

# Security of Shared Data in Large Systems

Arnon Rosenthal  
Marianne Winslett

Obtain slides at VLDB web site, <http://dais.cs.uiuc.edu/pubs/>, or from speakers' USB devices

## Agenda

- **Introduction**
  - **History, and an attempt to diagnose what inhibited technology transfer**
  - **Challenge problems appetizer**
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- Security issues and opportunities in example application areas

## This tutorial is unusual

*We want to help researchers move into this area, and produce results with broad impact*

- Most tutorials teach you about the state of the art
- We emphasize *open problems* (research+ practical steps)
  - Securing large systems and large information structures (databases, middleware objects, document bases)
    - From n-tier to emerging
  - Security problems where data management skills are helpful
  - General DB problems whose solutions
    - help us improve security
    - can benefit from security techniques
- We select problems for
  - leverage with previous DB research and skills
  - benefit to the most widespread DB applications

## What's been added to DBMS security since 1980s

- Roles, role hierarchies
  - SQL role is a set of privileges *or* users
  - But industry did roles, DB researchers arrived after
- Receive “identity” information from middleware or OS
  - But can't use it in a view definition
- Filter query response based on row or column security labels (described later)
- Security for new features added to SQL
  - Triggers, nested tables, objects, procedures
  - Security features are *tightly* coupled to data model

## Which additions owed a debt to data security researchers?

*Why were we unable to help vendors (enterprises) improve this (now-critical) aspect?*

- Vendors' interest in security was mild (but nonzero)
- Too few ideas were worth transferring --- why?
  - Do we *respect* the concerns of DBMS and tool vendors?
    - Simple, rigorous semantics, e.g.,
      - Few fundamental constructs
      - Few tricky feature interactions
    - Compatibility with the past
    - Manageable size for each extension

*These generate neat research issues, too*

## Wrong problems

- Inelegant – unlikely to yield clear insights that may be useful in other situations
- Unrealistic: fail the “giggle test”, *even long term*  
Without laughing, describe a *full* scenario where customers might pay -- buy the software, capture system descriptions, specify policies, ...
- Too many preconditions that are difficult to meet
  - *Distributed DB security*: relied on Deny to override Grant
  - *Prevent an adversary from inferring info they cannot access*: Enterprise must first protect individual columns! Also, document what an adversary knows, forbid anonymous access, be able to query past accesses.

## Right problems, wrong proposals

### Results were unready to transfer to developers

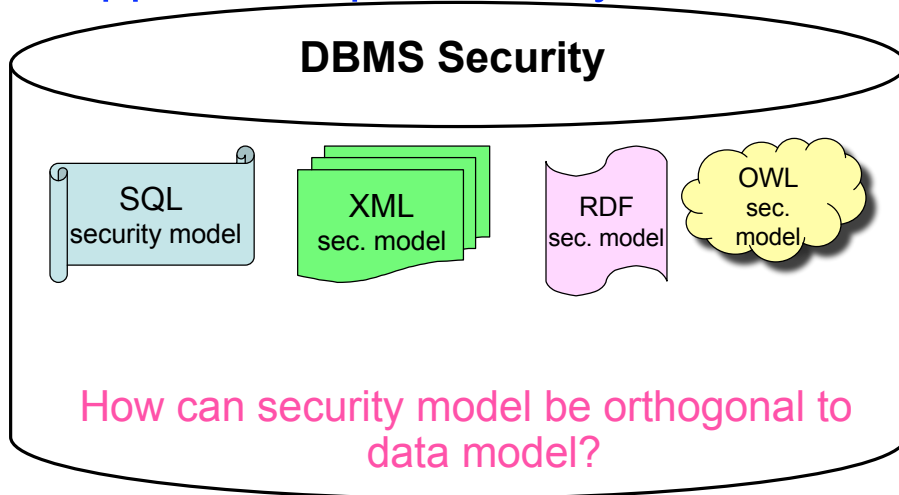
- Non-modular
  - Reinvents non-security functionality, e.g., new query optimizers, temporal and spatial datatypes
  - Need several difficult features at once (distribution, negatives)
- Useful functionality, but administration did not scale
- Semantics were filled with special cases (e.g., Deny)
- Features not reconciled with *full* SQL
  - Often created for middleware policy engines
  - Unknown interactions with view and metadata security, trigger semantics, ...

*Excellent problems for a beginning researcher*

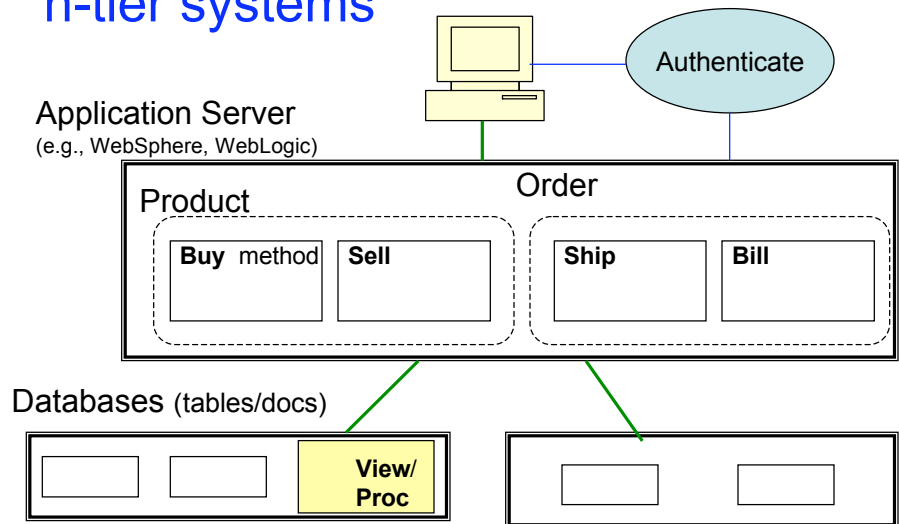
## Three “big” research challenges to whet your appetite

- Allow one DBMS to support multiple security models
- Compile high level policies down to executable mechanisms
- Rewrite another system’s policy in your own terms

# 1. How can one DBMS best support multiple security models?



# Security policy chaos in today's n-tier systems



## 2. Compile “business” policies to physical implementation

15

Individually identified medical data shall be available only to professionals treating the patient, *with medium-high confidence*



Install policies on tables, documents

*Suitable*

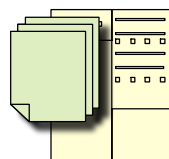
- *data allocation*
- *execution plan*

DAIS The Database and Information Systems Laboratory  
at The University of Illinois at Urbana-Champaign

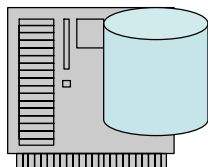
MITRE

## 3. Translate, transfer policy across organization and system boundaries

16



**Aetna Travel Insurance**  
Enforcement: Application server  
Policy applied: US (NY)  
Roles: HiPAA (Aetna version)



**Paris Hospital**  
Enforcement: DBMS  
Policy applied: France  
Roles: Hospital (Emergency Care)

DAIS The Database and Information Systems Laboratory  
at The University of Illinois at Urbana-Champaign

MITRE

## Common themes to these and other research challenges

- Reduce workload and skill to administer policies
- Cope with heterogeneity
  - In security info (formalisms, role sets, policies)
  - In data (data model, schema, instances, semantics)
- Compare desired policy and actual result
- Trust in partners for policy specification and/or enforcement
- Cope with distribution, autonomy, evolution, but exploit favorable simpler cases

## Agenda

- Introduction
- **Security basics**
  - **Desirable properties**
  - Getting there
- State of the art and open problems
- Policies as a unifying framework
- Security issues and opportunities in example application areas

security basics

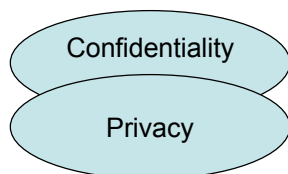
19

## Confidentiality

- Prevent information from going to the wrong recipient
- Not synonymous with privacy

## “Privacy-preserving X” *harmfully* blurs a useful distinction

21



*Inhibits communication with conventional systems, privacy advocates*

- Confidential info sharing (non-disclosure) is useful for proprietary info, with no privacy issues
- Privacy advocates include many other measures in their policy – e.g., must notify



## Integrity

- Ensuring data is right
- Definitions of “right” in different communities:
  - System Security*: Not changed inappropriately
    - E.g., tamper-evident signed message digests
  - IT Security*: Produced appropriately [Biba, Clark-Wilson]
  - IT*: Data quality (freshness, precision, provenance, ...)
  - DB*: Satisfies all relevant constraints
    - E.g., ACID transactions, key constraints
- Related issue: trust
- Too rarely all considered together

## Trust & data provenance

- Trust: willingness to rely on an entity for a particular purpose
  - Hot topic in open systems
- Trust in data depends on its integrity, freshness, accuracy, provenance, its source’s reputation and objective properties, etc.
  - Data provenance is a hot issue for scientists and intelligence analysts
- How can we integrate these concepts to specify and reason about the level of trust in a data item?
  - Particularly interesting in the context of derived data and in information integration



security basics

## Authorization

- Can this party do this action on this object
  - Should there be a side effect (e.g., audit log entry, email notification,...)
- Some approaches to authorization policies
  - Unix file system
  - Role-based access control
  - Attribute-based access control
  - Security levels

security basics

## Intellectual property issues

- Easy case: recipient cooperates, e.g., between government agencies
  - Pass policy to recipient, in terms of objects the recipient understands
  - IBM, others work on sticky policies
- Tough case: adversary owns the machine
  - Not necessarily about secrecy 
  - Goal: cradle-to-grave control over access 

*Not addressed in this tutorial*

security basics

26

## Confidence

- Likelihood that desired security properties hold
  - Relative to a threat model
- Some practices to judge confidence, and use it:
  - Certify: reviewer announces their confidence in a description of system behavior
  - Accredit: executive decides that benefits exceed the risks

27

## Agenda

- Introduction
- **Security basics**
  - Desirable properties
  - **Getting there**
- State of the art and open problems
- Policies as a unifying framework
- Security issues and opportunities in example application areas

## Access control and release

- *Access control policy* governs pull situations
  - Bob wants to do an action on Alice's object; will Alice let him?
- *Release policy* governs push situations
  - Assuming Alice has read an object, can she send Bob a copy?
  - Used in government, and for proprietary info (mostly for read-only objects)
- Not independent:  
Bob can Access  $\Rightarrow$   
Alice can Release to Bob

## Delegation

- Your declaration of when another party will be speaking for you / acting for you
- Most often: one party grants a right to another party
  - E.g., to perform a specific kind of action on a specific object
- Examples
  - SQL "with grant option": unconditional delegation
  - Verisign delegates right to create identity credentials
  - Trust management languages offer conditional delegation  
Authorize(Arnie, Purchase) :=  
Authorize(Marianne, Purchase), Purchase.Amt < \$100

## Enforcement, credentials

- Enforcement approaches
  - Server routes all requests through a “reference monitor” (DBMS, application server, OS)
  - Check when a boundary is crossed (usually physical): firewalls, gateways
    - Can be very small server, hardware assisted, with high confidence for *simple* policies (e.g., filter for forbidden words, XML filtering)
- Credentials approaches
  - Server holds them and checks (e.g., DBMS authorization)
  - Mobile (single sign-on, trust management)

## How to decide if you’re “there”

1. Where is “there”?
  - Decide what actions/states wrt your data are legitimate/forbidden (create your *policies*)
  - Determine the likely threats
2. Pick/develop technology to mitigate the risks to acceptable levels
  - Consider implementation constructs’ resistance to known threats (e.g., data partitioning in case of machine takeover)
  - Do a cost/benefit analysis
3. Evaluate your proposed technology as follows

## Evaluation criteria (for both researchers and developers), 1

- Passes the giggle test (on cost/benefit)
- Usable
  - No CS degree should be required of users *or administrators*
- Cheap enough
  - Development effort, learning curve, admin
- Scalable
  - To large numbers of objects, subobjects, actions, subjects, organizations, sites
- Analyzable
  - Current state: what a given subject/object can do/have done to it
  - What-if queries: determine effect of changes in advance

security basics

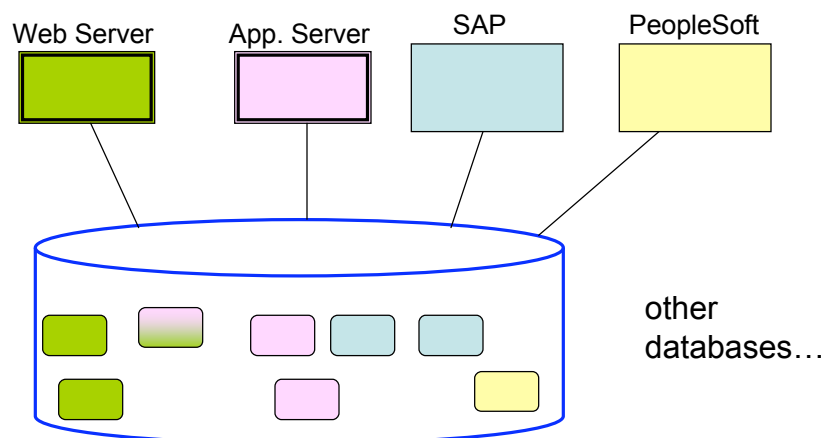
## Evaluation criteria, 2

- Flexible, extensible
  - Rapid response to unanticipated emergencies, opportunities
- Modular/universal/orthogonal/composable/compatible
  - Applicable in many places, many futures
  - Can others build on your solution (clean, high quality)?
- Rigorous (thorough)
  - Behavior of foundational components should be fully captured by the model---**hard to anticipate future uses**
  - If implementations leak info (e.g., about “secret” view definition), bring into the model by requiring release privilege

## Agenda

- Introduction
- Security basics
- **State of the art and open problems**
  - Problem context (a reality check)
    - SQL
    - Privilege limitation
    - Role-Based & Attribute-Based Access Control (*RBAC*, *ABAC*)
    - Label-based access control
- Policies as a unifying framework
- Security issues and opportunities in example application areas

## A common architecture: each DB object belongs to ~one server



problem context

## Policy administration in enterprises

- DBs are not the center of the policy admin universe
  - Few researchers at the Access Control conference (SACMAT04) really knew the SQL security model
- A policy must be conceptually near the resources it controls
  - Middleware knows application methods, e.g., Admit(Patient)
  - DBMS is smart, fast with structured info, consistent when there are multiple paths to same datum
- Database security administration is often ignored
  - 30% assign privileges to real users or roles, mostly to entire tables
  - 70% use DBMS security only to restrict each table to one app
- Consider nontechnical fixes: Packaged applications may move to a built-in security policy

problem context

## Scale

- SAP has  $10^{**4}$  tables, GTE over  $10^{**5}$  attributes
- A brokerage house has 80,000 applications, a US government entity thinks that it has 350K
- Admin and implementation require
  - Automated help
  - Massive delegation (within limits)
- Our advice
  - Start with broad, general security policy statements
  - Refine under pressure
  - Beware: in formal acquisitions, contractors often build to the letter of specifications, not the spirit



problem context

## Policy administration in enterprises

- DBAs are considered untrustworthy (too casual) to be given superuser-type powers
  - But they still have complete privileges
  - Thus: extra layer, controlled by security officers, to limit/audit DBAs
- Administrators need training in both technology and judgment – making evolution costly and slow. Simplify!
- Single sign-on is typically the top priority, rather than policy specification

problem context

## Management of security data

- We collect lots of security-related data
  - Audit trails, surveillance video/camera, RFIDs, GPS, cell phones, electronic lock records, etc.
- How can we analyze it and assess its quality in a scalable manner?
  - Relevant research: mining patterns of normal/ anomalous operation, metadata management, protection against alteration, privacy issues

*Not discussed much in this tutorial*

## Agenda

- Introduction
- Security basics
- **State of the art and open problems**
  - Problem context
  - **SQL**
  - Privilege limitation
  - Role-Based, Attribute-Based Access Control
  - Label-based access control
- Policies as a unifying framework
- Security issues and opportunities in example application areas

## SQL security model overview

- Privileges on data objects (tables, columns), schema objects, stored procedures, triggers, more in the future
 

```
grant <list of operations>
  on <list of objects>
  to <list of identities>
  [with grant option] /* right to delegate */
```
  - A privilege must be supported by a chain from owner
    - When grantor loses rights, revoke cascades. *So DBA grants all?*
    - Delegation is only for privileges you have
  - Object creator is “owner”, with full privileges
    - Ownership cannot be transferred
  - Schema is visible iff user has some rights on the object
    - View/procedure definitions only for the owner
- Models for distributed trust, label security, XML security diverge from these design choices*

## SQL lacks many essentials

*Some (neatly bounded) extensions needed by SQL2003, RDBMSs, and many other data/query models*

- **Manage security for a *distributed* relational database** (Issues: double admin for views (even synonyms); local autonomy)
- **Infer a user's right to view a subset of the data, *transparent* to application writers (views are not)**
  - Without changing query semantics
- **Guarantee that administrators do not delegate excessive privileges**
- **Decentralize power appropriately** (ownership, DBA roles)
- **Abstract and modularize the specification of the standard, so it can be extended safely and easily**

## Build grad students' muscles Rework "ownership"



- Owner of container currently gets full rights to the contents!
  - Owner's real contribution was metadata and creating a container, not data content
    - So why should they have full privileges?
  - Upon creation, transfer creator's content and metadata privileges to "domain" administrators
  - Allow any user (including owner) to "move" their rights to someone else
    - Avoid cascading revoke
    - Allow recipient to gain *sole* ownership

## Build grad students' muscles

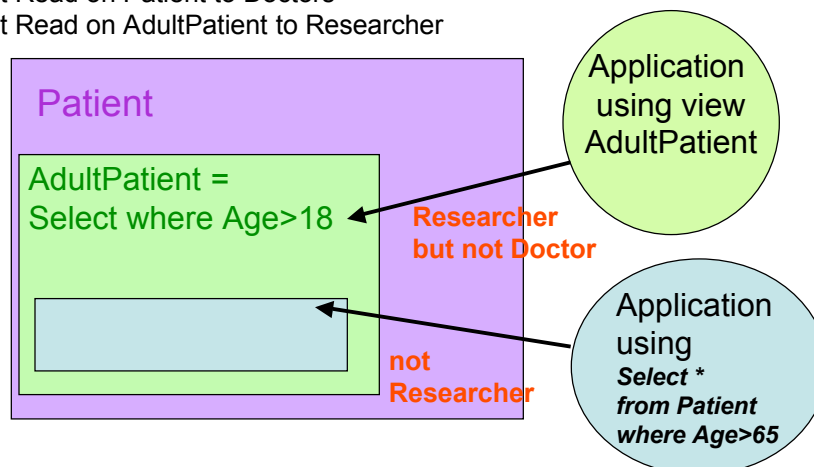
# Control metadata visibility

Select GoodCredit from Customer where  
scoringFunction(ZipCode, Age) > 6789

- Devise a suitable model for metadata protection
  - Publish or protect business process info in view definitions
  - Controlled browsing of catalogs by users who lack access to underlying data
- Requirements for the solution
  - Minimize admin work
  - Retain privileges that users have already granted
  - Avoid loose ends (e.g., who may use each m'data item to enforce a constraint or rewrite a queries)

## SQL view privilege $\equiv$ the right to use the view interface

Grant Read on Patient to Doctors  
Grant Read on AdultPatient to Researcher



## Privileges on views and procedures (i.e., derived objects)

- Principle: Infer a privilege *when you detect* that it does not increase user's power
  - Interacts with metadata, distribution, ownership, ...
- *Implement privilege inference efficiently*
  - Adapt the query optimizer to generate equivalent forms
  - Detect equivalences that hold in the current db instance [Rizvi et. al. 04]
    - Practical case(?): Examine just the query result
- *Handle federation and warehouse (materialized) views, with minimum new semantics and mechanism*
  - Autonomy: control over security stops at organizational boundaries.
  - Negative privileges are a big, controversial add-on

Often a query will not be answerable from user-visible info. (This is a general problem in publish/subscribe)

- *Suggest an alternative query that the user can execute, and explain how it differs from what they requested*

## Build grad students' muscles Abstract models for SQL

- *Help restate the standard (+ vendor products), in a way that enables easier extension, integration*
  - Describe query/update execution semantics in a way that shows what operations may be executed [RoSc04b]
    - Use it to explain needed privileges
    - Rewrite statements on views as SQL statements on underlying tables
  - Use abstract concepts, e.g., contains, is-a, derived object (perhaps from object models)
    - Compare with constructs in other models

## Kinds of privilege limitations [skip](#)

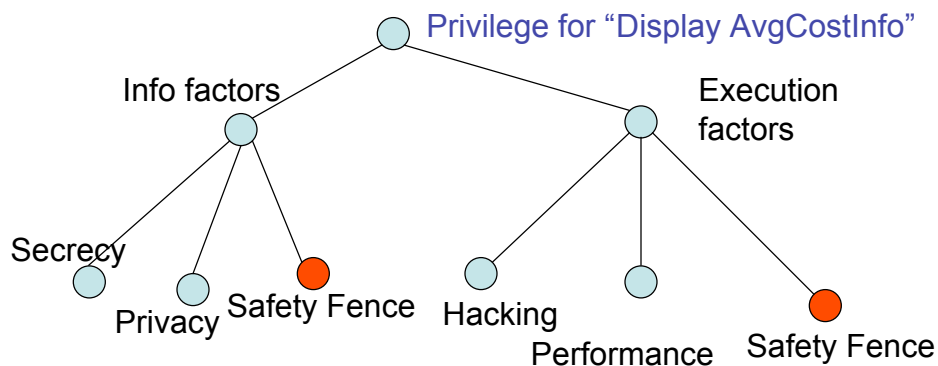
- Revoke: an extreme case of privilege limitation?
- Local Deny (w.r.t. a given grant): Equivalent to imposing a predicate restricting use of the privilege [RoSc00, Sadhigi03]
- Global “Deny” (asserted/revoked as grants), sometimes with predicates, overrides [many, e.g., Jonscher, Jajodia, Bertino, ...]
  - Violates delegated administration?
  - Can administrator understand the state?
- Privilege factors: separate concerns among collaborating administrators (Semi-static, organization-friendly)
- [Attach predicates to privileges \(or denials\)](#)

DAIS

The Database and Information Systems Laboratory  
at The University of Illinois at Urbana-Champaign

MITRE

## Help administrators *collaborate*: Decompose privileges



Get an attribute if all children are satisfied or by direct assertion  
 Simpler decisions, single skill, independently administered  
 Changes easily: When situation changes, review just that part  
 Safety fence provides guarantees

⋮

## Denial versus safety fences

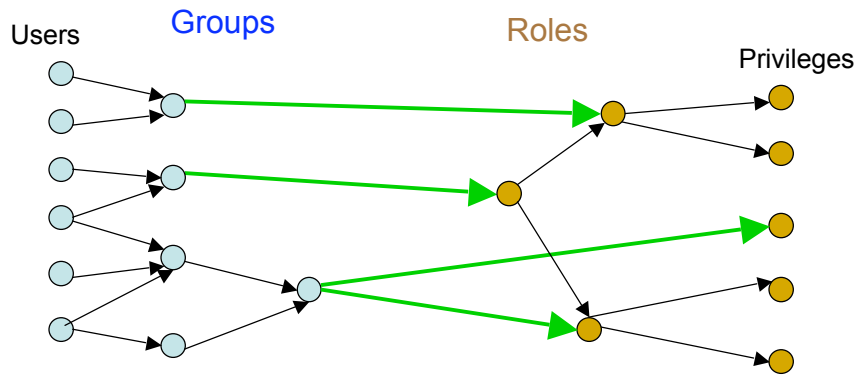
- Compare *pragmatics* of denial-based approach and “safety fence” factors
  - Reformulate as a trust management problem, with factors as predefined attribute types
- *Meta-problem*: Define and apply criteria for comparing proposed facilities’ “simplicity”
  - Ease of administration (learning curve & admin effort at small/large scales)
  - Expressiveness and flexibility (suiting the needs)
  - Ease of implementation by vendors, in various architectures (e.g., policy mgmt system downloads grants to DBMS)
  - Efficient implementation
- Implement *the best* admin models (once known)

## Agenda

- Introduction
- Security basics
- **State of the art and open problems**
  - Problem context
  - SQL
  - Privilege limitation
  - **Role-Based and Attribute-Based Access Control**
  - Label-based access control
- Policies as a unifying framework
- Security issues and opportunities in example application areas

## Group/role graph

- Reduces admin labor
- Decentralizes admin



How far can this “graph” visualization go?

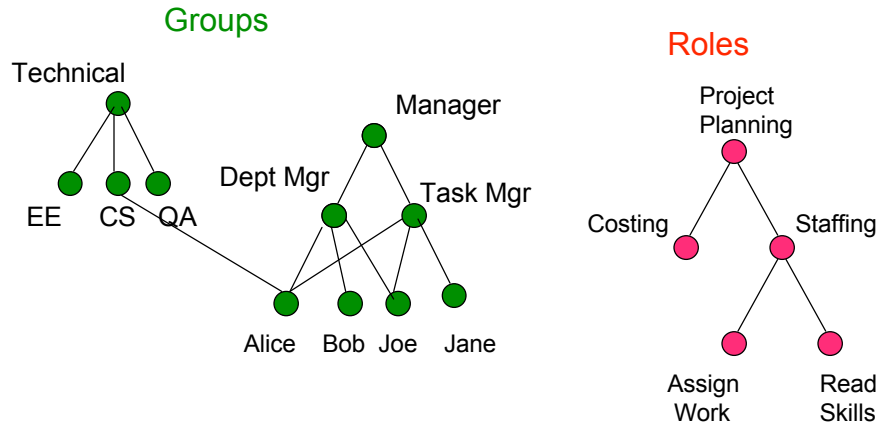
- Grants that require multiple authorizations
- Communities of mutual trust

## Two guidelines for thinking about RBAC

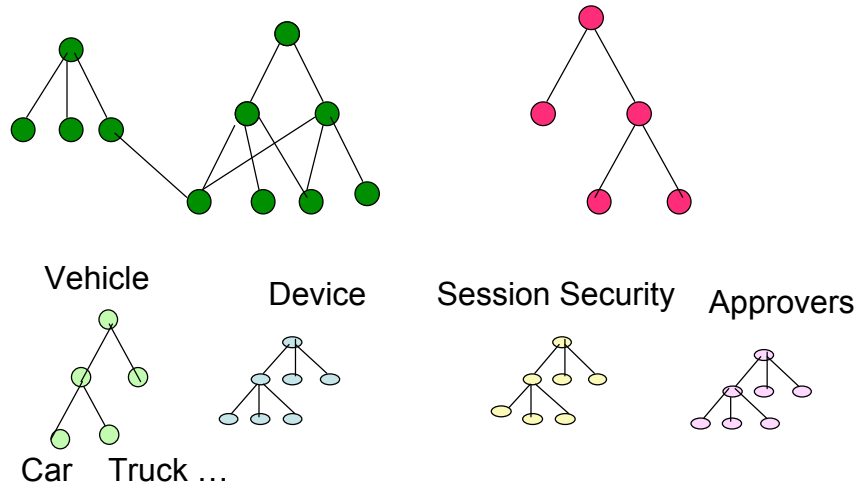
- Security policy is hard, inevitably a tradeoff. Minimize the need to make it!
  - Treat each group, each role as just a definition
  - Create a clear membership criterion for new arrivals, suited to routine
  - Now, authorizing a group for a role is the only real *security* decision
- The distinction between groups and roles is *essential* for admin, minor for enforcement
  - Debates are confusing, because both sides are right



## RBAC is not sufficient



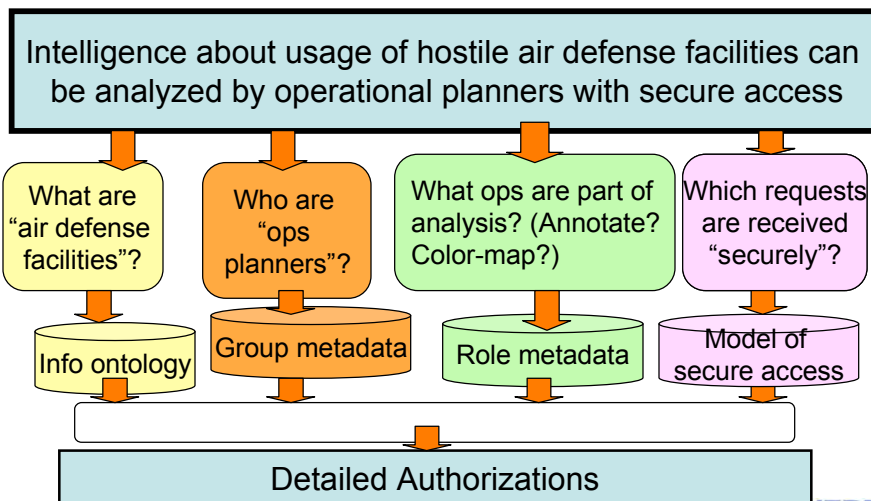
## Policies can involve many other hierarchies



## Attribute Based Access Control

- RBAC extensions are awkward
  - Unnecessarily asymmetric: Task Mgr in CS Department--which is the group?
    - *Several* attributes can have hierarchy
    - “Parameterized roles” bring in additional attributes, and allow predicates over all. But only one attribute can be hierarchical “Role”
  - Some attributes are not role-like (e.g., user location) or not associated with the user (e.g., time of request submission)
- Attribute-based access control: policy can be *any* predicate over *any* attributes
  - E.g., roles, groups, where/when submitted, alert-level, approvals...
- Beyond IS-A: Derive attributes from other attributes
  - Derive using logic? SQL? Arbitrary functions?

## Unify reasoning about semantic aggregates (i.e., support “is-a” just once)



## Research issues for RBAC and ABAC

71

- Role engineering: How should an organization select groups and roles? (determine appropriate clusters)
  - Mine the existing workload, to suggest “good” roles, groups, and privilege assignments
- Policy admin: Which groups *should* get which roles (generalizes “Which users *should* get which privileges?”)
  - Infer logically, mine similar workloads to reduce effort
- *Elegant* models needed!
  - Provide clear criteria to explain why a model is good
    - E.g., be minimal, formalized enough to be analyzable
  - New feature = New paper? More is better? *No!*
- Issues from earlier sections still apply: Ownership, privilege limitation, use of ontologies in policy specification, ...

## Supporting technologies for ABAC

72

- Standards
  - Pass attribute assertions (*SAML*)
    - Ed@bc.edu says Patient.BirthYear = 1984
  - For each action, attach predicates that reference attributes (*XACML*)
    - Four valued propositional logic expressions
    - Connect actions to policies (with conflict resolution)
- Semantic web (*OWL*) or logic (many Datalog dialects) for reasoning about hierarchies, restriction predicates, derivations
- Federated data perspective needed to get attributes to evaluators
  - CORBA specified a standard way to *pull* an attribute from a particular server

## Agenda

- Introduction
- Security basics
- **State of the art and open problems**
  - Problem context
  - SQL
  - Privilege limitation
  - RBAC & ABAC
  - **Label-based access control skip**
- Policies as a unifying framework
- Research issues and opportunities in example applications

## Mandatory versus discretionary security

- *Discretionary*: owner and owner's delegates can change the access rights
  - Although controls over arbitrary delegation can be useful, to limit eventual spread of rights
- *Mandatory*: A party possessing an object cannot
  - Release it to arbitrary others
  - Change the policy
- Policy is often inherent in object *label*
  - E.g., Top Secret, Proprietary

## A mandatory policy: multi-level secure databases (MLS)

- Read allowed if *dominated*:  $\text{SessionLabel} \geq \text{ObjectLabel}$   
(e.g., suppose Proprietary > Public)

<u>Public</u> :	aspirin	.1	.5	.23
<u>Proprietary</u> :	aNewDrug	.6	.9	.85
<u>Public</u> :	aNewDrug	.6	.9	.85

- “High” session cannot put data where “low” sessions can read it
  - Write allowed if  $\text{ObjectLabel} \geq \text{SessionLabel}$
- Prevents inadvertent mistakes by programmers
  - Inadvertent writes without needed labels
  - Enforces hierarchical rules even if administrator is careless
  - **Protect against malicious user or Trojan Horse – no info “leak” (?)**
    - For high confidence, must also restrict export from user program

## Market drivers for commercial label-based access control

- Application hosting and outsourcing
  - Independent franchises share a single table at headquarters (e.g., Holiday Inn)
  - Application runs under requester’s label, cannot see other labels
    - Its Read and Write operations on the shared table are quite safe
  - Headquarters runs Read queries over them
- Proprietary data consolidated from many sources
  - E.g., at a government agency or system integration contractor
- Individuals’ privacy preferences?

## Commercial label security

*Guarantees that application requests are directed to a parameterized view (and handles the parameters)*

- Runs in normal environment
- Policy applies to operations on policy-governed tables
  - Conjunction with ordinary SQL security
  - Finer grained than table privileges
  - Transparent to user code, but changes semantics
- Is easily turned off – everything is optional, controllable
- Programs can write files, send email, ... since OS is not MLS

## Oracle's label security

- SessionLabel, ObjectLabel are tuples of atoms, e.g.,  
(Secret, Manager, {heart, blood}) [see Oracle website]
  - Ordered slots: [Unclassified, ..., Secret, Top Secret]
  - Group slots (management hierarchy, projects, ...)
  - Unordered slots (compartments)
- Implementation: system creates, manages views (Read) and Instead-Of triggers (Write)
  - Admin declares a table as labeled (system adds "label" column)
  - System generates labels on insert
  - System rewrites user's action, to apply only to the view
  - For performance, tweaked the query optimizer
 Semantics: "Return filtered result", not "reject"

## Research issues in label security

- Support both filter and reject semantics?
- Manage “structured”, audited exceptions (downgrading)
  - Use SQL grant option for exception?
  - Integrate access controls with audit?
- Indexing and query opt. for row, column, cell labels [Lefevre]
  - Too slow to first filter, then merge
  - Oracle labels were too slow until query processor was tweaked
- Allocate (partition) data to provide sufficient confidence
  - Precategorize potential implementations w.r.t. how much “confidence” they give
  - Partition data among rows, tables, DBMSs, machines, networks

## Agenda

- Introduction
- Security basics
- State of the art and open problems
- **Policies as a unifying framework**
- Security issues and opportunities in example application areas

## What is a policy?

- A statement regarding who/what is allowed/required/forbidden to take what actions, when and where, with respect to whom/what objects
  - May also describe what happens after the action is taken, or if the policy is not followed
  - May be stated in terms of abstract security properties such as availability, privacy, etc.
- A *consistent* set of assertions a system view *must satisfy*
  - System view may be partial, include history, and future (obligations)
  - Constraints, obligations are nondeterministic policies

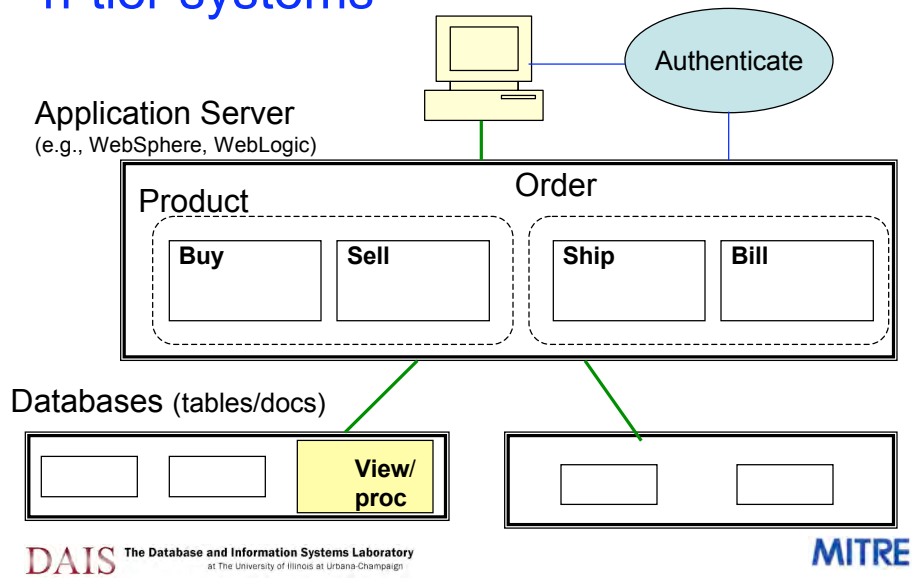
## Example policies for access control, authentication, info release

- EMTs can access blood type info in the ambulance
- Every patient can read their own medical record
- Physicians have dial-up access to medical records
- Nurses cannot examine billing information
- Hospital administrative staff can modify policies
- Purchase transactions over \$1000 require 2 forms of authentication (retina scan, employee ID, passport)
- Asserted behavior can depend on many attributes
  - User, operation, role, object type, object attributes, where submitted, when submitted, + **trust**
  - Policy's action may include "reject", filtering, notification, penalty, ...
- Policies are requirements, and have the whole gamut of software engineering issues (details later)



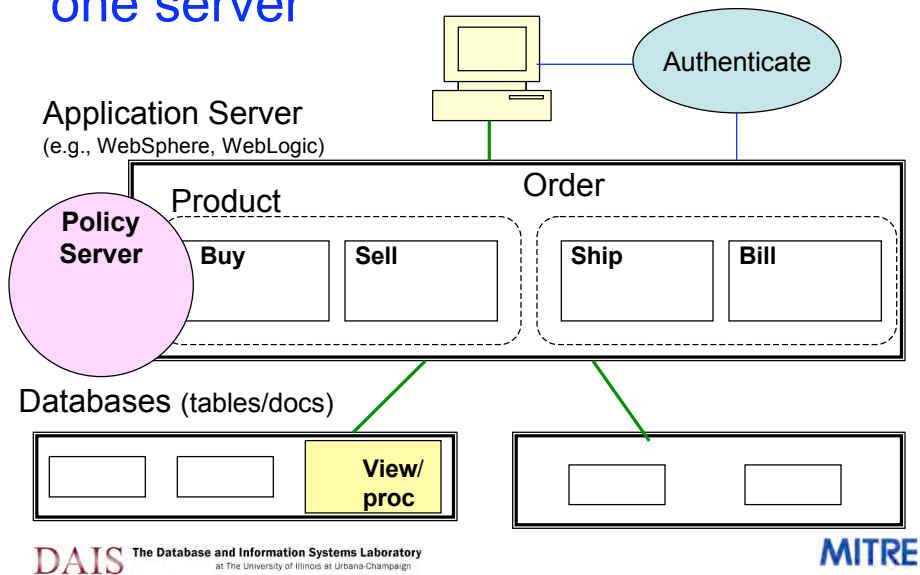
# Security policy chaos in today's n-tier systems

87



# Gather into policy space: one server

89



## Where are policies captured and enforced today?

90

- They tend to stay in one place
  - Captured for a database, app server, or policy server, *in terms of objects that server knows*
  - Delegation is checked there
  - Entire policy is enforced there
- Desired scenarios
  - Capture in server, enforce redundantly in client GUI (better interactive behavior)
  - Capture at one server, but delegate enforcement to elsewhere
    - E.g., ACM delegates to SIGMOD the task of ensuring adherence to its guidelines for in-cooperation conferences
  - Split enforcement into several parts
    - E.g., evaluate SecureChannel attribute, evaluate UserAuth, and conjoin to determine RequestAuth

## Abstraction in policies aids decentralized security admin

94

- Subjects
  - Residents of California over 21 years of age
  - Parents and legal guardians of children enrolled in King School
  - Purchasing agents of the University of Illinois
- Objects
  - Anything containing the SSN “123456789”
  - Anything about underground democracy movements in country xyz
    - May require IR techniques to identify
  - Any file in any subdirectory of this directory
- Actions
  - Sending email, FTP, GET/POST requests, IP packet transmission, queries, invoking a method, ...
  - “Push” systems: release control policy for object to be pushed to subject (see next slide)
  - Actions triggered by the user request (including actions of the security system itself)

ABAC had just IS-A hierarchies, but much more is needed

*How can we provide a good formalism for deriving abstractions?*

## Making policies more abstract

- Describe policies
  - At all levels of a system
  - For all kinds of subjects, objects, and actions
    - *At least* DBs, formatted messages, service calls, general documents
  - From administrative and implementation viewpoints
- Specify each of subject/object/action declaratively (e.g., queries, views, datalog, OWL) rather than by enumeration
- More detail in trust management section
- Hot in AI community for semantic web

## Example policy in Cassandra

### *Treating-clinician reads patient's record item*

```
permits(cli, Read-record-item(pat, id)) ←
  hasActivated(cli, Clinician(org, spcty)),
  canActivate(cli, Treating-clinician(pat, org, spcty)),
  count-access-denied-by-patient(0, (pat,id), (org, cli, spcty)),
  Get-EHR-item-subjects(pat, id) ⊆ Permitted-subjects(spcty)
```

### *Prerequisite for Treating-clinician*

```
canActivate(cli, Treating-clinician(pat, org, spcty)) ←
  org.canActivate(cli, Group-treating-clinician(pat, group, spcty)),
  org@ra.hasActivated(x, NHS-health-org-cred(org, start, end)),
  ra ∈ NHS-registration-authorities( ),
  Current-time( ) ∈ [start, end]
```

Source: <http://www.cl.cam.ac.uk/users/mywyb2>, encoding UK's Electronic Health Record policies

## ABAC+TM research issues

- TM policy languages are logic-based, not user friendly
  - Express/reason about arbitrary relationships, e.g., delegation
  - TM style: “This attribute value has been asserted, and here’s why you should trust it”
    - Requires ability to formulate, reason about trust metrics
    - Internal DBMS support for these new security-related features
  - Classical style: RBAC, ABAC, privilege factors that are structured, updatable, visualizable
- Needed: policy templates and methodologies for policy administration
  - Usable at enterprise, cross-enterprise levels
  - Appropriate expressiveness
    - Monotonic privilege limitation constructs to guarantee what will *never* happen
    - Simple delegation models, with revocation
    - Privilege inference rules, integrated with data ontologies, rules, groups, derived data (views, procedures)

## Policy analysis

- Administrator needs help to analyze policies
  - Show me all the policies that definitely/possibly apply in this situation
  - With the current set of delegations, are users of this type definitely (or possibly) able to perform this action?  
*A killer app for logic databases? What logic? Datalog++ or OWL?*
- Who can *potentially* obtain the right to perform this action (via delegation from untrusted users)?
  - Undecidable in traditional HRU model. Even simpler ones are NP-hard
  - Get user help with policy constructs that break the inference engine
- Metaquestion: can the underlying theory support convenient admin?
  - E.g., how does stratification (for clear semantics) affect admin?

## New application domains that need security policy services

- Pervasive computing
- Sensor, mobile, wireless, and ad-hoc networks
- Semantic web, peer-to-peer systems, grid computing

Security and privacy for these applications are open areas for research

Arnie's rebuttal: "Build a new world of your own design" problems are for wimps. For a *big* challenge:

Security research that simplifies multi-purpose enterprise systems

- *Interaction of many technologies, policies, requirements*
- *Existing systems and languages*
- *Precise semantics*

## Agenda

- Introduction
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- **Security issues and opportunities in example application areas**
  - **Trust management in open systems**
    - **Trust middleware**
    - Open problems
  - DB capabilities for data that *really* needs to be secure
  - Semantic web and XML
  - Enterprise security

## Motivation: move toward open computing systems

Open = resources shared across organizational boundaries

Ability to rapidly form relationships, cooperate to solve urgent problems is vital

- Requires unanticipated sharing
- Supply chain management, crisis response, peer-to-peer computing, semantic web, grid computing, cross-national military activities, joint corporate ventures

## Current DB app trust middleware<sup>106</sup> is awkward in open systems

- Management headaches
  - No abstraction at user (subject) level
    - E.g., clothing vendor has to set up a separate login for each Walmart authorized purchaser
  - Managing passwords is #1 help desk call
  - High turnover in suppliers/users/customers
    - What happens when an authorized purchaser is fired?
- Error handling may be opaque

## What's missing

- Traditional security describes monolithic building blocks
  - Does not help in attaching separate blocks together to build a global perspective in distributed situations
- *Distributed trust management*, an emerging technology,
  - Gives a box of Legos™ and a language (usually Datalog + constraints) for connecting building blocks together

Key goals of work on supporting *modular, distributed, decentralized* trust management:

- Make it easy to use and administer
- Provide improved security and privacy
- Make it *ubiquitous*
  - Facilities available to all types of parties
  - Wherever they are, whatever they might be doing

## Ingredients for generalized trust middleware, 1 <sup>110</sup>



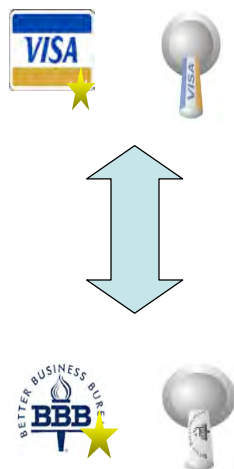
- Credentials, so subjects can prove what attributes they possess
  - Verifiable, unforgeable
  - Provide way to prove ownership or delegation of authority to use
- Party receiving a credential
  - Read and interpret fields (ontologies)
  - Verify ownership
- X.509, PKI and *beyond*

## Ingredients for generalized trust middleware, 2 <sup>111</sup>



- Policy, e.g., for acceptable credit cards for purchases:
  - Acceptable issuers (VISA, MasterCard)
  - Require ownership/delegation to be demonstrated
  - Check for expiration
  - Contact card issuer
    - Revocation, credit limit
- More generally, an access control policy (and possibly other policies) for every resource that a stranger might be allowed to access

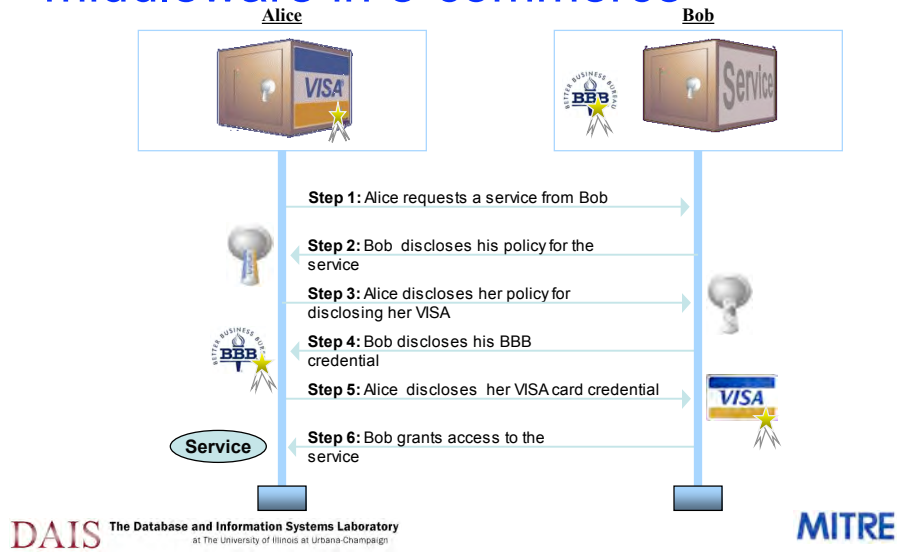
## Ingredients for generalized trust middleware, 3 <sup>112</sup>



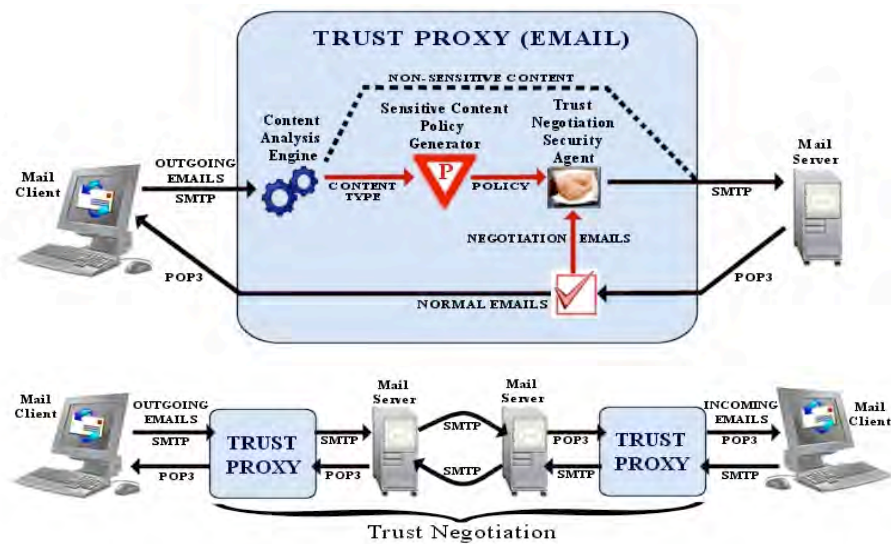
- Ability to export policies (to be read elsewhere, enforced elsewhere)
  - A stranger may need to understand them to gain access to my resources
  - E.g., which credit cards does this merchant accept? What will I require from the merchant?
- Trust negotiation software to control the process of gaining trust



## Example use of trust negotiation middleware in e-commerce



## SMTP-based trust middleware for release policies



## Agenda

- Introduction
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- **Security issues and opportunities in example application areas**
  - **Trust management in open systems**
    - Trust middleware
    - **Open problems**
  - DB capabilities for data that *really* needs to be secure
  - Semantic web
  - Enterprise security

## Policy, credential capture and interpretation

- Expressive policy languages (details follow)
- Administrative tools and algorithms
  - Write, update, understand, and analyze (details follow) policies
- Standard schemas/ontologies for popular types of credentials and policies
- Efficient policy compliance checkers
- First-class policies
  - Search them, query them, ...
  - Give them protection as strong as for any other resource
- Policy integration, translation, compilation (details follow)
- Verification of approaches to all the above in the context of particular applications

## Needed policy language features for trust negotiation

- Well-defined semantics
- Monotonicity (sort of)
- Everything relational algebra can do, plus transitive closure
- Support for delegation
- Clean integration with reputation-based trust systems
- References to the local environment and external functions (e.g., time of day, current user)
- Explicit specification of authentication requirements
- Tractable for analysis

*Datalog + constraints [Cassandra, RT], OWL (for its ontologies) are viewed as likely policy language choices in various research communities*

–May have good complexity for analysis tools

## Monotonicity and outcalls

Purely monotonic languages are not expressive enough for trust negotiation

- Do not want customer's withholding of a credential to increase their privileges
- But need elegant handling of time, revocation checks, ...
- Anything less than Turing-complete will require outcalls (but must bound them, as analysis capability is *vital*)

*For realism, language design needs to be application driven*

## Trust middleware architectures

- Trusted third parties that are not vulnerable to attack
- Direct peer-to-peer
  - With disclosure of credentials/policies
  - Zero knowledge/hidden credentials/OSBE

## Obtaining and storing credentials

- How do I get them?
- Where do I keep them, to keep them private?
- How can I quickly find credentials I haven't cached already, during a negotiation?
  - Credential chain discovery, n-party trust negotiation, push/pull paradigms, federated DBs, ...
- Efficient ways to deal with revocation
  - Get rid of revocation, don't check for revocation, check quickly, ...

## Scalability and deployment

- Good implementations of trust management facilities
  - Modular, scalable, reusable
  - Support ubiquitous trust negotiation
- Deployment of trust negotiation
  - In today's popular communication and query/response protocols (SOAP, IPsec, TLS, etc.)
  - Backward compatible

## Vulnerabilities

- What kinds of attacks is trust negotiation vulnerable to?
- How can we mitigate the danger?
- What parts of the process/system must be trusted, and to what degree?
- What integrity/privacy/confidentiality/... guarantees can we give?

## Confidentiality guarantees

- Can outsiders eavesdrop on negotiations?
- Can I disclose just part of a credential?
- Can there be a concept of “need to know”?
  - Can its administration scale?
- What can be inferred about my credentials without my directly disclosing them?
  - Fix by adding release privileges for “leaked” info?

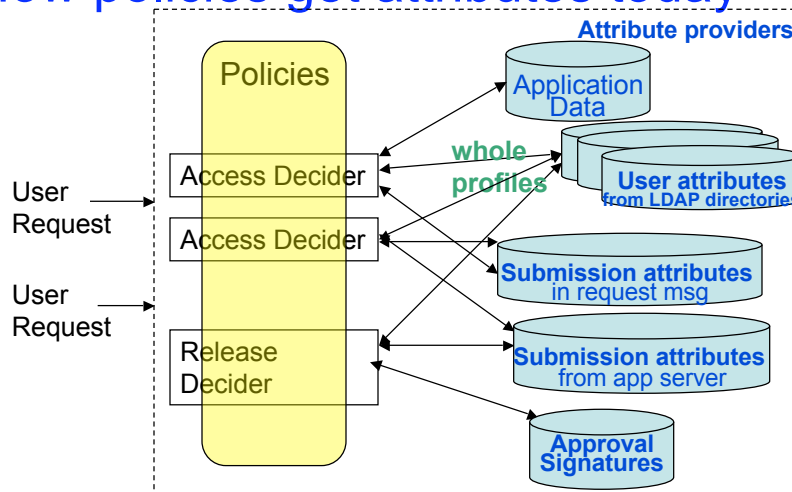
## Managing multiple identities

- Support for many identities has many benefits for issuers and owners, today and in the future
- How to prove I possess several identities, while preventing or penalizing collusion?
- How to make my identities unlinkable?

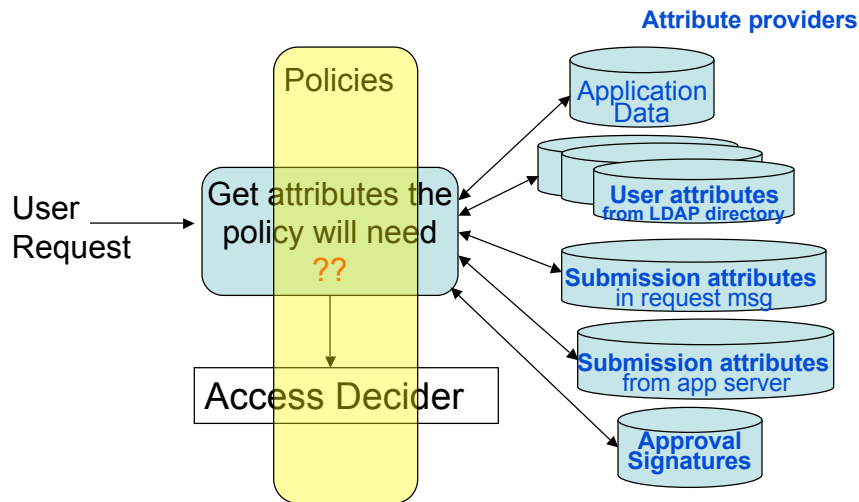
## Agenda

- Introduction
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- **Security issues and opportunities in example application areas**
  - Trust middleware
  - DB capabilities for data that *really* needs to be secure
    - ABAC as a DB application
    - Data mgt challenges for security-critical data
  - Semantic web and XML
  - Enterprise security

## ABAC as a data intensive app: How policies get attributes today



## XACML approach



## ABAC policy evaluation as an ordinary data intensive application

- We need to apply federated DB technology to security systems to manage:
  - Semantics: what do attributes mean?
    - Managing definitions and doing semantic integration (e.g., via communities of interest?)
  - Locating attributes: held in directories, DBs, services
  - Trust: why should I believe the attribute?
    - Integrate delegation, data quality, provenance, source selection...
- Metadata and policies need access controls too
  - Need fine grained protection!



## Hardening a DBMS-based system against malice

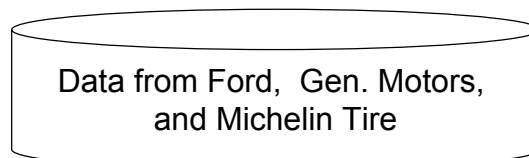
136

- Secrecy, correctness are crucial in many data intensive applications
  - Finance, medicine, military operations, control systems (chemical, nuclear, aircraft, ...)
  - Security (user and other attributes)
- DBMSs *are* used in such environments (less for security), but ... how to mitigate malice?
  - Example vulnerabilities:
    - Accessible from the Internet
    - Multiple classification levels on same system
    - Competitors on same system, e.g., Ford user reads Gen. Motors data
  - Help design physical separation

## Approaches based on physical separation

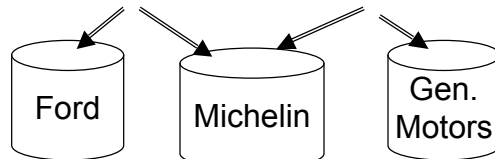
137

- Harden the system against attack, e.g., “Appliance” offering few services, no end user access
- Physically separate sensitive data from users who may attack it
  - Methodologies are ad hoc, seem to have no tools*



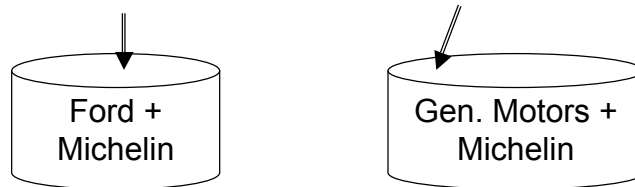
## Approaches based on physical separation

- Harden the system against attack, e.g., “Appliance” offering few services, no end user access
- Physically separate sensitive data from users who may attack it  
*Methodologies are ad hoc, seem to have no tools*



## Approaches based on physical separation

- Harden the system against attack, e.g. “Appliance” offering few services, no end user access
- Physically separate sensitive data from users who may attack it  
*Methodologies are ad hoc, seem to have no tools*



## Data intensive applications and security/correctness

140

*Target systems: DBMSs, middleware, document managers*

- Create models and tools to
  - Calculate attack resistance of a particular design, from a given threat
  - Allocate data automatically (extend autonomic admin)
  - Adjust query processing techniques
- Integrate data quality, provenance and transitive trust (for both “normal” and secure applications)

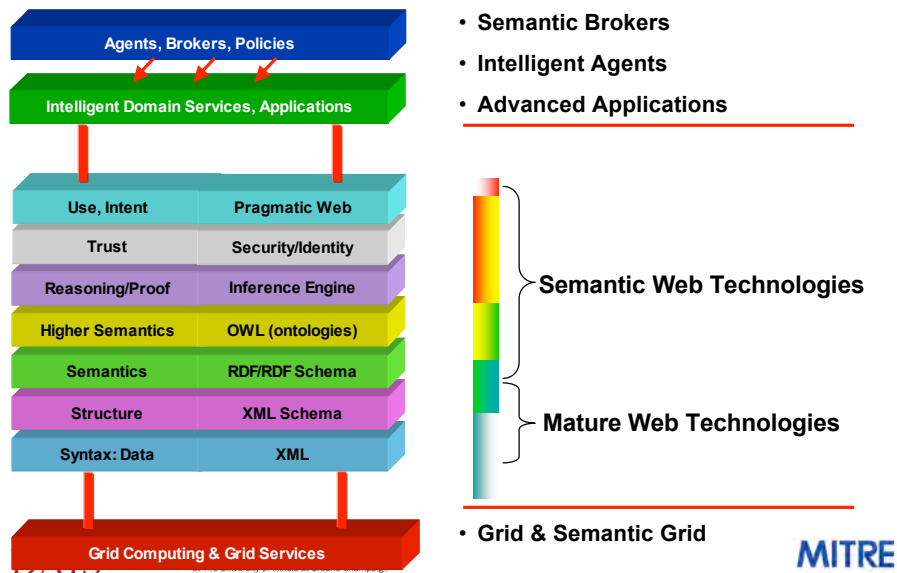
## Agenda

141

- Introduction
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- **Security issues and opportunities in example application areas**
  - Trust middleware
  - DB capabilities for data that *really* needs to be secure
  - **Semantic web and XML**
  - Enterprise security

# Semantic web context

142



143

# XML security directions: examples

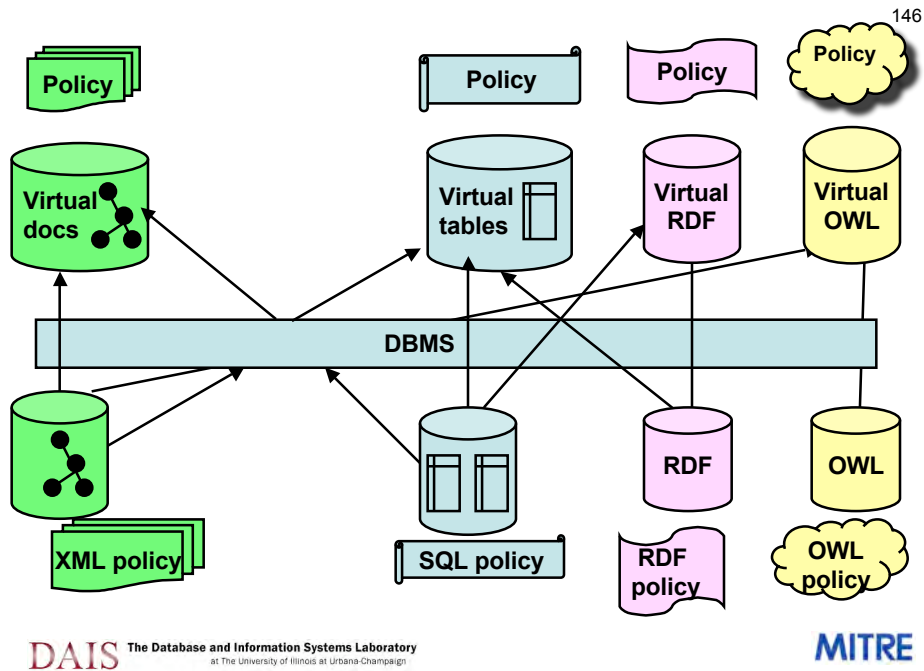
- Use XML as a language syntax for *any* sort of language, to make it tool-friendly
  - For security languages too: XACML for access policy on *any* resource
- There are standard ways to express security labels, now also in XML
  - “UltraProprietary, release to Drug\_Trial(foo)”
  - “Secret, No Foreign except Canada”
- Many XML security issues also arose with object DBs
  - E.g., IS-A, part-of
  - Arise also with SQL’s object constructs

## XML security research examples

- Several models to protect XML documents e.g., Bertino, Damiano
  - Factor the problem to exploit: X-languages, query processing, SQL security, temporal/spatial data types
- Policy partly at schema level, partly instance-specific
  - Accommodate nesting and other XML properties
- Efficient processing of schema-level labels
  - E.g., twig queries with MLS labels [Cho et al.]
    - Asking administrators to specify more goes against the trend toward zero-administration
    - Is MLS realistic there? DoD will not mix major levels on same system. What if labels are not totally ordered?
- Protect schema-less documents
  - Use IR to derive document attributes

## Semantic web languages

- RDF and OWL are likely to become important, even if the ambitious vision remains elusive
  - RDF offers schema-less entry of individual facts, natural labeled “graph” structures
    - Resource is anything on the web
  - OWL adds inference
- There will be a niche for security models optimized for each of XML, RDF, OWL
  - But will they play well together? Will they require duplicate administration? Duplicate software?

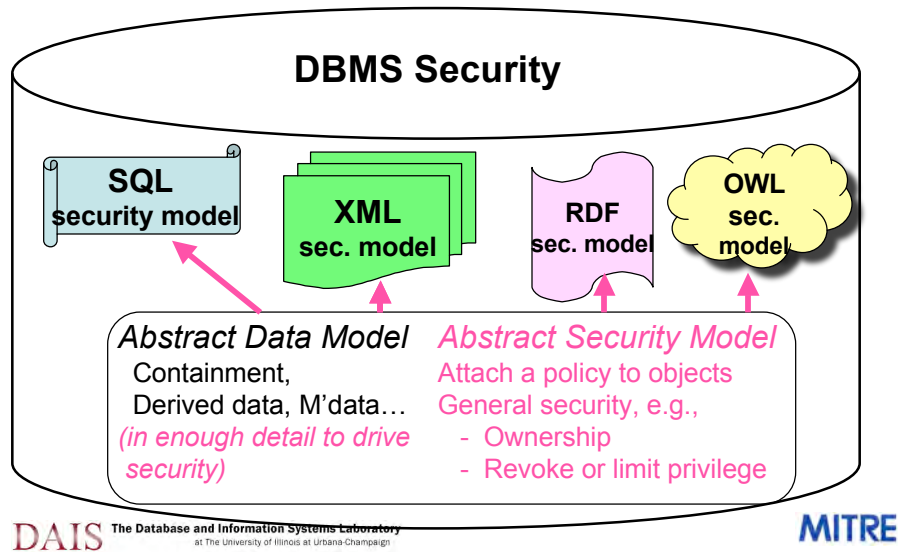


147

## Security for multi-model databases

- DBMSs are becoming dual personality
  - They see (+ store) the same data as relations or XML
  - Support SQL, XPath, XSL, XQuery, ...
- But have separate security systems for each of these, plus RDF, OWL, etc.
- For vendors: Support SQL, XML, RDF, OWL security models on the same code base
- Avoid double administration, inconsistent policy, when crossing model boundaries
  - Translate policies across models
    - To provide consistency regardless of model used to access data
    - To apply policies consistently to subobjects/across links
  - Double enforcement is often OK (e.g., at GUI and trusted)

## How to support multiple security models?



## A possible research approach

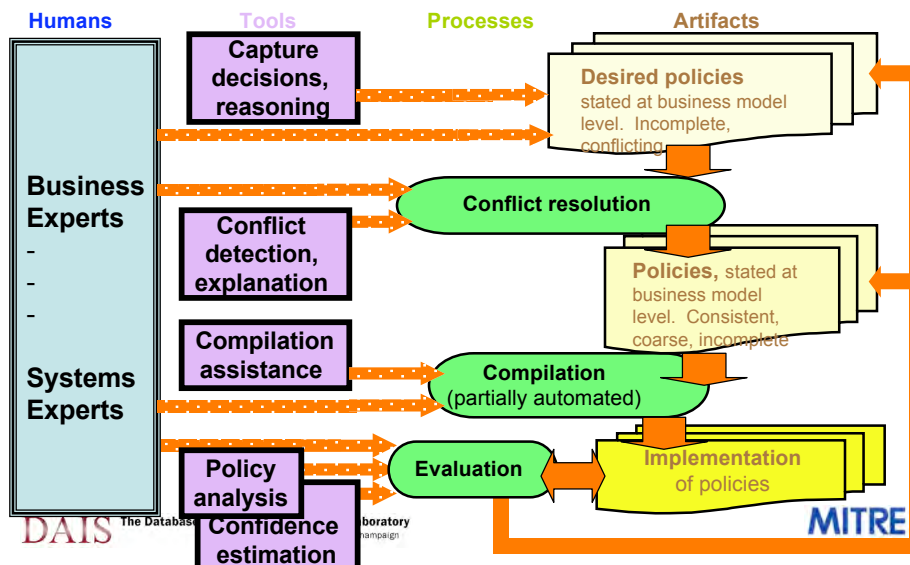
- Devise *rich* object metamodel and map it to SQL, XML
  - Identify common abstractions for models of
    - *data* (metadata, derived object, is-a, part-of, ...)
    - *security* (delegation, revoke, limit privilege, session...)
  - *Cover all objects that SQL protects*
- Avoid gratuitous incompatibility with SQL
  - Where new applications *really* need more, generalize to apply to both models
- Specify and implement the delta, not separate systems

# Agenda

- Introduction
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- **Security issues and opportunities in example application areas**
  - Trust management in open systems
  - DB capabilities for data that *really* needs to be secure
  - Semantic web and XML
  - **Enterprise security**
    - From high level statement to implementation
    - Between organizations

what an enterprise needs:

## Policy engineering environment





# Reprise: Compile “business” policies to physical implementation

153

Individually identified medical data shall be available only to professionals treating the patient, with medium-high confidence



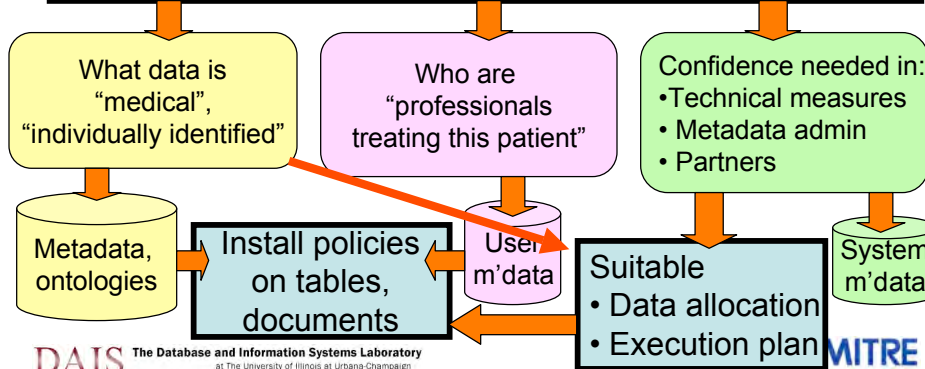
Install policies on tables, documents

Suitable  
• Data allocation  
• Execution plan

# Compile “business” policies to physical implementation

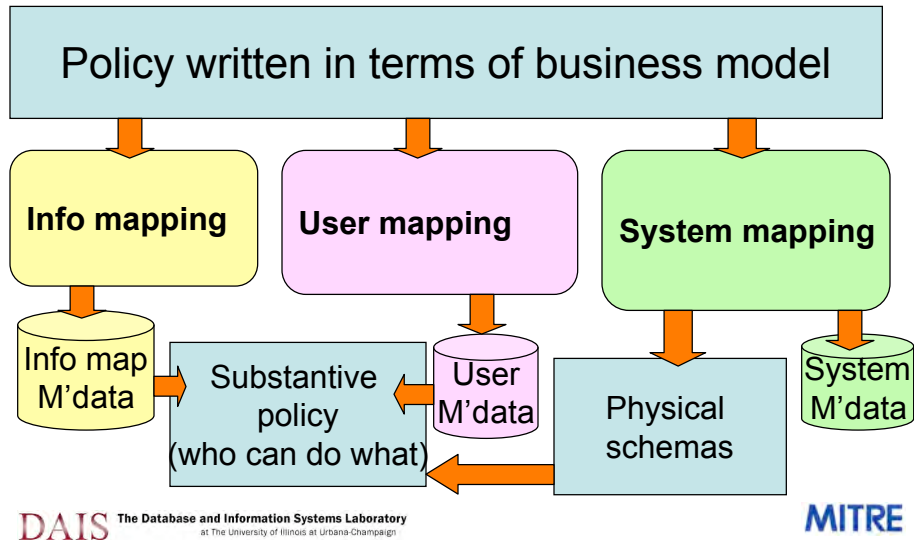
154

Individually identified medical data shall be available only to professionals treating the patient, with medium-high confidence



## Subject/object/action each require own set of mappings

155



## Kinds of mappings needed

156

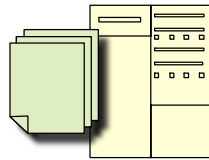
- Down: compile from policy specification to implementation
    - Up: Reverse engineer a rough high level policy from a detailed policy
      - Analogy: derive ER schema from relational schema
  - Down: allocate data and execution, for suitable confidence (next slide)
  - Horizontal: translate policies between organizations, data models (later)
- Giant Opportunity(?): Use same underlying theory and/or implementation for all?**

## Physical DB design problems

### Physical design systems need to know about security req's

- Organizations partition data to minimize the number of users “close” to sensitive info
  - Partition among machines, DBMSs (some not on internet), tables, tuples, ... (increasingly easy to hack)
  - Which systems are trusted to filter what data in queries
  - Select appropriate communications (e.g., encrypt wireless)
  - Enhance data allocators, query planners to provide necessary confidence
- Index securely
  - Imagine the risk of having one file with a full text index for an entire intelligence agency
    - Encryption works in some cases. Will it make the system brittle?
  - How to partition indexes for security?

## Reprise: Translate and transfer policy across organizations and systems

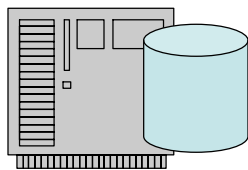


### **Aetna Travel Insurance**

*Enforcement: Application server*

*Policy applied: US (NY)*

*Roles: HIPAA spec (Aetna version)*



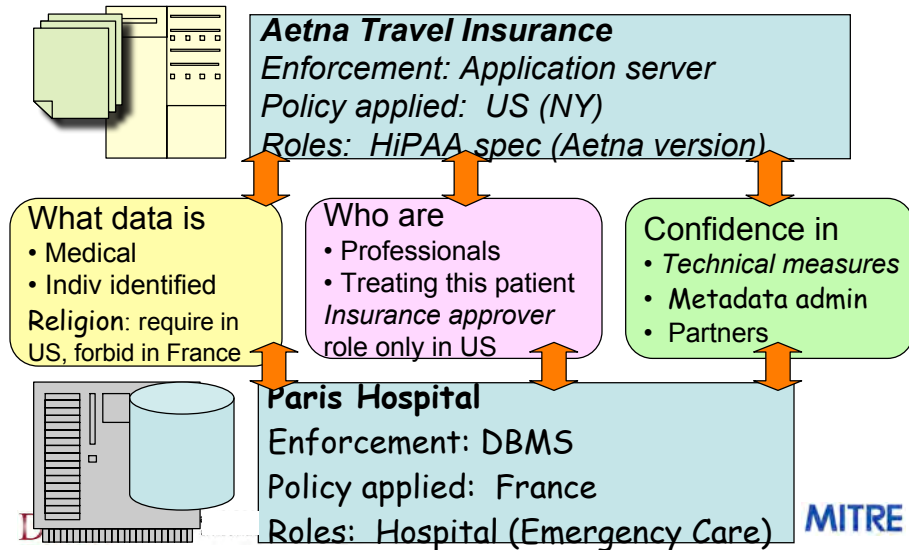
### **Paris Hospital**

*Enforcement: DBMS*

*Policy applied: France*

*Roles: Hospital (Emergency Care)*

## Translate and transfer policy across organizations and systems



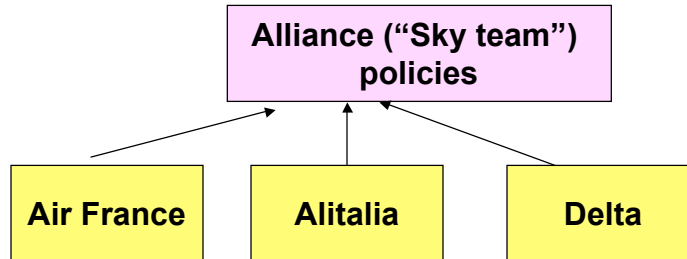
## Policy translation (horizontal mappings)

- Agencies won't share unless they approve the partner's protections. Each has its own policies
- How to enforce X's policies in Y's domain, overcoming differences in data and security?
  - Data: structures, query operators, instance identification [ss#, emp#], schema, ...
  - Security: model, policy language, policy implementations
- How to explain to X what Y is enforcing, and the difference?

*Impression: 70% of semantic integration challenges have security analogs*

- Semantic integration seems to precede security integration

## Policy *integration* challenges



Integration challenges: conventional semantic integration plus:

- Integrate role & group hierarchies
- Integrate policies

*Consider OWL as a common formalism*

## Research areas applicable to mapping of security specs

- Semantic modeling
- Query processing for federated heterogeneous databases
  - Secrecy-friendly algorithms
- Automated physical design
- Model management theory [Bernstein]

## Preventing disclosure during info integration: a contrarian view

### *Skeptical notes*

- How often is such high confidence *essential*?
- Exact match rarely works for names!
- Do we want to treat these queries different from all others?

### *Unifying perspective* (U. Maurer @ SIGMOD04)

- We can do it all with a “trusted subject” in the middle
  - Cryptography is *one* way to create a trusted subject.
  - Other techniques may be more flexible or efficient, but lower confidence, e.g., a trusted SQL DBMS appliance
- Start with the policy to be enforced:
  - What may be revealed to what system
  - Describe what they’re trusted to do, and how confident we’ll be

## Agenda

- Introduction
- Security basics
- State of the art and open problems
- Policies as a unifying framework
- Security issues and opportunities in example application areas
- **Summary**

## Summary advice, 1

- The big wins occur when tools drive the cost of something to *zero* (not 50%)
  - Compile specs (functional + implementation properties) to an implementation
  - Automate “where are we” and “what if” analyses
- Exploit common abstractions for multi-lingual security
  - Containment, IS-A, derived data, delegation, ...
- Extend SQL smoothly -- do not be gratuitously different
  - Feature interactions, granularity differences appear when constructs are examined in full context

## Summary advice, 2

- Security should not be a stovepipe
  - Reuse existing concepts from query languages/ derived data/..., rather than reinventing them
  - Security components that can be reused (services, policies)
- Rich policies need rich runtime input
  - General data access and exchange services, federated DB capabilities will be needed at run time to feed into policy decisions
- Trust models are *broadly* relevant to data quality and suitability

## Summary advice, 3

- Security system has high requirements for data integrity, availability, threat resistance
  - Could build DBMSs to treat these as “normal” requirements (i.e., to provide high integrity, availability, threat resistance)
- First define *correctness* criteria. Do algorithms afterward
  - e.g., for role hierarchy integration, privilege inference rules

## Some relevant further reading

- Policies
  - IEEE Policy Conference
- Data security
  - Conferences: IFIP 11.3, ACM SACMAT
    - Modeling and analysis weak by SIGMOD standards
  - Journals: ACM TISSEC
  - Books: Castano et al. 1995, for earlier research
- General security
  - Textbooks: many choices
  - Conferences (systems-oriented): CCS, NDSS, Oakland
    - Mostly aim at securing systems and system access



## What problems receive too much attention, in unreal settings?

### *Inference control (1990s)*

- Limiting the ability of a party to use additional knowledge to figure out things that they have not been told explicitly
- Administration prerequisites are daunting
  - Need fine grained policies (e.g., columns, not tables)
  - Document adversary's knowledge (logical and statistical)
- System prerequisites
  - Tracking requesters' identity
  - Assuming requesters don't collude
  - Keeping a history of all past requests and responses

## Deserve lower priority, continued

- Conclusion re inference control:
  - Worth doing for carefully examined static publications (census bureau, health statistics)
  - For enterprise systems, it's like locking a 5<sup>th</sup> floor window
  - Research on inference control is unlikely to attract vendors, and hence will lack broad real-world impact

### *“Privacy-preserving” data mining*

- The work to date looks costly, fragile
- Probably not a great place for a stampede of researchers unless more practical look is given
- Trusted third party appliances (stand-alone machine & software) could help