Specification and Analysis of Attribute-based Authorization Policy

William H. Winsborough
Center for Secure Information Systems
George Mason University

Joint work with:

Ninghui Li, Purdue University
John C. Mitchell, Stanford University

Attribute-based Authorization Policy

- The Big Goal
  - Flexible, scalable authorization for decentralized, collaborative environments and open systems
- The Approach
  - Authorization decision is based on attributes of resource requestor
  - Policy language based on logic programming supports key trust management needs
  - Credentials are signed policy statements about attributes of principals & rules for deriving same
  - Provide policy-understanding support

Outline: Problems We Address

- Need a language for authorization policy to support collaboration in open systems
  - *RT*: A Role-based Trust-management* framework
- Need techniques for understanding and managing policy
  - Safety and availability analysis in trust management*

* "Trust management" was coined by Blaze, Feigenbaum, and Lacy to describe a collection of desiderata for decentralized authorization systems.

Language for Policy and Credentials

- Pubs
  - Design of a Role-Based Trust Management Framework. Ninghui Li, John C. Mitchell, and William H. Winsborough. *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002
- Outline
  - Requirements
  - Examples
  - Syntax
  - Semantics
  - Language extensions

Policy Language Wish List

- Decentralize authority to define attributes
  - Utilize policy and credentials from many sources
- Delegation of attribute authority
  - To specific principals
  - To principals with certain attributes
- Inference of attributes
  - E.g., derive access rights based on roles or other characteristics
- Intersection of attributes
- Parameterization
- Support for thresholds, separation of duty

Role-based Trust Management (*RT*)

- A family of credential / policy languages
  - Simplest, RT_0, has no parameterization, thresholds, or separation of duty
- RT_0 example: student discount subscription
  - EPub.studentDiscount ← StateU.student
  - StateU.student ← URegistrar.fulltimeLoad
  - StateU.student ← URegistrar.parttimeLoad
  - URegistrar.parttimeLoad ← Alice

Role-based Trust Management (*RT*)

- A family of credential / policy languages
  - Simplest, RT_0, has no parameterization, thresholds, or separation of duty
- RT_0 example: student discount subscription
  - EPub.studentDiscount ← StateU.student
  - StateU.student ← URegistrar.fulltimeLoad
  - StateU.student ← URegistrar.parttimeLoad
  - URegistrar.parttimeLoad ← Alice
- Credential chain proves authorization

Example: Attribute-based Delegation

- Accepting student ID from any university
  - EPub.studentDiscount ← FAB.accredited.student
  - FAB.accredited ← StateU
  - StateU.student ← URegistrar.fulltimeLoad
  - StateU.student ← URegistrar.parttimeLoad
  - URegistrar.parttimeLoad ← Alice

Example: Expressivity in Credentials

- Deferring a Guaranteed Student Loan
  - BankWon.deferGSL ← FAB.accredited.fulltimeStudent
  - FAB.accredited ← StateU
  - StateU.fulltimeStudent ← URegistrar.fulltimeLoad
  - StateU.fulltimeStudent ← URegistrar.parttimeLoad ∩ StateU.gradOfficer.phdCandidate
  - URegistrar.parttimeLoad ← Bob
  - StateU.gradOfficer ← Carol
  - Carol.phdCandidate ← Bob

© William H. Winsborough

RT_0 Syntax

- Basic structure is a role (i.e., an attribute): A.r
  - A is an principal (authority for A.r), r is a role name
- Four types of policy statement
  - A.r ← D
    Role A.r contains principal D as a member
  - A.r ← $B.r_1$
    A.r contains role $B.r_1$ as a subset
  - A.r ← $A.r_1.r_2$
    A.r contains $B.r_2$ as a subset, for each B in $A.r_1$
  - A.r ← $A_1.r_1 \cap A_2.r_2$
    A.r contains the intersection
- A credential is a statement signed by A, the credential issuer and the authority over A.r
- The first 3 statement types give a language equivalent to pure SDSI

© William H. Winsborough

A Brief Intro to Logic Programming

- A program *P* is a set of clauses:
  - $h(\boldsymbol{t}_0)$:- $b_1(\boldsymbol{t}_1)$, ..., $b_n(\boldsymbol{t}_n)$ where *h* and b_i are predicates and $\boldsymbol{t}_i$ are tuples of logical terms
    - ":-" is read "if"
  - p(c, ?X) :- q(b, ?Z), r(?Z, ?X).
  - q(b, a).
  - r(a, d).
- A query $\mathcal{Q}$ has the form ?- $b_1(\boldsymbol{t}_1)$, ..., $b_n(\boldsymbol{t}_n)$
  - ?- p(?U, ?V).
- An answer is an instance $\mathcal{Q}'$ of the query $\mathcal{Q}$ that is logically entailed by the program ($\mathcal{P} \models \mathcal{Q}'$) , e.g., $\mathcal{Q}'$ = p(c, d).

Benefits of LP Semantics

- Makes complexity results easy
- Facilitates extending RT_o
  - Parameters, thresholds, sep. of duty
  - Other semantic foundations do not easily support important extensions
    - String rewriting [Clarke et al., JCS 2001]
    - Sets provide a good intuition
      - A role is the set of principals in the role
      - Parameterization requires generalization
  - With LP semantics, extension is easy

$SP(\mathcal{P})$: A Logic-Programming Semantics for RT_0 policy $\mathcal{P}$

- Translate each statement of $\mathcal{P}$ to a clause:
  - For each $A.r \leftarrow D$ in $\mathcal{P}$, add
    m(A, r, D).
  - For each $A.r \leftarrow B.r_1$ in $\mathcal{P}$, add
    $m(A, r, ?X) \text{ :- } m(B, r_1, ?X).$
  - For each $A.r \leftarrow A.r_1.r_2$ in $\mathcal{P}$, add
    $m(A, r, ?X) \text{ :- } m(A, r_1, ?Y), m(?Y, r_2, ?X).$
  - For each $A.r \leftarrow A_1.r_1 \cap A_2.r_2$ in $\mathcal{P}$, add
    $m(A, r, ?X) \text{ :- } m(A_1, r_1, ?X), m(A_2, r_2, ?X).$

Globally Unique Role Names

- Application Domain Specification Document (ADSD)
  - Declares a collection of related role names
  - Unique name space for each ADSD
  - Role names declared in different ADSDs are different
  - They refer to the URI of the ADSD in which they are declared
- In RT_1, where roles are parameterized, ADSD also gives type signature

RT_1: Adding Role Parameters

- Roles have the form A.R = A.r(h_1, …, h_n)
- Each h_i is a data term whose type is that declared for r's i^{th} parameter in the ADSD
- Example:
  - BigCorp.evaluatorOf(?Y) ← BigCorp.managerOf(?Y)
  - BigCorp.raise ← BigCorp.evaluatorOf(this).exceedsExpectations

Parameterization: Semantics and Complexity

- LP semantics simply adds several m's of different arity
  - E.g., A.r(h_1, …, h_n) ← B.r_1(s_1, …, s_m) translates to m(A, r, h_1, …, h_n, ?X) :- m(B, r_1, s_1, …, s_m, ?X)
- Apply known complexity results: The atomic implications of $SP(\mathcal{P})$ can be computed in $O(N^{v+3})$
  - v is the max number of variables per statement
  - Each role name has a most p arguments
  - $N = \max(N_0, pN_0)$
  - N_0 is the number of statements in $\mathcal{P}$

Further LP Advantage

- Can further extend to efficiently support simple constraint domains
  - Datalog with Constraints: A Foundation for Trust Management Languages. Ninghui Li and John C. Mitchell. Fifth International Symp. on Practical Aspects of Declarative Languages (PADL), Jan 2003

RT^T: Supporting Threshold and Separation-of-Duty

- Threshold: require agreement among *k* principals drawn from a given list
- SoD: e.g., purchase requires approval by buyer and manager
  - Want to achieve SoD without mutual exclusion, which is nonmonotonic
- Though related, neither subsumes the other
- RT^T introduces a primitive that supports both: manifold roles
- RT^T can be combined with either RT_0 or RT_1, yielding RT_0^T and RT_1^T, respectively

Manifold Roles

- While a standard role is a set of principals, a manifold role is a set of sets of principals
- A set of principals that together occupy a manifold role can collectively exercise privileges of that role
- Two operators: $\odot$, $\otimes$
  - $A.R_1 \otimes B.R_2$ contains sets of two distinct principals, one a member of $A.R_1$, the other of $B.R_2$
  - $A.R_1 \odot B.R_2$ does not require them to be distinct
  - gradSchool.docCommittee(?s) ←
    gradSchool.docAdvisor(?s) ⊗
    gradSchool.commMember(?s) ⊗
    gradSchool.commMember(?s) ⊗
    gradSchool.commMember(?s) ⊗
    gradSchool.externCommMember(?s)

RT^T Syntax and Complexity

- Manifold roles can be used in basic *RT* statements
- Also add two new types of policy statement
  - $A.R \leftarrow A_1.R_1 \odot A_2.R_2 \odot \ldots \odot A_k.R_k$
    - $\text{members}(A.R) \supseteq \text{members}(A_1.R_1 \odot A_2.R_2 \odot \ldots \odot A_k.R_k) = \{s_1 \cup \ldots \cup s_k \mid s_i \in \text{members}(A_i.R_i) \text{ for } 1 \le i \le k\}$
  - $A.R \leftarrow A_1.R_1 \otimes A_2.R_2 \otimes \ldots \otimes A_k.R_k$
    - $\text{members}(A.R) \supseteq \text{members}(A_1.R_1 \otimes A_2.R_2 \otimes \ldots \otimes A_k.R_k) = \{s_1 \cup \ldots \cup s_k \mid (s_i \in \text{members}(A_i.R_i)\ \&\ s_i \cap s_j \neq \emptyset) \text{ for } 1 \le i \neq j \le k\}$
- ADSD must declare a *size* for each manifold role
- Given a set $\mathcal{P}$ of *RT*^{T} statements, let *t* be the maximal size of all roles in $\mathcal{P}$. The atomic implications of $\mathcal{P}$ can be computed in time $O(MN^{v+2t})$.

Distributed Credential Chain Discovery

Credential Availability and Light-weight Evaluation

Distributed Credential Chain Discovery

- Pubs
  - Distributed Credential Chain Discovery in Trust Management. Ninghui Li, William H. Winsborough, and John C. Mitchell
    - *Journal of Computer Security,* 11(1):35-86, February 2003
- Outline
  - Sound and complete evaluation model for RT_0
  - Efficient search for proof of authorization
  - Support for distributed discovery

Algorithmic Contributions

- Search algorithms:
  - Worst case efficiency as good as any existing algorithm
    - Forward. $O(N^3)$ time, N = number of credentials
    - Backward. $O(N^2M)$ time, M = sum of credential sizes
    - Both directions. $O(N^2M)$ time
  - Well suited to the application
    - Efficient when there are lots of unrelated credentials
    - Changes to credential pool do not degrade performance
    - Graph search can drive credential discovery

Prior Work on Evaluation

- All present at least one of the following problems for discovery:
  - Some inherently require credential to be centralized
    - E.g., SDSI evaluation [Clarke et al. 2001]
  - Evaluation doesn't naturally drive collection process
    - E.g., Delegation Logic [Li 2000]
  - Evaluation drives chain collection in only one direction or the other, but not both
    - E.g., QCM [Gunter & Jim 2000] and SD3 [Jim 2001]
    - Can't store some credentials with issuer and some with subject

Example: Student ACM Discount

- EPub.studentACM ← EOrg.student ∩ ACM.member
- EOrg.student ← EOrg.university.student
- EOrg.university ← FAB.accredited

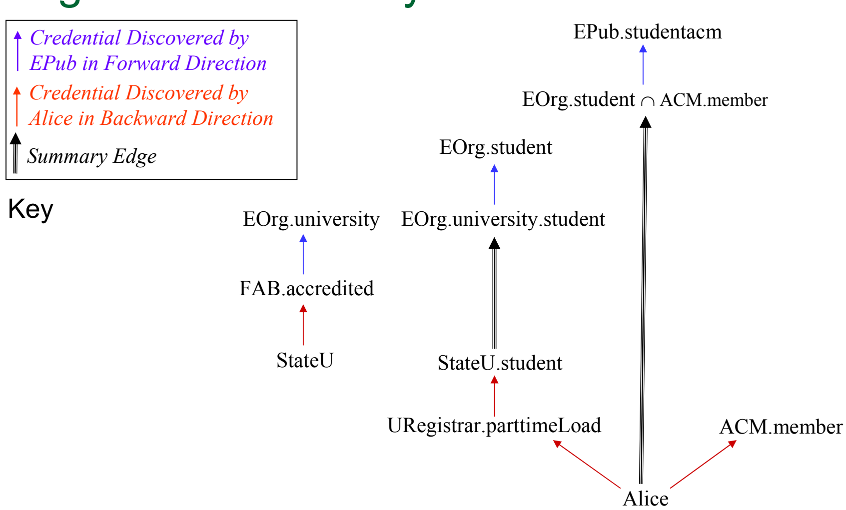
Credential Discovered in Forward Direction

- FAB.accredited ← StateU
- StateU.student ← URegistrar.parttimeLoad
- URegistrar.parttimeLoad ← Alice
- ACM.member ← Alice

Credential Discovered in Backward Direction

Credential Graph Organizes Discovery

EPub gives a double subscription discount



Storage Type System

- Storage type of role name determines where credential is stored: with issuer or with subject
- Well-typing ensures credentials are stored where they can be found by tracing the credential graph

Credentials	Attribute Name	Type	Credential Stored by
1) EPub.studentDiscount ← StateU.student	studentDiscount	backward-traceable	EPub
2) StateU.student ← URegistrar.parttimeLoad	student	forward-traceable	URegistrar
3) URegistrar.parttimeLoad ← Alice	parttimeLoad	forward-traceable	Alice

Security Analysis

Understanding and Managing Authorization Policy

Motivation: A Higher Vantage Point

- Authors of policy statements need assistance in understanding global impact of delegations, revocations
- Who could get access to what? (Safety)
  - Assessing exposure
- Who could be denied? (Availability)
  - Ensuring applications have authorizations needed for correct operation

Pubs and Outline

- Pubs
  - Beyond Proof-of-compliance: Safety and Availability Analysis in Trust Management. Ninghui Li, William H. Winsborough, and John C. Mitchell. *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003
- Outline
  - Abstract security analysis problem
  - Instantiating the analysis problem for *RT*
  - Usage scenarios
  - Solving simple analysis problems
  - Complexity of other analysis problems
  - Future work

Reachable Policy States

- An individual or organization normally controls only a portion of the global policy state
  - Other statements may be added or removed
  - Analysis factors in those potential future changes
- *Restriction rule* $\mathcal{R}$ defines how state $\mathcal{P}$ may be changed to $\mathcal{P}'$ ($\mathcal{P} \mapsto_{\mathcal{R}} \mathcal{P}'$)
- Existential analysis problem
  - Does there exist $\mathcal{P}'$ such that $\mathcal{P} \mapsto_{\mathcal{R}} \mathcal{P}'$ and $\mathcal{P}' \vdash \mathcal{Q}$?
- Universal analysis problem
  - For every $\mathcal{P}'$ such that $\mathcal{P} \mapsto_{\mathcal{R}} \mathcal{P}'$, does $\mathcal{P}' \vdash \mathcal{Q}$?

Example Analysis Problem Instances

- "Can Alice ever get access to the database?"
  - Simple Safety -- Existential
- "Will Bob always have access to the database?"
  - Simple Availability -- Universal
- "Can anyone besides you and me ever get access?"
  - Bounded Safety -- Universal
- "Will there always be somebody that has access?"
  - Liveness -- Existential
- "Can anyone ever be both a buyer and an accountant?"
  - Mutual Exclusion -- Universal
- "Will all managers always have access?"
  - Containment: Availability -- Universal
- "Can anyone who is not an employee ever get access?"
  - Containment: Safety -- Universal

Instantiating the Analysis

- Language used to express $\mathcal{P}$
- Form of restriction rule $\mathcal{R}$
- Form of query $\mathcal{Q}$

Policy Language and Restriction Rule

- $\mathcal{P}$ is an RT_0 policy
- $\mathcal{R}$ gives two sets of roles, $\mathcal{G}$ and $\mathcal{S}$
  - Growth restriction: additional statements defining roles in $\mathcal{G}$ cannot be added to state
  - Shrink restriction: statements defining roles in $\mathcal{S}$ cannot be removed from state

Three Forms of Query

- Membership: A.r ⊒ { D_1, ..., D_n }
- Boundedness: { D_1, ..., D_n } ⊒ A.r
- Inclusion: X.u ⊒ A.r
  - Formally, $\mathcal{P} \vdash$ X.u ⊒ A.r if and only if
    { Z | *SP*($\mathcal{P}$) ⊨ m(X, u, Z) } ⊇
    { Z | *SP*($\mathcal{P}$) ⊨ m(A, r, Z) }

© William H. Winsborough

Example P and R

- SA.access ← HR.manager
- SA.access ← HR.manager.access ∩ HR.employee
- HR.employee ← HR.manager
- HR.employee ← HR.programmer
- HR.manager ← Alice
- HR.programmer ← Bob
- HR.programmer ← Carl
- Alice.access ← Bob
- $\mathcal{G}$ = { SA.access, HR.employee }
- $\mathcal{S}$ = { SA.access, HR.employee, HR.manager }

© William H. Winsborough

Example Problem Instance (1 of 4)

- SA.access ← HR.manager
- SA.access ← HR.manager.access ∩ HR.employee
- HR.employee ← HR.manager
- HR.employee ← HR.programmer
- HR.manager ← Alice
- HR.programmer ← Bob
- HR.programmer ← Carl
- Alice.access ← Bob
- $\mathcal{G}$ = { SA.access, HR.employee }
- $\mathcal{S}$ = { SA.access, HR.employee, HR.manager }
- Simple safety: Is SA.access ⊒ { Eve } possible? (Yes)

Example Problem Instance (2 of 4)

- SA.access ← HR.manager
- SA.access ← HR.manager.access ∩ HR.employee
- HR.employee ← HR.manager
- HR.employee ← HR.programmer
- HR.manager ← Alice
- HR.programmer ← Bob
- HR.programmer ← Carl
- Alice.access ← Bob
- $\mathcal{G}$ = { SA.access, HR.employee }
- $\mathcal{S}$ = { SA.access, HR.employee, HR.manager }
- Simple availability: Is SA.access ⊒ { Alice } necessary? (Yes)

Example Problem Instance (3 of 4)

- SA.access ← HR.manager
- SA.access ← HR.manager.access ∩ HR.employee
- HR.employee ← HR.manager
- HR.employee ← HR.programmer
- HR.manager ← Alice
- HR.programmer ← Bob
- HR.programmer ← Carl
- Alice.access ← Bob
- $\mathcal{G}$ = { SA.access, HR.employee }
- $\mathcal{S}$ = { SA.access, HR.employee, HR.manager }
- Bounded safety: Is { Alice, Bob } ⊒ SA.access necessary? (No)

Example Problem Instance (4 of 4)

- SA.access ← HR.manager
- SA.access ← HR.manager.access ∩ HR.employee
- HR.employee ← HR.manager
- HR.employee ← HR.programmer
- HR.manager ← Alice
- HR.programmer ← Bob
- HR.programmer ← Carl
- Alice.access ← Bob
- $\mathcal{G}$ = { SA.access, HR.employee }
- $\mathcal{S}$ = { SA.access, HR.employee, HR.manager }
- Containment: Is HR.employee ⊒ SA.access necessary? (Yes)

Security Analysis: Usage Cases

- Security requirement = analysis problem instance + acceptable answer
  - Organization defines a set of requirements
- Sanity check
  - Some principals are trusted
  - They analyze proposed policy changes with respect organization's requirements before committing
- Insider threat assessment
  - Can vary the principals that are trusted by changing the restriction rule
  - In this way, organization can determine how it is exposed to the principals

Membership and Boundedness Queries

- Efficient algorithms based on two non-standard LP semantics
  - $LB(\mathcal{P}, \mathcal{R})$
  - $UB(\mathcal{P}, \mathcal{R})$
- Solves 4 analysis problems:

	$\forall$	$\exists$
Membership	*LB*	*UB*
Boundedness	*UB*	*LB*

$LB(\mathcal{P}, \mathcal{R})$: Lower Bound Program

- Construct $\mathcal{P}|_{\mathcal{R}}$ from $\mathcal{P}$ by dropping all statements defining roles not in $\mathcal{S}$
- Construct $LB(\mathcal{P}, \mathcal{R})$ from $\mathcal{P}$:
  - For each A.r ← D in $\mathcal{P}|_{\mathcal{R}}$ add lb(A, r, D).
  - For each A.r ← B.r_1 in $\mathcal{P}|_{\mathcal{R}}$, add lb(A, r, ?Z) :- lb(B, r_1, ?Z).
  - For each A.r ← A.r_1.r_2 in $\mathcal{P}|_{\mathcal{R}}$, add lb(A, r, ?Z) :- lb(A, r_1, ?Y), lb(?Y, r_2, ?Z).
  - For each A.r ← A_1.r_1 ∩ A_2.r_2 in $\mathcal{P}|_{\mathcal{R}}$, add lb(A, r, ?Z) :- lb(A_1, r_1, ?Z), lb(A_2, r_2, ?Z).

$LB(\mathcal{P}, \mathcal{R})$

- Lower Bound Program handles:
  - Universal membership analysis
    A.r ⊒ { D_1, …, D_n } is necessary iff
    $LB(\mathcal{P}, \mathcal{R}) \models$ lb(A, r, D_i) for each $i \in [1..n]$
  - Existential boundedness analysis
    { D_1, …, D_n } ⊒ A.r is possible iff
    { D_1, …, D_n } ⊇ { Z | $LB(\mathcal{P}, \mathcal{R}) \models$ lb(A, r, Z) }

$UB(\mathcal{P}, \mathcal{R})$: Upper Bound Program

- Construct $UB(\mathcal{P}, \mathcal{R})$ from $\mathcal{P}$:
  - Add ub(⊤, ?r, ?Z).
  - For each A.r ∈ Roles($\mathcal{P}$) – $\mathcal{G}$ add ub(A, r, ?Z).
  - For each A.r ← D in $\mathcal{P}$, add ub(A, r, D).
  - For each A.r ← B.r_1 in $\mathcal{P}$, add
    ub(A, r, ?Z) :- up(B, r_1, ?Z).
  - For each A.r ← A.r_1.r_2 in $\mathcal{P}$, add
    ub(A, r, ?Z) :- ub(A, r_1, ?Y), ub(?Y, r_2, ?Z).
  - For each A.r ← A_1.r_1 ∩ A_2.r_2 in $\mathcal{P}$, add
    ub(A, r, ?Z) :- ub(A_1, r_1, ?Z), ub(A_2, r_2, ?Z).

$UB(\mathcal{P}, \mathcal{R})$

- Upper Bound Program handles:
  - Existential membership analysis
    A.r ⊒ { D_1, …, D_n } is possible iff
    - A.r ∉ $\mathcal{G}$
    - $UB(\mathcal{P}, \mathcal{R})$ ⊨ ub(A, r, ⊤), or
    - $UB(\mathcal{P}, \mathcal{R})$ ⊨ ub(A, r, D_i) for each $i \in [1..n]$

    Cf. HRU model of safety, which is undecidable
  - Universal boundedness analysis
    { D_1, …, D_n } ⊒ A.r is necessary iff
    { D_1, …, D_n } ⊇ { Z | $UB(\mathcal{P}, \mathcal{R})$ ⊨ ub(A, r, Z) }

Inclusion Complexity Depends on RT_0 Sublanguage

- We consider four subsets of RT_0
  - *RT*[ ] has only facts & simple delegation
    - $A.r \leftarrow D$
    - $A.r \leftarrow B.r_1$
  - *RT*[↞] = *RT*[ ] + linking
    - $A.r \leftarrow A.r_1.r_2$
  - *RT*[∩] = *RT*[ ] + intersection
    - $A.r \leftarrow A_1.r_1 \cap A_2.r_2$
  - *RT*[↞, ∩] = RT_0

Complexity of Inclusion Queries

- Polynomial algorithms for *RT*[ ]
- Complexity results
  - *RT*[↞] : **PSPACE**-complete
  - *RT*[∩] : **coNP**-complete
  - *RT*[↞, ∩] : in **coNEXP**

Possible Future Work: A Security Policy Management Assistant

- Assistant should automatically generate proposals for how to guarantee security requirements are met
  - Needed:
    - When requirements change
    - When you change whom you trust
  - Assistant should explain why some requirements cannot be met
- Assistant should help assess insider threat
  - Which semi-trusted parties could really hurt you?
  - Assess your exposure to colluding groups of insiders
  - Assistant should suggest ways to reduce your exposure, e.g. through separation of duties
- Heuristical analysis for expensive queries

© William H. Winsborough

Summary: Problems We Have Addressed

- Provided a language for authorization policy to support collaboration in open systems
  - *RT*: A Role-based Trust-management framework
  - Distributed Credential Chain Discovery
- Provided techniques for understanding and managing policy
  - Safety and availability analysis in trust management

© William H. Winsborough