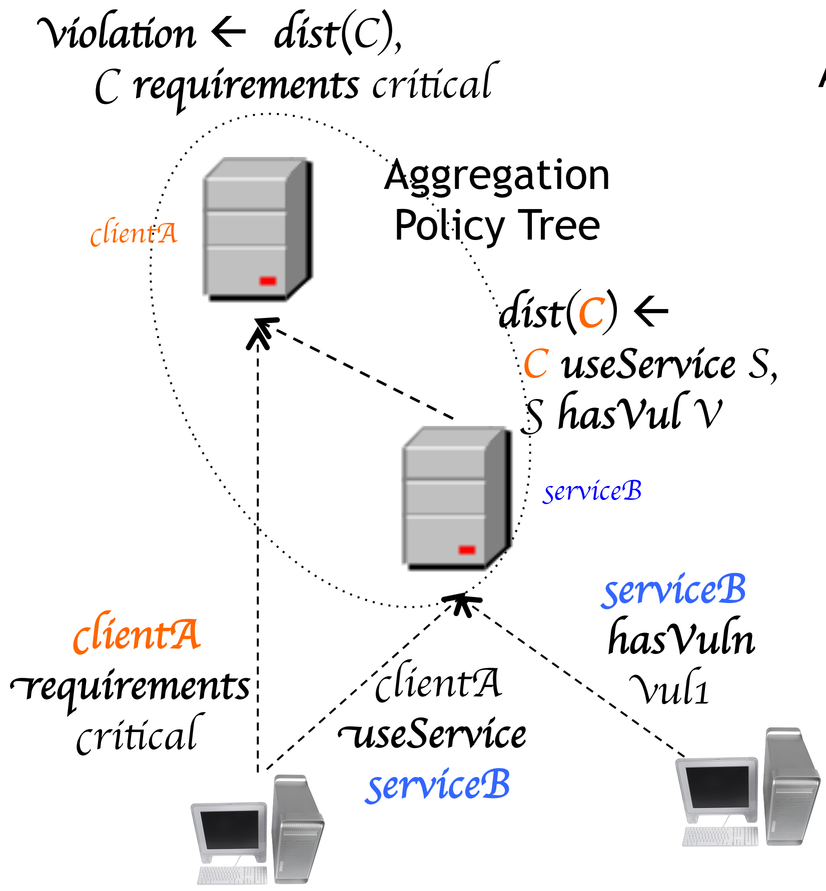


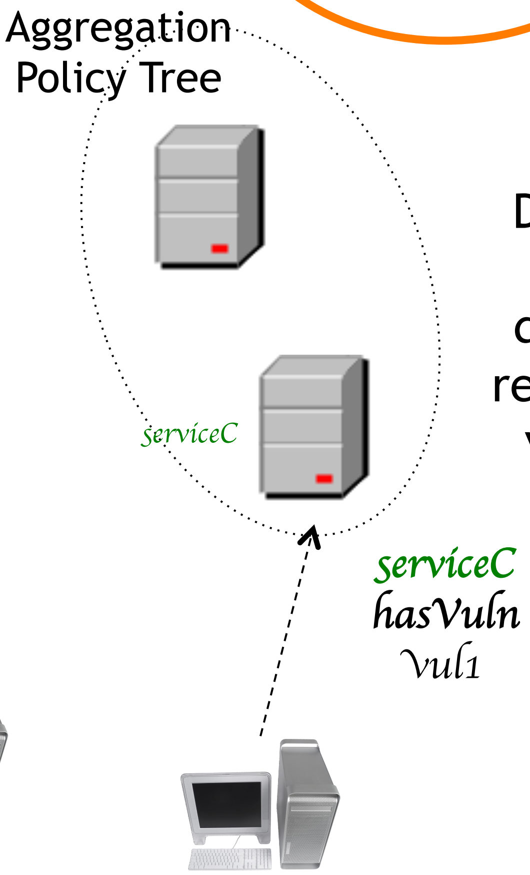
Different portions of the policy are matched at different nodes, and the result is forwarded so that validation can continue

2) Distributed Correlation

We use the policy to define **Aggregation Policy Trees** that correlate subsets of the policy at each level




violation ← dist(C), C requirements critical
Aggregation Policy Tree
clientA
dist(C) ← C useService S, S hasVul V
serviceB
clientA requirements critical
clientA useService serviceB
serviceB hasVuln Vul1



Different portions of the policy are matched at different nodes, and the result is forwarded so that validation can continue

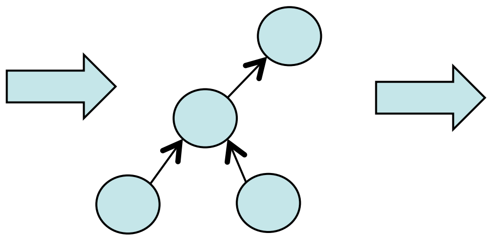
2) Distributed Online Rule Analysis

Intuition formalized in an algorithms with two steps

Compilation: Creates a set of *rule elements* and *state triggers*

Violation ←
C ***requirements*** *critical,*
C ***useService*** *S,*
S ***hasVulnerability*** *V*

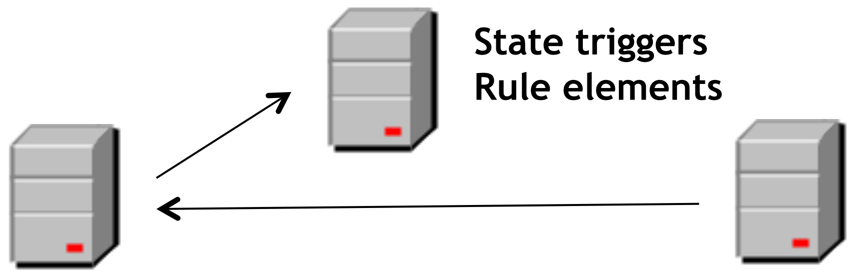
aggregation tree



State triggers: *define which messages to send and their destination*

Rule elements: *partial validation of the policy*

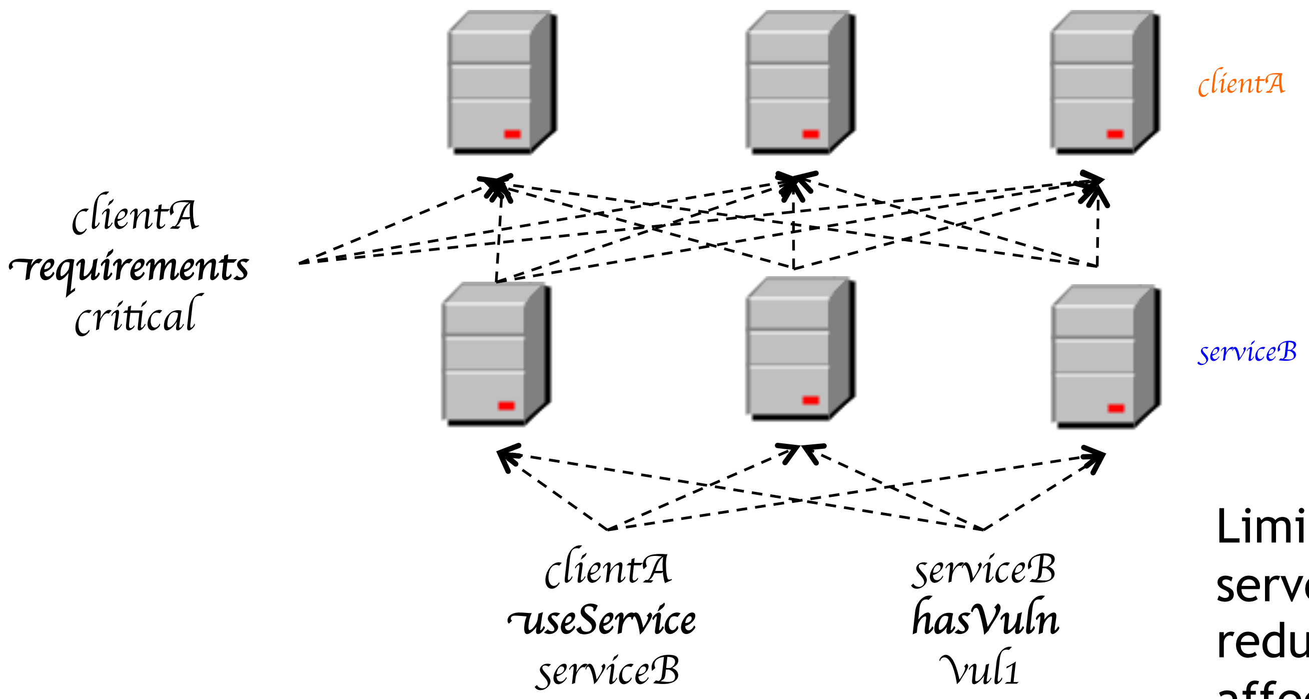
Execution: Messages are exchanged across hosts



3) Protecting against server integrity compromises

Each level of the policy aggregation tree can be made redundant

violation ←
C ***requirements*** *critical*,
C ***useService*** *S*,
S ***hasVuln*** *V*



Limited load on each server permits to do redundant work without affecting performance

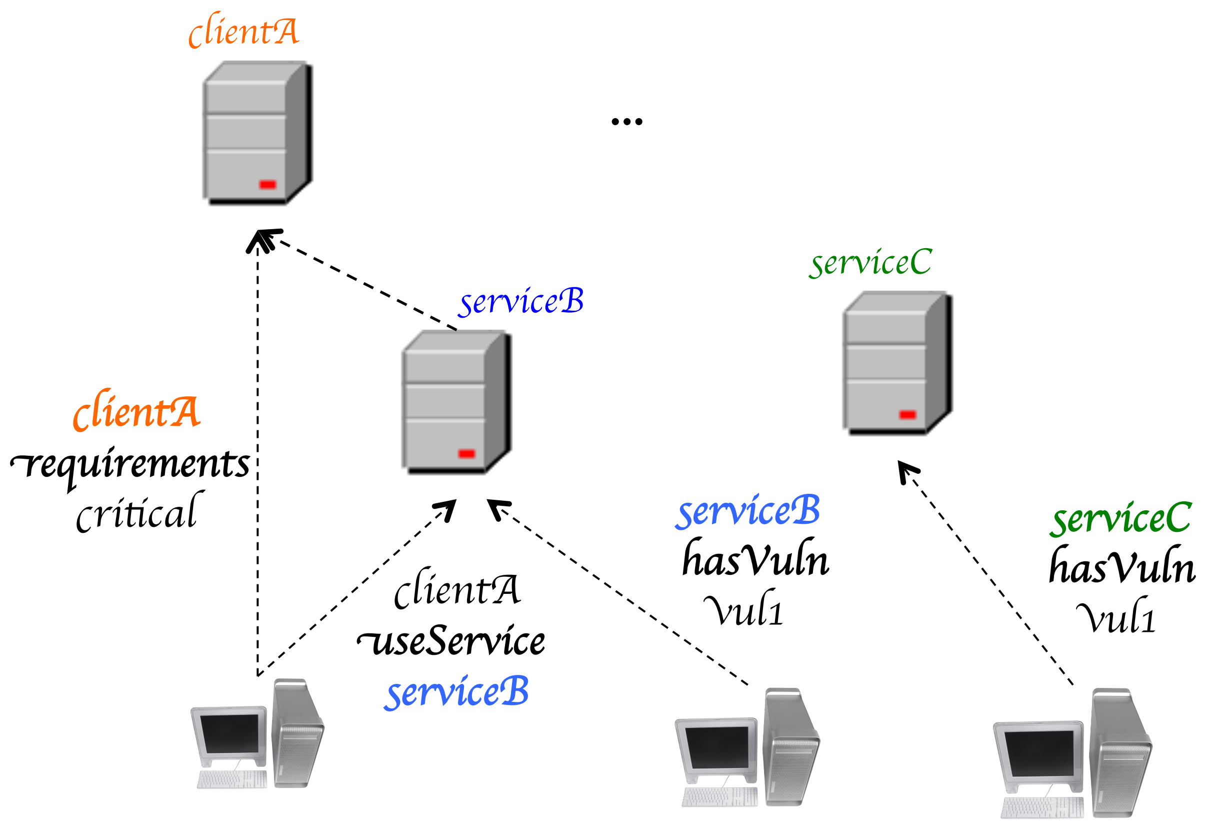
The compromise of one server does not affect integrity

The compromise of the majority of servers at the same level only affects one aggregation tree, not the entire policy validation

4) Detection of host failures

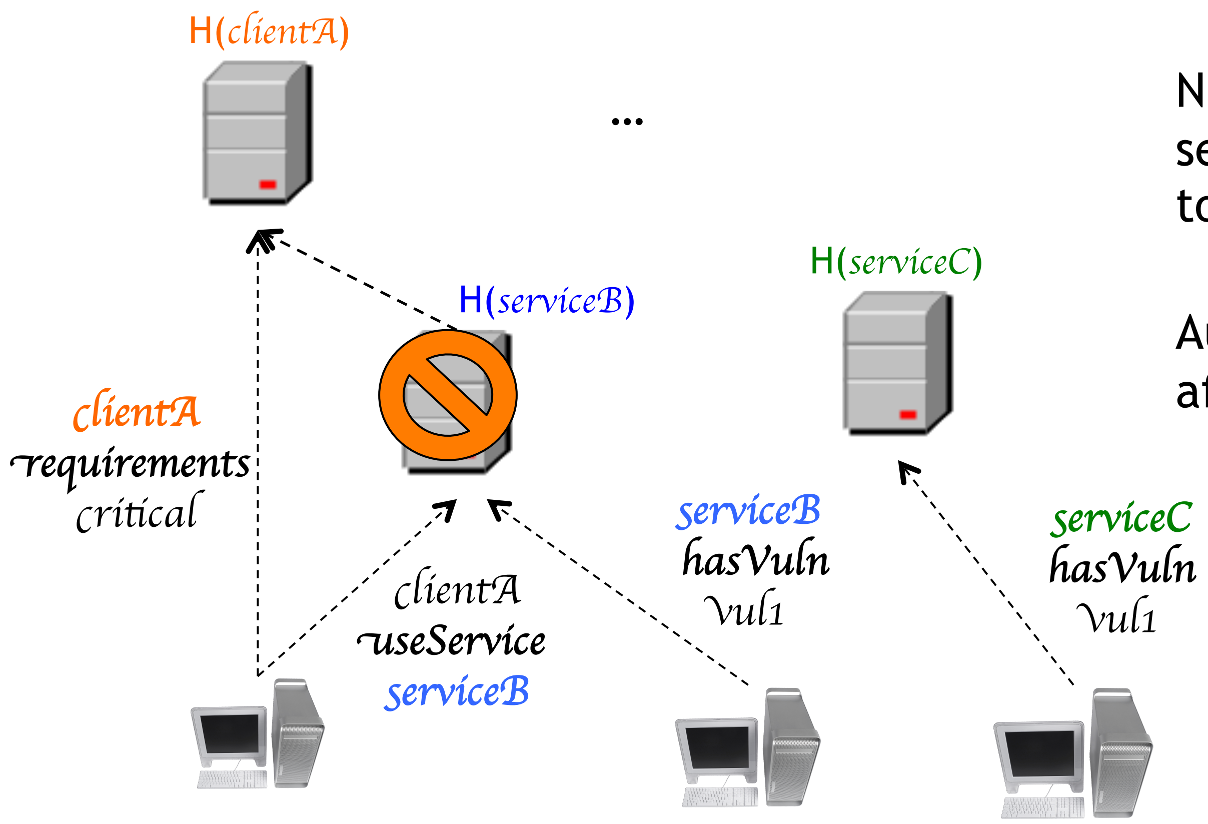
Scalable communication and detection of failures obtained by building the system on top of a DHT

Correlation servers are connected using a DHT



4) Detection of Failures

Scalable communication and detection of failures obtained by building the system on top of a DHT



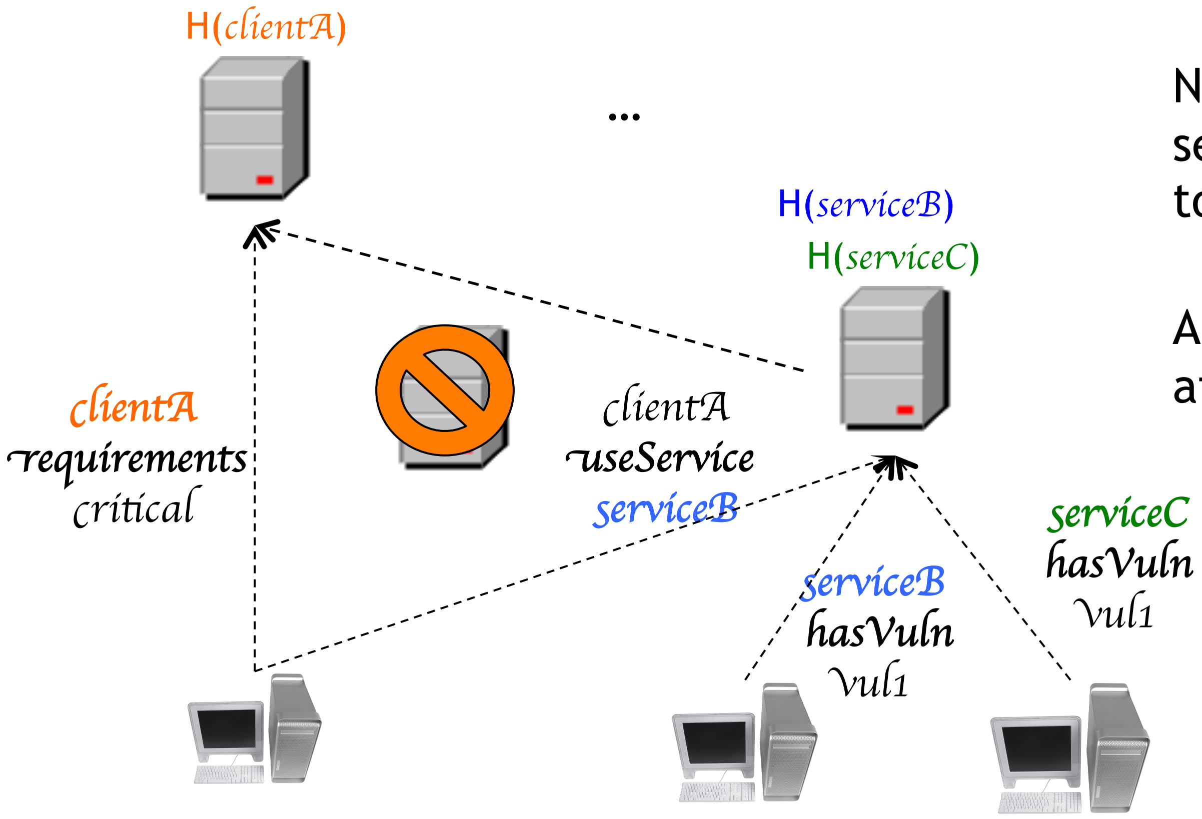
Correlation servers are connected using a DHT

Name of resources is used for selecting the correlation node to use

Automatic reconfiguration after server failures

4) Detection of failures

Scalable communication and detection of failures obtained by building the system on top of a DHT



Correlation servers are connected using a DHT

Name of resources is used for selecting the correlation node to use

Automatic reconfiguration after server failures

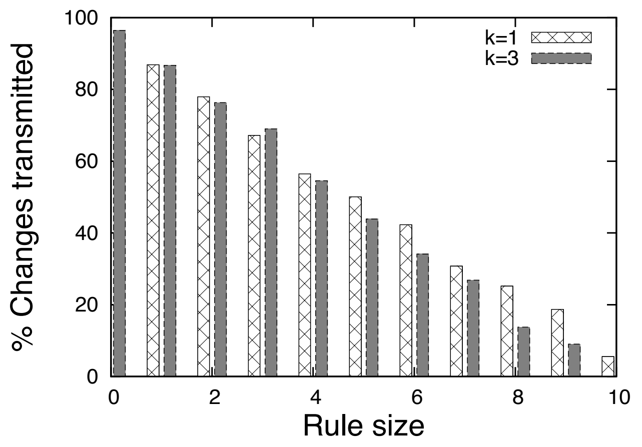
Experimental Results

- Odessa implemented in Java and C
  - Communication built on top of Freepastry
  - To increase the trustworthiness of agents, we run them in Dom0 when possible.

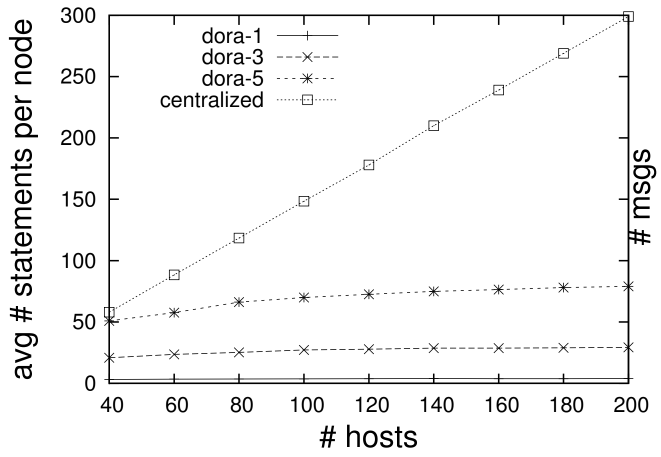
Mechanism	Configuration obtained
Dom0 (XenAccess, file system)	Running processes, network connections, configuration files
Host VM (Linux kernel module)	Fast detection of new network communications

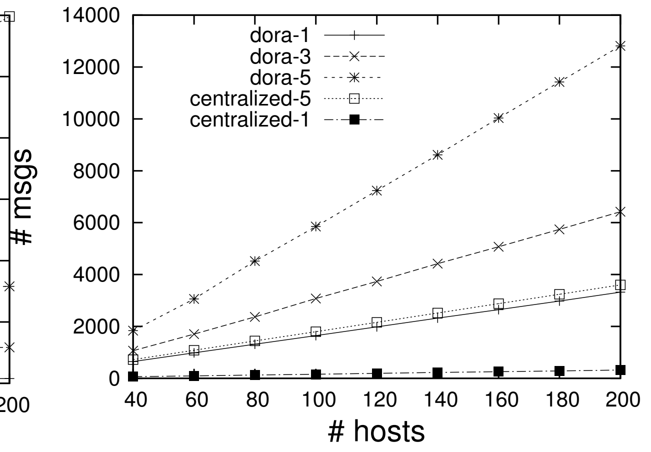
- Using such information, we implemented policies for validating:
  - Presence of specific programs
  - NFS authorizations across networks
  - Attack graph generation

Delegation Experiments: Reduced Load

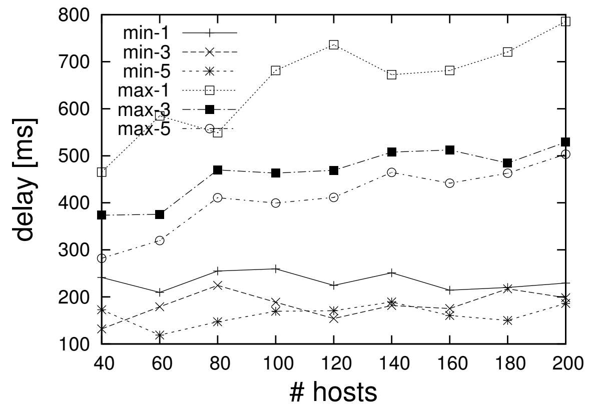


Local processing reduces the amount of events delivered (rule size: local portion of the rule)



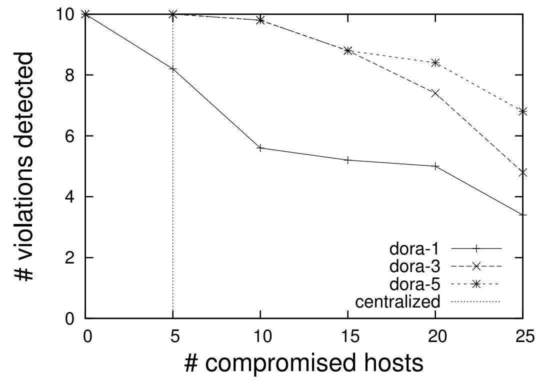


Correlation servers need to store a limited number of events and send a limited number of messages



← Delay in detection violations remains limited

Replication provides → graceful degradation of integrity in case of compromises



Future Work

- Synthesize global properties from placement
- Confidentiality of monitoring data
  - Protecting confidentiality of event data both within a cloud and in between cloud providers
- Integrity of event sources
  - What if data are corrupted at the source?
- Explore automatic reconfigurations in assured cloud computing
  - Policy violation data can be used for reacting to problems
- Availability

Conclusions

- Assured Cloud Computing requires technologies that permit predictability in performance and in security.

- Monitoring is required for maintaining dependability, security, and performance in a dynamic cloud environment
- We focus on policy compliance to monitor security requirements

- Our approach formalizes analyses of policy languages to identify appropriate distributed placements for security policy compliance evaluation
  – *We use information about the distributed system and the policies to generate a set of equivalent rules to enable a* ***scalable*** *and* ***secure*** *detection of complex events*

- Our approach outperforms centralized approaches and protects the system against integrity compromises

Bibliography

[Meng et al. – K&DE 2011] Meng, S., & Liu, L. (2011). State Monitoring in Cloud Datacenters. Knowledge and Data Engineering, IEEE Transactions on, 23(9), 1328-1344.

[Chaves et al. – IEEECOMM 2011] Chaves, S. A. D., Uriarte, R. B., & Westphall, C. B. (2011). Toward an Architecture for Monitoring Private Clouds. IEEE Communications Magazine, (December), 130-137.

[Garg et al. - CCS 2011] Deepak Garg, Limin Jia and Anupam Datta. Policy Auditing over Incomplete Logs: Theory, Implementation and Applications. ACM CCS 2011

[Lam el al. – TRUSTBUS 2009] Peifung E. Lam, John C. Mitchell and Sharada Sundaram. A Formalization of HIPAA for a Medical Messaging System. Lecture Notes in Computer Science, 2009, Volume 5695/2009, 73-85.

[Laguna et al. – Middleware 2009] Ignacio Laguna, Fahad A. Arshad, David M. Grothe and Saurabh Bagchi. How to Keep Your Head above Water While Detecting Errors. IFIP/ACM/IEEE Middleware 2009

[PADRES] Fidler, E., Jacobsen, H., & Li, G. (2005). The PADRES distributed publish/subscribe system. Feature Interactions in Telecommunications and Software Systems.

[AMIT] Adi, A., & Etzion, O. (2004). Amit-the situation manager. The VLDB Journal â" The International Journal on Very Large Data Bases, 13(2)