Efficient, Usable Proof-Construction Strategies for Distributed Access-Control Systems

Mike Reiter

University of North Carolina at Chapel Hill

Joint work with Lujo Bauer and Scott Garriss.

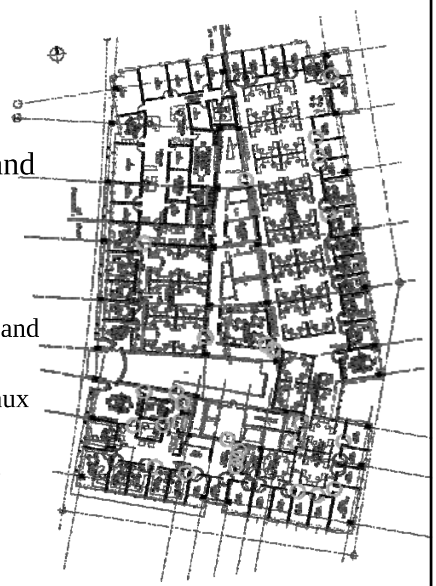
Device-Enabled Authorization

- Smartphones on a trajectory to “win” in the market
  - Stand to inherit mobile phone market that shipped over 1.1 billion units in 2007 [IDC]—or **more than one phone per six people in the world**
- Goal: to use smartphones to intelligently control environment
  - Loan you my car without giving you my phone
  - Send money from my phone to my daughter’s phone
  - Give my secretary temporary access to my email without revealing information (e.g., password) that could be used at a later time
  - Use my phone to open my hotel room door, without ever stopping by the front desk

… and do it all from a distance

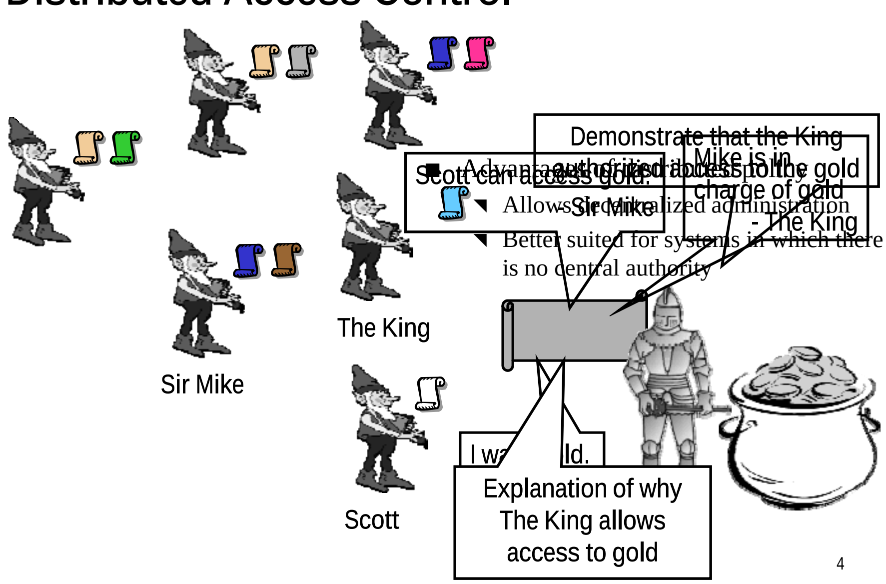
The Grey System

- Universal, flexible, end-user-driven access-control system for physical and virtual resources
- Deployed in CMU’s Collaborative Innovation Center
  - Approximately 35 Grey-capable doors and 30+ users at the moment
  - Grey-compatible Windows XP and Linux login modules
  - Access-control module for web servers under development
- Plus a deployment coming at UNC



3

Distributed Access Control



4

Logic-Based Access Control

- Advantages of expressing policy in logic
  - Unambiguous policy specification
  - Allows flexible delegation, and role and group creation
  - Greater assurance of correctness
- Demonstration can take the form of a logical proof
  - Efficiently verifiable
  - Resource monitor (knight) verifies that:
    - The credentials are valid
    - The credentials imply that access should be granted
  - Knight need not know the entire policy beforehand
- Challenge: proof construction must be efficient

5

Challenges in Proof Construction

- Potentially many different ways to derive authority
  - Must look for them all before determining that access is not authorized
- Credentials are distributed
  - Nodes not always online
  - Communication may incur a high latency
- Desired feature: allow dynamic credential creation
  - Requesting a credential may result in user interaction
  - Must guide user to create appropriate credential

6

Talk Outline

- Introduction
- **Distributed Proof Construction**
  - Lazy proving strategy
  - Evaluation
- Efficient Proving for Practical Systems
- Identifying and Resolving Policy Misconfigurations
- Related Work and Conclusions

7

Sample Access-Control Logic

<u>Expressing Beliefs:</u>

`Bob signed` ***F***
- Bob states (cryptographically) that he believes that *F* is true

`Bob says` ***F***
- It can be inferred that Bob believes that *F* is true

<u>Types of Beliefs:</u>

`Bob says open(resource, nonce)`
- Bob wishes to access a resource

`Bob says (Alice speaksfor Bob)`
- Bob wishes to delegate all authority to Alice

`Bob says delegate(Bob, Alice, resource)`
- Bob wishes to delegate authority over a specific resource to Alice

8

Example Inference Rules

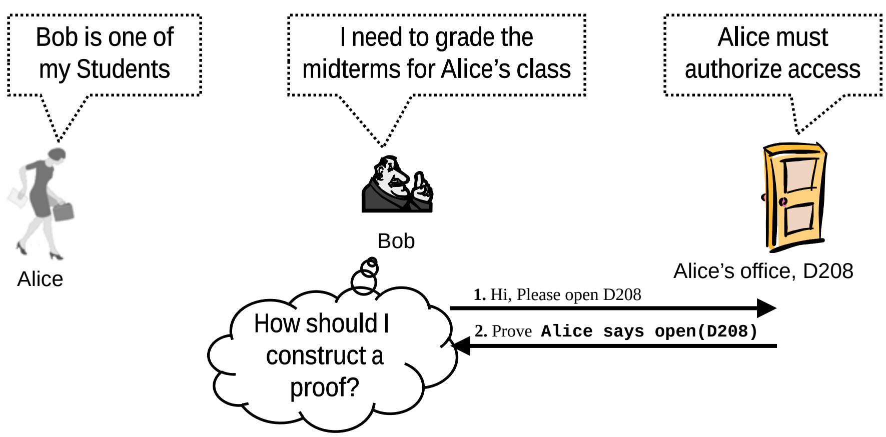
$$\frac{\texttt{A signed F}}{\texttt{A says F}}$$

$$\frac{\texttt{A says (B speaksfor A)} \qquad \texttt{B says F}}{\texttt{A says F}}$$

$$\frac{\texttt{A says delegate(A, B, resource)} \qquad \texttt{B says open(resource)}}{\texttt{A says open(resource)}}$$

9

Example Scenario



10

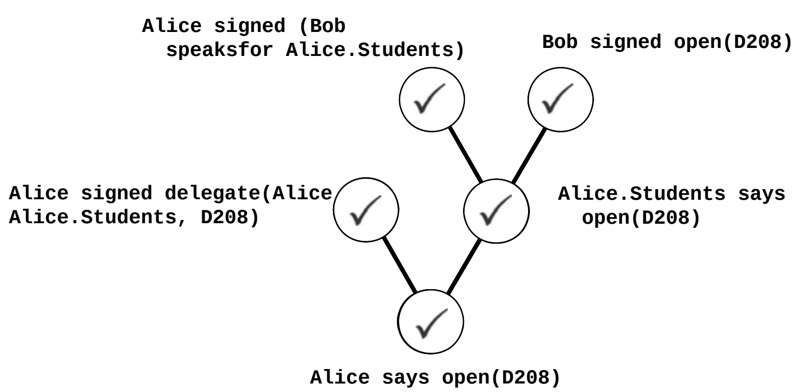
Common Proving Approach: Backward Chaining

- Used by Prolog

- Decompose goal into subgoals using tactics
  - A *tactic* applies one or more inference rules in a single step
- Recursively try to prove each subgoal
- Only performs computation relevant to query at hand

11

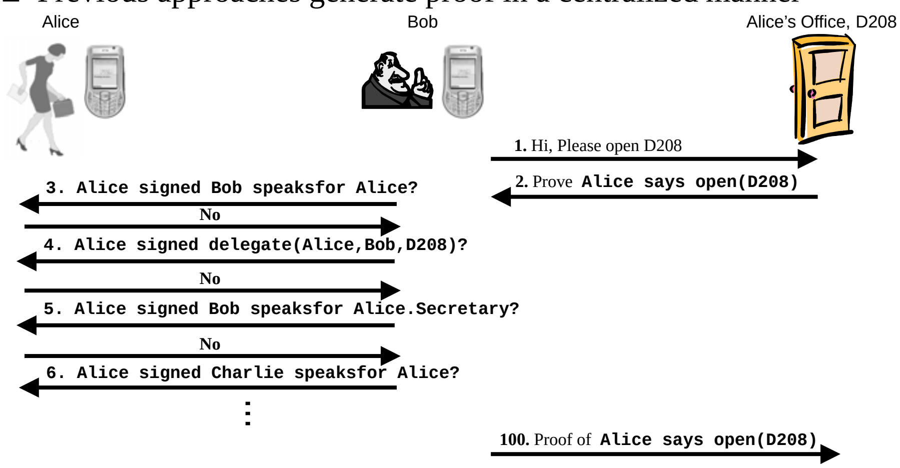
Example Proof (Using Backward Chaining)



12

Traditional Approach to Proof Generation

- Previous approaches generate proof in a centralized manner



- We classify this approach as "Eager"

Talk Outline

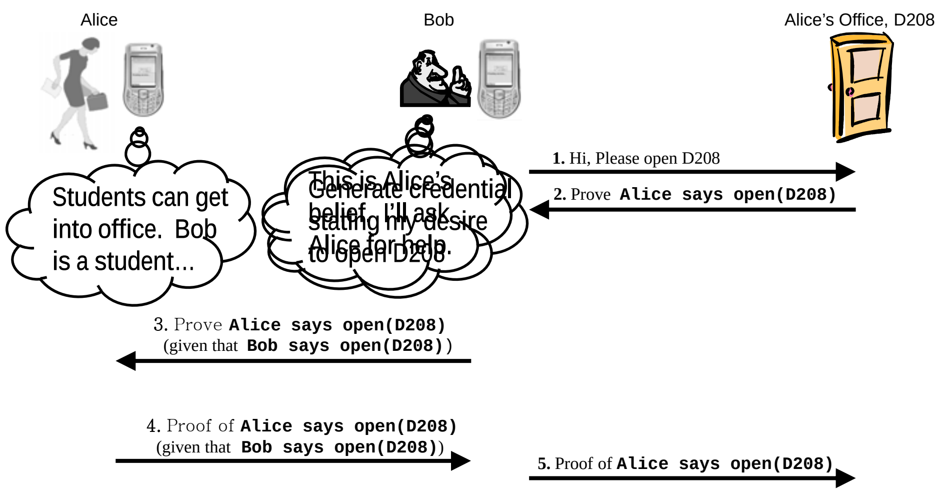
- Introduction
- Distributed Proof Construction
  - **Lazy proving strategy**
  - Evaluation
- Efficient Proving for Practical Systems
- Identifying and Resolving Policy Misconfigurations
- Related Work and Conclusions

Our Proposal: A Lazy Approach

- Ask Alice to *prove* **`Alice says open(D208)`**
- Alice's prover may use its local knowledge to complete proof
  - Certificate lookups may now be performed locally, rather than remotely
- More generally, when reasoning about the beliefs of principal **`A`**,
  - Always ask **`A`** to prove **`A says F`**

15

Lazy Proof Generation



16

Talk Outline

- Introduction
- Distributed Proof Construction
  - Lazy proving strategy
  - **Evaluation**
- Efficient Proving for Practical Systems
- Identifying and Resolving Policy Misconfigurations
- Related Work and Conclusions

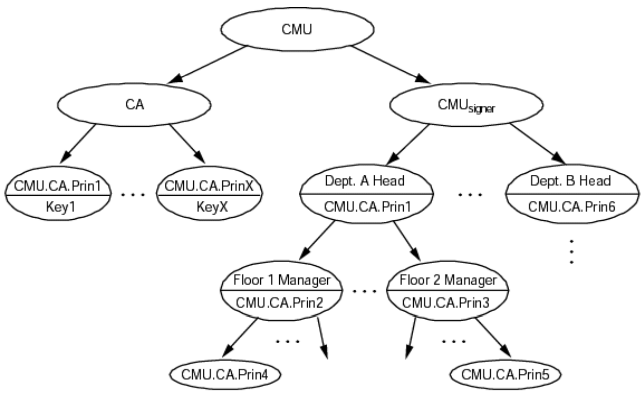
17

Analysis

- No loss in proving ability compared to eager approach
  - Proof in paper
- Evaluation metric: number of network interactions (*requests*) made by prover
  - Lazy Requests → Proof requests
  - Eager Requests → Credential lookups
- Requests can incur a high latency, delay proof construction
  - Requests may travel over the cellular network or result in user interaction
- Simulate proof construction for all authorized accesses; average

18

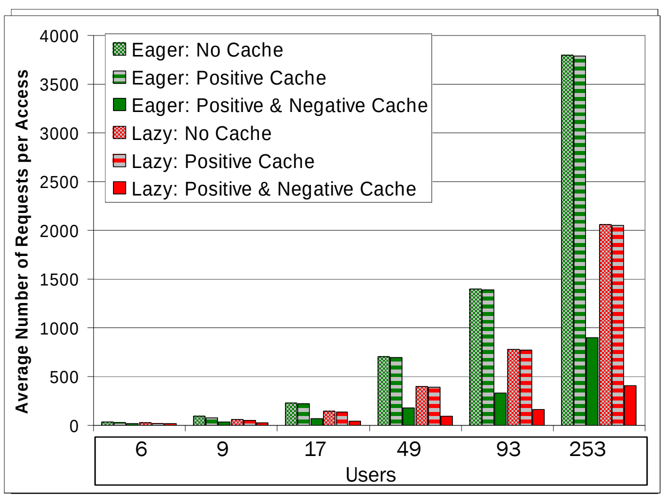
Policy for Simulations



- Based on real policy for ECE Department
- Delegations are made to roles, then principals bound to those roles
- CA run by university, binds principal names to public keys
- Simulate various policy sizes by specifying the branching coefficients for the tree rooted at CMU_{signer}

19

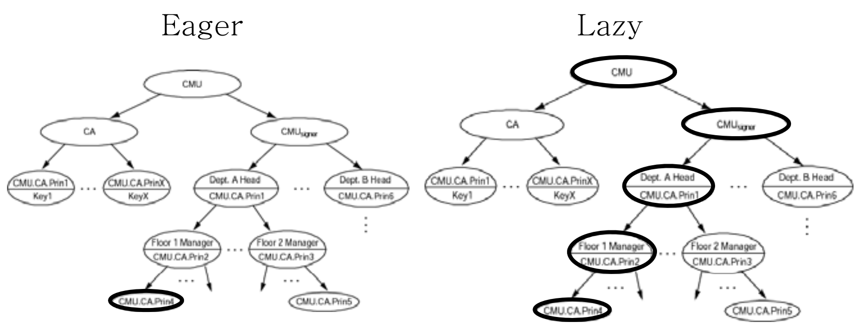
Eager vs Lazy: Initial Access



- Lazy makes half as many requests as Eager
- Optimization: Cache completed proofs (Positive Cache)
- Most redundant requests fail
- Optimization: Cache the fact that a request failed (Negative Cache)
- Negative Cache reduces queries by 75%

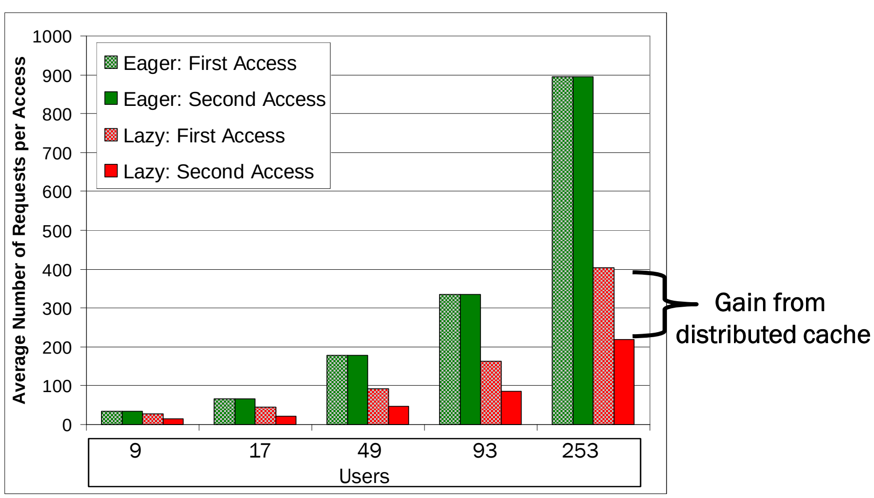
20

Principals that Cache Results



21

Access by a Second Principal



22

Limitations

- Asking CMU to prove `CMU says open(X)` results in:
  - CMU receiving a large number of requests
  - An unnecessary request if derivable from local credentials
  - Both of these addressed in next section
- Consistency
  - Certificates may be created and revoked
  - Need to update all corresponding positive and negative caches
    - Use mechanism of [Minami and Kotz ’06]
  - Frequent changes → all credentials might not be simultaneously valid
    - Use multiple rounds or commitments [Lee and Winslett ’06]

23

Checkpoint

- Distributed (lazy) proof construction :
  - Completes proofs with fewer requests than eager
  - Distributed cache enables further gains for lazy
- Caching negative results reduces queries by 75% (for both lazy and eager) in our tests

24

Talk Outline

- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
  - **Requirements**
  - Algorithm
  - Evaluation
- Identifying and Resolving Policy Misconfigurations
- Related Work and Conclusions

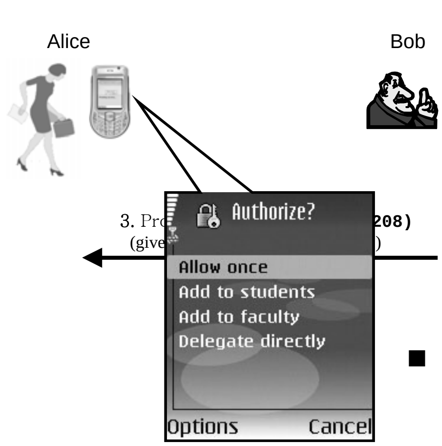
25

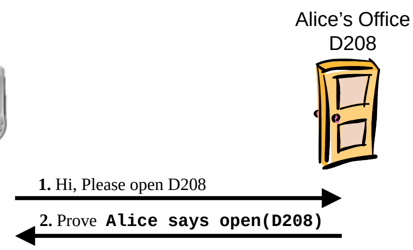
Requirement #1: Minimize Expensive Subgoals

- Prover can progress by proving different “expensive” subgoals
  - “Ask Alice for help”
  - “Ask Charlie for help”
  - “Prompt user to modify policy”
- One may be the obvious choice to a human, but prover will investigate them in the order they are found
- Requirement: aggregate choices and ask user for input
  - Avoids avenues that are unlikely to succeed
  - Increases computation – must determine all possible next steps

26

Requirement #2: Guided Credential Creation





- Prover must reason about ALL credentials that *could* be created
  - Increased computation

27

Requirement #3: Local Proving

- Requirement: only ask for help if local knowledge is insufficient to generate proof
  - I.e., may not always ask A to prove `A says F`
  - Previously, cache responses to requests
  - However, cached credentials can derive other formulas too
- Increases computation – must try to prove all formulas locally

28

Prover Must Support:

1) User-guided proving
   - User intuition helps avoid fruitless search
2) Guided credential creation
   - Prover tells user what credentials could grant access
3) Local proving
   - Only ask for help when absolutely necessary

- Straightforward implementations very inefficient
  - Prior to this work, best version took 5 *minutes* to construct the list of choices on Alice's phone

29

Talk Outline

- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
  - Requirements
  - **Algorithm**
  - Evaluation
- Identifying and Resolving Policy Misconfigurations
- Related Work and Conclusions

30

Naïve Approach (We built it: it was abysmal)

- Use Backward Chaining
  - Easier to reason about hypothetical credentials than alternatives
- Started with tactics = inference rules
- Hand-tuned to gain efficiency
  - Sacrificed some proving ability
    - Kept needing to expand tactics to cover unforeseen scenarios
  - Tedious to construct
  - Difficult to analyze formally
- Very slow: Alice's prover will take almost 5 *minutes*

31

Reasons for Complexity

- Multiple rules to apply to each formula
- No proof from local credentials → must exhaustively find all expensive subgoals
- Must reason about hypothetical credentials
  - Many different credentials *could* be created
  - Alice could delegate as **`Alice`**, **`Dept.Faculty`**, **`Alice.machine-room.staff`**, **`CMU.CA.Alice`**
  - To determine who Alice should delegate to, must finish proof with unbound variables → vastly more possibilities to explore

32

Insight

- Computation that occurs off of the critical path of access is transparent to the user
- Goal: move all possible computation off of the critical path
  - Precomputation done without knowing which resource user will access

33

Common Proving Approach: Forward Chaining

- Forward Chaining
  - Combine credentials in every way allowable by the inference rules
  - Stop if goal is derived or no inferences can be made
  - Can be run incrementally, adding one credential to KB at a time

34

Approach

- Use FC to precompute all possible facts from cached credentials
  - At time of access, simply look for precomputed proof
- At the time of access: if no proof can be found → need to identify all expensive subgoals
  - Identify who to ask for assistance, or what credentials could be created
  - Take advantage of FC results
    - If it's not in cache, don't try to derive it from credentials
  - Improvement: also precompute all possible paths of delegations
    - Tactics can then traverse a series of delegations in single step
    - Paths of delegation not expressible directly in the logic
    - Precomputed using Path Compression algorithm

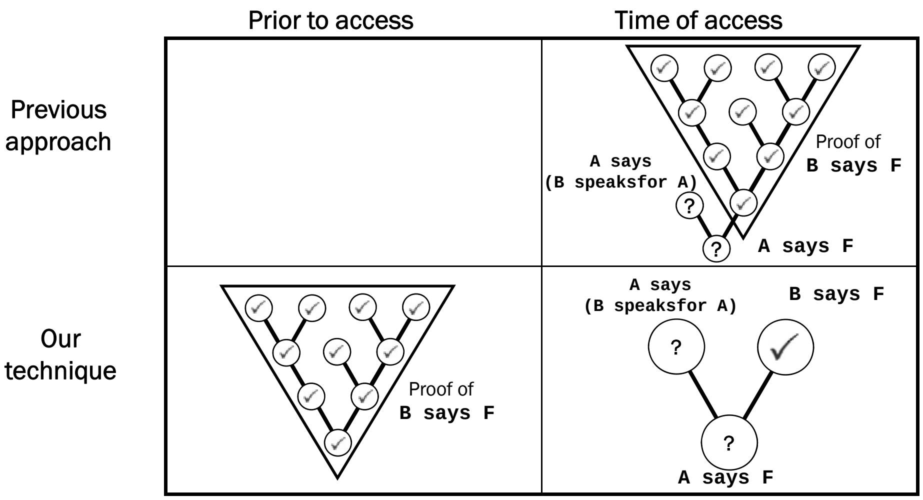
35

Identifying Expensive Subgoals
(when no proof derivable from local knowledge)

- Use backward chaining with tactics that use precomputed results
  - Tactics called LR (for left/right)
    - Each inference rule becomes two tactics: left and right
    - Left: use PC results
    - Right: use FC results
    - Systematically constructed from rules of the logic (no hand-tuning)
  - Optimized version: LR'
    - Only suggests credentials where Alice delegates as herself
    - Covers vast majority of practical scenarios

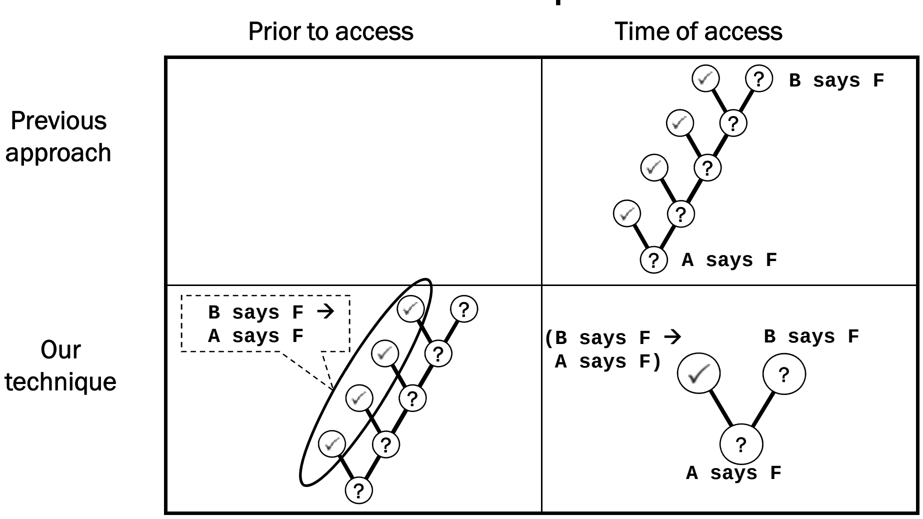
36

Right Tactics: Use Forward Chaining Results



37

Left Tactics: Use Path Compression Results



38

Algorithm Execution (Summary)

- Run before the time of access
  - Forward Chaining
  - Path Compression
- At the time of access
  - If no proof in FC results, find expensive subgoals using LR tactics
  - Use FC and PC results to avoid exploring all possibilities

39

Talk Outline

- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
  - Requirements
  - Algorithm
  - **Evaluation**
- Identifying and Resolving Policy Misconfigurations
- Related Work and Conclusions

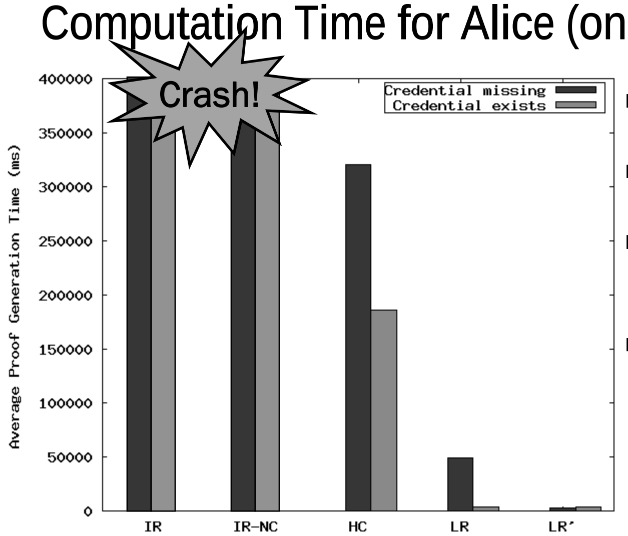
40

Analysis – Tactic Sets

- No loss in proving ability
  - If a proof can be found using IR tactics, it will be found using LR
- Primary evaluation metric: amount of work done at time of access
- Will vary the tactic sets used by Backward Chaining
- Previous work:
  - IR: inference rules
  - IR-NC: inference rules with basic cycle detection
  - HC: hand-crafted tactics
- New tactic sets (used with FC, PC):
  - LR: left-right
  - LR’: optimized LR tactics
- Platforms
  - Nokia N70 smartphone
  - Dual 2.8Ghz Xeon workstation, 1GB RAM

41

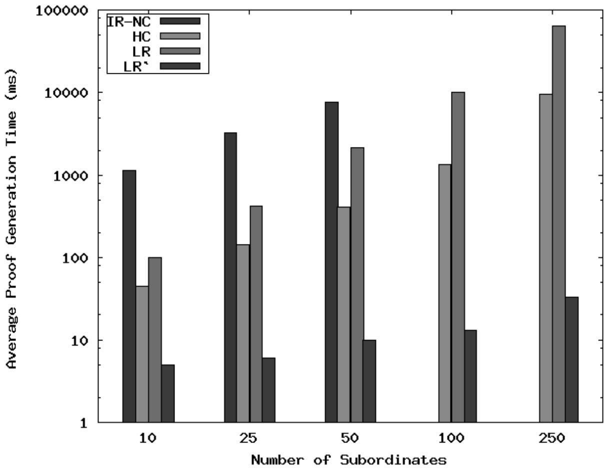
Computation Time for Alice (on phone)



- IR and IRNC crash on phone
- LR wins when credential missing
- LR wins by larger margin when proof completable
- LR’ dominates

42

Large Policies (on workstation, credential missing)



- Complexity indicative of university-wide policy
- IR crashes
- IR-NC crashes for sizes > 50
- HC only finds 21% of possible solutions
- LR finds all solutions
- LR' finds all solutions in the subset it is looking for

43

Tabling

- Tabling eliminates redundancy in proof search by internally caching results
- Similar to our use of FC, but no precomputation
- Compare IR vs LR (both with tabling)
- Restrict depth of nested names used by IR to prevent infinite expansion

- Findings:
  - LR much faster for small policies with tabling enabled
  - LR faster if IR allowed 2 or more degrees of nesting on larger policies

44

Effects of Precomputing Results

- Cache size
  - Must all fit in memory
  - Linear w.r.t. the number of credentials
- Total precomputation time
  - Amortized over many accesses
  - Quadratic w.r.t. the number of credentials
- Centralized prover can handle 1000's, but not 10,000's of users
  - Distributed caches likely to be much smaller

45

Checkpoint

- Distributed proof construction
  - Reduces queries by constructing proof on node with most relevant knowledge
  - Enables each node a degree of flexibility in how proof is constructed
- Present an efficient proof construction strategy that supports
  - User guided proving
  - Credential creation
  - Local proving
- Computation time reduced by utilizing pre-computed results to avoid exploring costly branches at the time of access
  - Forward Chaining: computes all true facts
  - Path Compression: computes all trust relationships
  - LR Tactics: systematically constructed to leverage above results
- Our strategy is efficient enough to be practical
  - Has been deployed for two years

46

Talk Outline

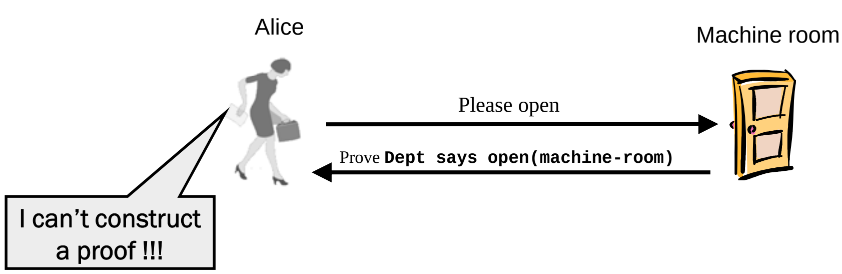
- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
- **Identifying and Resolving Policy Misconfigurations**
  - Identifying: techniques and evaluation
  - Resolving: techniques and evaluation
- Related Work and Conclusions

47

A Misconfigured Access-Control Policy

- Alice is a professor
- Professors should be allowed access to the machine room
- However, Alice has never needed physical access to her machines

- It is 3am, and Alice learns that the machine room is overheating



48

Policy Misconfigurations

- An incorrectly specified policy can:
  1. Allow an access that should be denied
  2. Deny an access that should be allowed

- Generally, access-control systems focus on preventing #1
- However, #2 can be bad also
  - Very annoying to users
  - Can have serious consequences if timely access is critical
    - Servers overheating in machine room
  - Conjecture: denying legitimate access is so undesirable that many systems are (inadvertently) configured with overly permissive policy

- In this talk, misconfiguration is a mistake in policy that results in #2

49

Types of Policy

- *Implemented policy:* accesses that are permitted by the system
- *Intended policy*: accesses that "should" be allowed

- Inconsistencies between implemented and intended policy are misconfigurations
- Can occur because:
  - New users or resources
  - Organizational changes
  - Policy not fully specified at the outset

50

Objective

- Ideally, would like to correctly configure policy before Alice attempts to gain access to the machine room
  - So Alice can immediately gain access
- Mechanism involves two steps:
  - Identifying misconfigurations
  - Resolving misconfigurations
- Construct a mechanism that is independent of policy-specification language
  - Specifically, use only access logs

51

Talk Outline

- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
- Identifying and Resolving Policy Misconfigurations
  - **Identifying: techniques and evaluation**
  - Resolving: techniques and evaluation
- Conclusions

52

Identifying Misconfigurations

- Observation: access-control policy exhibits patterns
  - Inconsistencies in these patterns can indicate misconfigurations
  - These patterns are observable from access-control logs
  - Need centralized collection of logs to analyze
- Use Association Rule Mining [Agrawal and Srikant ’94]
  - Input: series of records characterized by a fixed number of attributes
    - E.g., record is a shopping cart, attributes describe contents
  - Output: rules (or statistical patterns)
    - People who buy peanut butter and jelly usually buy bread
- Use rules to identify anomalies
    - Alice bought peanut butter and jelly – did she forget bread?

Data Representation

	AttA	AttB	AttC	AttD
Record1	T	-	T	-

Constructing Rules

	AttA	AttB	AttC	AttD
Record1	T	-	T	-
Record2	T	T	-	-
Record3	T	T	T	-
Record4	T	-	T	T

Rule: A → B Confidence = 0.5

Rule: A → C Confidence = 0.75

55

Identifying Misconfigurations

	ResA	ResB	ResC	ResD
Alice	T	-	T	-
Bob	T	T	-	-
Charlie	T	T	T	-
David	T	-	T	T

Resources

Users

Rule: ResA → ResC

Potential Misconfiguration
(a.k.a., a prediction)

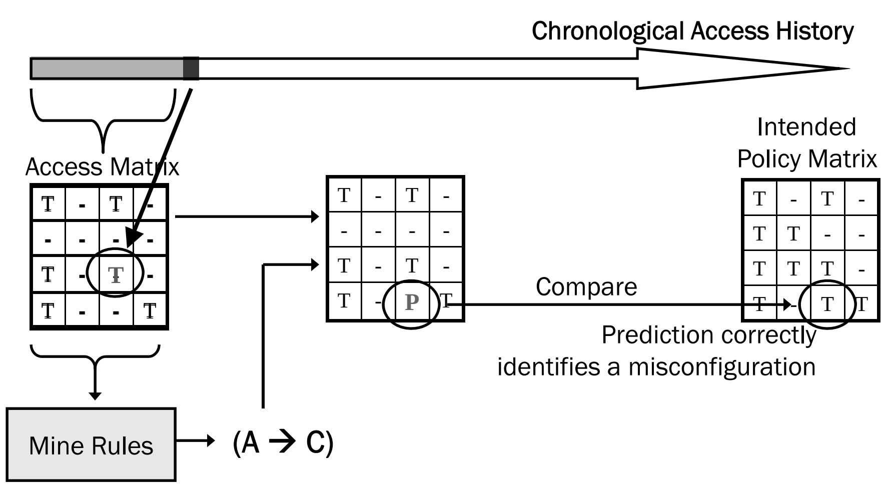
56

Dataset

- Log of 10,911 accesses drawn from Grey deployment
- Spans 16 months
- Contains accesses by 25 users to 29 resources

- Policy matrix: indicates what accesses should be allowed under implemented and intended policy
  - Implemented: accesses that occurred in log
    - assumes no unauthorized access
  - Intended: surveyed users

Identification Simulation

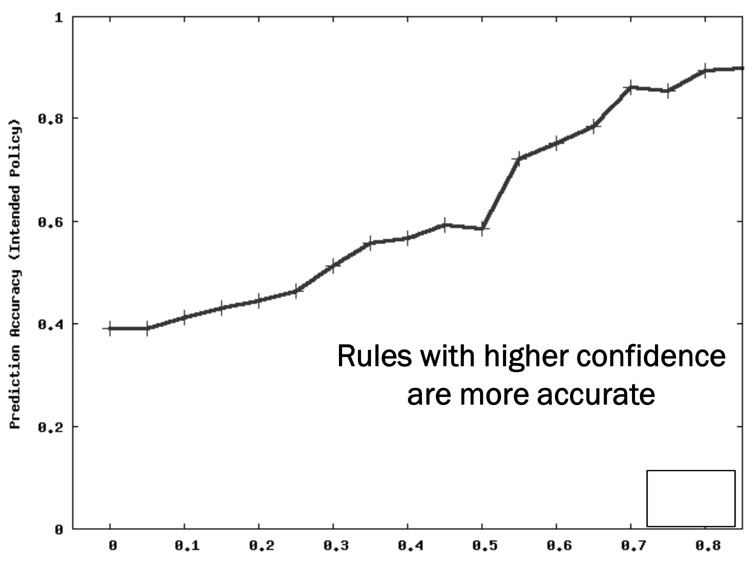


Identification Metrics

- Two measures of success:
  - Accuracy: what percentage of predictions are "correct"
  - Coverage: what percentage of misconfigurations are predicted
- We measure accuracy and coverage versus intended policy
  - Results for implemented policy in paper
- Parameter: *minconf*
  - Only predict using rules with confidence > *minconf*

59

Prediction Accuracy (Intended Policy)



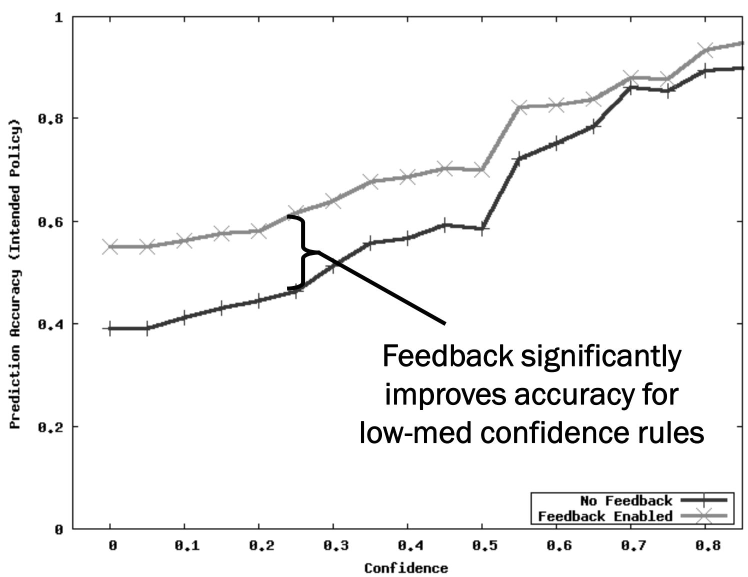
Feedback

- Some mined rules are not a good indicator of policy
- Use feedback to prevent these rules from repeatedly making incorrect predictions
  - Incorrect prediction: decrease score
  - Correct prediction: increase score
- If score of rule falls below a threshold, stop using rule

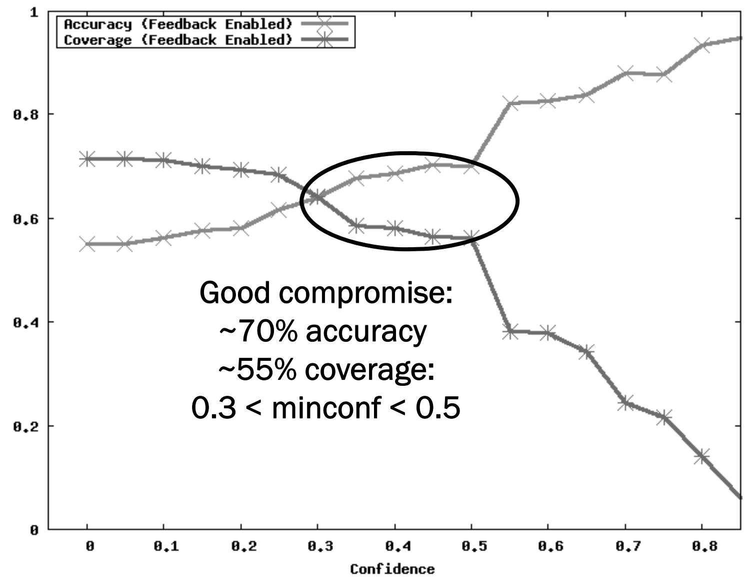
- Score each premise of a rule independently
  - Allows us to quickly prune groups of similar rules
  - For details, see paper

61

Prediction Accuracy (Intended Policy)



Accuracy/Coverage Tradeoff (with Feedback)



Talk Outline

- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
- Identifying and Resolving Policy Misconfigurations
  - Identifying: techniques and evaluation
  - **Resolving: techniques and evaluation**
- Conclusions

64

Resolving Misconfigurations Proactively

- Once a misconfiguration is identified, a human must determine if it should be repaired – but which human?
- Want to figure out without bothering the user that might eventually need access
- Single administrator → trivial
- Distributed administration → multiple users can repair
  - Asking all of them is very annoying
- Strategy: use logs to determine how similar misconfigurations were resolved in the past
  - Logs tell us who users asked to help resolve misconfigurations

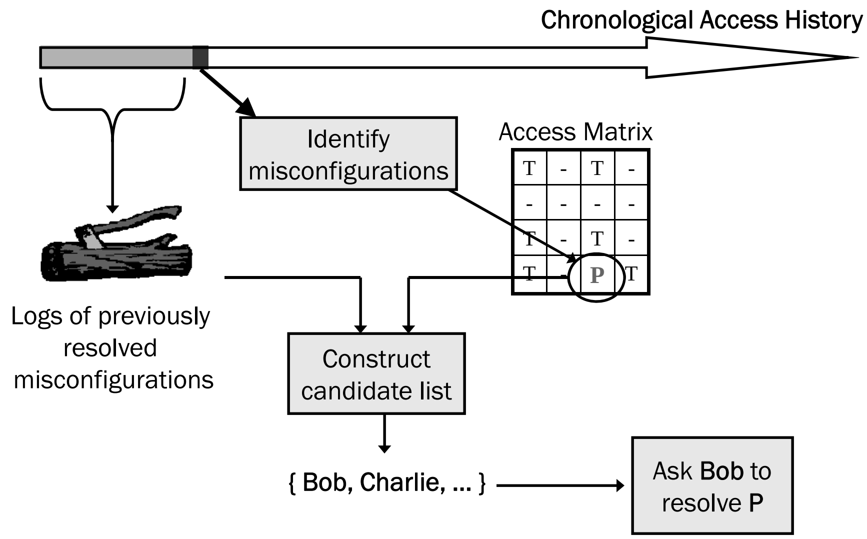
65

Strategies for Directing Resolution Requests

- Construct a candidate list of users based on previous history
- Sort candidate list by the number of times they helped
- Strategies we evaluated for constructing candidate list:
  - OU: who helped when Other Users accessed this resource
  - OR: who helped when this user accessed Other Resources
  - OU + OR: Union of OU and OR
  - OU + OR + PPA: Union plus Principals who Previously Accessed this resource

66

Resolution Simulation



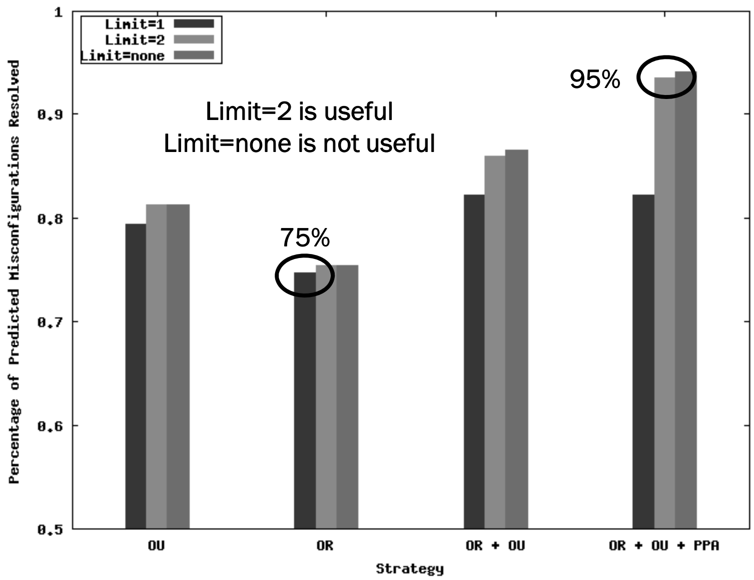
67

Resolution Metrics

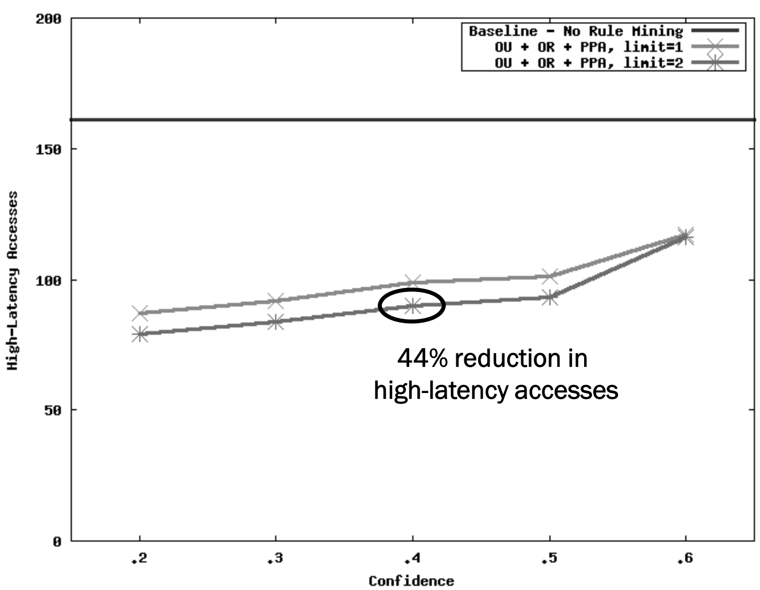
- **Success rate**: % of correctly predicted misconfigurations that can be repaired
  - Essentially, how often we ask the right person
  - Identifying a misconfiguration is only useful if corrected
- **High-latency accesses**: # of misconfigurations repaired at the time of access (fewer = better)
  - Each time, one user must be interrupted, and another user must wait
- **Total user interaction time**: rough approximation of the amount of time all users spend interacting with the system

68

Resolution Success Rate



High-Latency Accesses



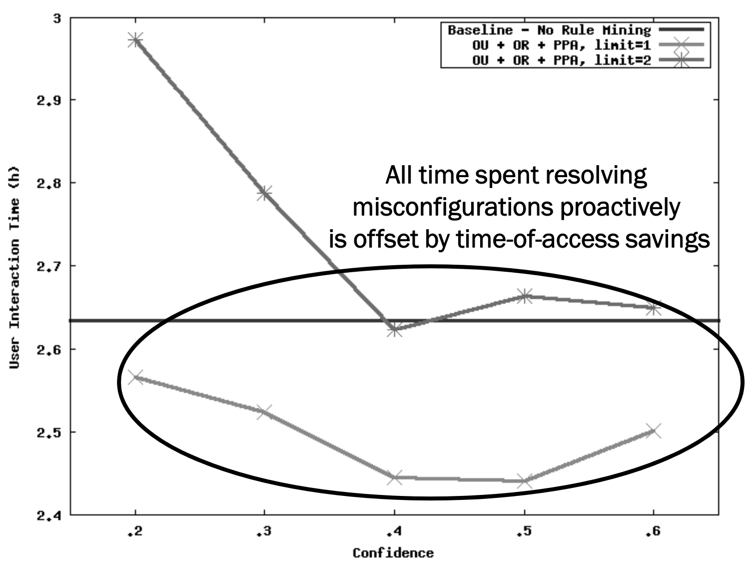
Total User Interaction Time

- VERY rough approximation
- Factors in time spent resolving predictions proactively and time saved by reducing high-latency accesses

- Times measured from our deployment, applied to simulated events
- Places where user interaction is required:
  - Time a user waits while misconfiguration repaired
    - Ranged between 25 seconds and 18.6 hours
    - Median = 98 seconds
  - Time user spends repairing misconfiguration
    - Avg = 23 seconds

71

Total User Interaction Time (across all users over 16 months)



Partial Data

- All data may not always be available
  - Users may opt out for privacy concerns
  - Some resources might not be logged

- How well do our techniques work when some data is missing?
- Grouped users by activity, removed data for one set at a time

- Findings:
  - Withholding data for users primarily impacts those users

73

Checkpoint

- Distributed proof construction
  - Reduces queries by constructing proof on node with most relevant knowledge
  - Enables each node a degree of flexibility in how proof is constructed

- Efficient proving in practical scenarios
  - Dramatically more efficient on practical policies
  - Enables misconfigurations to be resolved at the time of access

- Identifying and resolving policy misconfigurations
  - Policy misconfigurations can deny legitimate access
    - This is highly annoying to users
    - Can have severe consequences in some scenarios (e.g., overheating machine room)

  - For reasonable parameters, on our dataset we can simultaneously
    - Identify ~55% of misconfigurations w.r.t. intended policy
    - Eliminate ~45% of misconfigurations that would delay an access
    - Reduce total user interaction time

  - Techniques work well with portions of the dataset removed

74

Talk Outline

- Introduction
- Distributed Proof Construction
- Efficient Proving for Practical Systems
- Identifying and Resolving Policy Misconfigurations
- **Conclusions**

75

Summary

- Constructing proofs is difficult in a distributed access-control system using formal logic
  - Credentials are distributed
  - Credentials may be created dynamically
  - Must consider human factors
- We present a practical suite of proof-construction techniques
  - Distributed proving: reduces requests for assistance
  - Efficient proving: reduces computation and incorporates user interaction
  - Resolving policy misconfigurations: reduces costly time-of-access delays

76

Bibliography

- L. Bauer, S. G. Garriss and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, May 2005.
- L. Bauer, S. G. Garriss and M. K. Reiter. Efficient proving for practical distributed access-control systems. In *Computer Security – ESORICS 2007: 12th European Symposium on Research in Computer Security* (Lecture Notes in Computer Science 4734), pages 19–37, September 2007.
- L. Bauer, S. G. Garriss and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 185–194, June 2008.

77