

Trust Management A Tutorial

Scott D. Stoller



Outline

- Introduction and Motivation
 - ◆ Traditional policy frameworks
 - ◆ Policy frameworks for decentralized systems
 - ◆ Trust management
- Design Issues and Features
- Trust Management Frameworks
- Sample Application Domains
- Research Directions

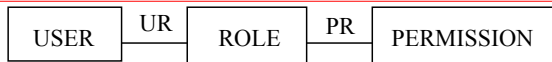
Traditional Frameworks: Access Control Lists

- **ACL**: a list of pairs <principal, allowed operations> associated with a resource.
- **Example**: File permissions in most operating systems
- ACLs do not scale to large systems
 - ◆ **Redundancy**: users working on the same project have many of the same permissions
 - ◆ **Administrative cost**: updating the policy for a new user requires changing many ACLs
 - Easy to make mistakes, giving too many or too few permissions

Traditional Frameworks: Role-Based Access Control (RBAC)

- **Role**: an abstraction associated with a set of permissions, typically associated with a function or position in an organization.
 - ◆ **Examples**: doctor, nurse, patient, receptionist
- RBAC policy specifies:
 - ◆ the roles that each user may adopt
 - ◆ the permissions associated with each role.
 - Permission = [operation, resource]
 - Operation may be resource-specific, not limited to read/write.

RBAC



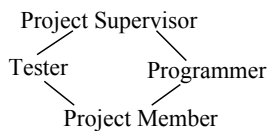
- A user has a permission if he is a member of some role with that permission.
- Benefits of RBAC
 - ◆ Greatly reduces redundancy: there is a set of permissions for each role, not each user
 - ◆ Easy to administer: A new user is added to a few roles.
 - Roles reflect organizational structure and change less frequently.

RBAC: Role Activation

- A user must **activate** a role in a **session** in order to use the permissions associated with that role.
- **Example**: A doctor is sometimes a doctor, sometimes a patient.
- **Analogy**: system administrator sometimes logs in as root, sometimes as a regular user
- Helps enforce the **principle of least privilege**.
- **Limits potential damage** due to human errors, software defects, etc.
- A **session** (a window, application, ...) belongs to one user. Multiple roles may be active in it.

RBAC: Role Hierarchy

- $r1 \geq r2$ ($r1$ inherits from $r2$, $r1$ is senior to $r2$) means every member of $r1$ is also a member of $r2$. thus, members of $r1$ have all the permissions that members of $r2$ have.
- Permission flows up. Membership flows down.
- Role hierarchy further reduces redundancy and eases administration.
 - ◆ New supervisor is added to one role, instead of four.



ACSAC 2005

Scott Stoller, Stony Brook University

7

RBAC: Policy Administration

- **Administrative policy:** security policy that controls changes to the security policy
- In large organization, decentralized policy administration
 - ◆ **Example:** senior admin, junior admin for each division
- **Administrative RBAC (ARBAC)** introduces
 - ◆ **administrative roles** (in addition to regular roles)
 - ◆ **administrative permissions** (for adding and removing users and permissions from regular roles)
- **Example:** project leader can add/remove users/permissions from roles on the project he/she leads.
- Who can change the ARBAC policy? ARBAC doesn't say

ACSAC 2005

Scott Stoller, Stony Brook University

8

Public-Key Infrastructure (PKI)

- **Public-key certificate:** a certificate signed by a **certification authority (CA)**, containing a public key K and a principal's name N , and stating that K belongs to N .
- **Example:** [public key 1a4deb6c5c belongs to AMA] signed by VeriSign
- Who is trusted to **issue** public-key certificates?

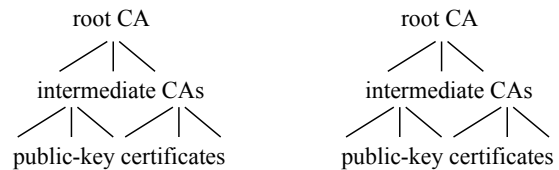
ACSAC 2005

Scott Stoller, Stony Brook University

9

Public-Key Infrastructure: X.509

- X.509 has a **hierarchical trust model**
- Trust is rooted at **root CAs**.
 - ◆ A list of them is embedded in your web browser.
- A CA can also issue certificates certifying other entities as CAs. This creates a **forest** of trust relationships.



ACSAC 2005

Scott Stoller, Stony Brook University

10

Public-Key Infrastructure: PGP

- **Pretty Good Privacy (PGP)** is based on a **web of trust**.
- Everyone may issue public-key certificates.
- Each user specifies a **level of trust** in each issuer.
- Each user specifies the **total confidence** needed for a public-key↔name relationship to be considered **valid**.
 - ◆ **Example:** one certificate from an issuer trusted at level 10, or certificates from two distinct issuers trusted at level 5 or higher.
- A user can also build confidence in such a relationship through successful communication using a particular public key.

ACSAC 2005

Scott Stoller, Stony Brook University

11

Public-Key Infrastructure: X.509 vs. PGP

- X.509's hierarchical trust is appropriate for **e-commerce**.
 - ◆ Accountability
 - ◆ Structure: known sources for certificates, revocation lists, etc.
- PGP's web of trust is appropriate for **personal communication**.
 - ◆ Individuals will not spend time and money to get VeriSign certificates (\$795 for a 3-year certificate for 40-bit encryption)
 - ◆ In current practice, authentication of personal communication is enforced mainly through non-technical means.

ACSAC 2005

Scott Stoller, Stony Brook University

12

Essential features of policy frameworks for decentralized systems: attributes and relations

- Policy can use application-specific **attributes and relations**.
- **Example:** Nurses in the workgroup treating a patient can access the patient's medical record.
 - ◆ **Attributes:** isNurse(employee)
 - ◆ **Relations:** treatingWorkgroup(patient,group).
- Encoding such policies as **identity-based** policies is
 - ◆ **impractical:** potential users are not known to resource owners in advance
 - ◆ **dangerous:** attributes can change
- RBAC is identity-based.

ACSAC 2005

Scott Stoller, Stony Brook University

13

Essential features of policy frameworks for decentralized systems: attributes and relations

- Attributes and relations can be defined in terms of other attributes and relations.
 - ◆ **Example:** Nurses in the workgroup treating a patient can access the patient's medical record. A nurse is in the workgroup if a manager assigned him/her to it.
- This allows interactions that are essential in decentralized systems.
- Standard RBAC does not support this. Each role is defined independently (aside from inheritance).
 - ◆ **Example:** RBAC does not support policies like $role1.members = role2.members \cap role3.members$

ACSAC 2005

Scott Stoller, Stony Brook University

14

Essential features of policy frameworks for decentralized systems: delegation

- Policy administration is **completely decentralized** at the top level. No globally-trusted administrators. No root of trust.
- Policies interact through **delegation**.
- **Example:** Hospitals, doctor's offices, insurers, and government agencies share information (medical, financial, and personnel records). They trust each other in limited ways.
- **Example:** Conference gives a discount to students enrolled at accredited universities. Conference trusts universities for enrollment information.
- **Example:** Military coalitions.

ACSAC 2005

Scott Stoller, Stony Brook University

15

Essential Features of Trust Management

- Each policy statement is associated with a principal, called its **source** or **issuer**.
- Each principal's policy specifies which **sources it trusts** for which kinds of statements, thereby **delegating** some authority to those sources.
- Policies may refer to domain-specific **attributes** of and **relationships** between principals, resources, and other objects.
- **Example:** Acme Hospital says "doc can access pat's medical record if AMA says doc is a licensed doctor and pat says doc is treating him." (patient consent)

ACSAC 2005

Scott Stoller, Stony Brook University

16

Simple Rule-Based Trust Management Language

- Essentially Datalog. Start simple. Extend later.
- **Atom:** issuer.relation(arguments)
- **Argument:** constant, variable, or constant(arguments)
 - ◆ **Relation** names and **variables** start with **lowercase**.
 - ◆ **Constants** start with **uppercase**.
 - ◆ Restrict the use of arguments so constants have bounded depth. In other words, allow tuples, not lists.
- **Rule:** atom :- atom1, atom2, ...
 - ◆ If atom1 and atom2 and ... hold, then atom holds.
- **Fact:** a rule with no hypotheses.
- **Policy:** a collection of facts and rules.

ACSAC 2005

Scott Stoller, Stony Brook University

17

Simple Trust Management Language: Example

- By convention, **issuer.allow(principal, operation(resource))** means **issuer** authorizes **principal** to perform **operation** on **resource**. Notation: similar to [Becker+ 2004].
- The default issuer of an atom in a rule is the owner of the policy database containing the rule.
- Acme Hospital says "doc can access pat's medical record if AMA says doc is a licensed doctor and pat says doc is treating him."
**AcmeHospital.allow(doc, Read(EPR(pat)) :-
AMA.doctor(doc),
pat.consentToTreatment(doc).**

ACSAC 2005

Scott Stoller, Stony Brook University

18

Simple Trust Management Language: Example

- SUNY says its employees can read the campus directory.
- `SUNY.allow(e, Read(Directory)) :- SUNY.employee(e)`
- SUNY says X is a SUNY employee if a SUNY campus says X is a campus employee.
`SUNY.employee(e) :- SUNY.campus(c), c.employee(e)`
- In this example, the conclusion of a rule is used as a premise of another rule.
- Variables that appear in premises and not in the conclusion are, in effect, **existentially quantified**.

Compliance Checking: Bottom-Up Algorithm

- Input:** a fact (the goal). **Output:** whether the goal is derivable.
 Boolean derivable(goal)
 while there exist
 rule "c :- p1,p2,..." in policy and
 facts f1, f2, ... in policy and **substitution** σ
 such that $\sigma(p1)=f1, \sigma(p2)=f2, \dots$ and $\sigma(c)$ not in policy
 add $\sigma(c)$ to policy
 return (goal in policy ? true : else)
- Pretend for now that policy is a global set.

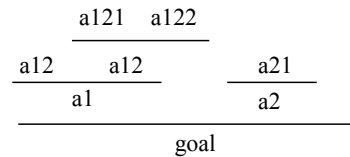
Compliance Checking: Goal-Directed Alg.

```

Boolean derivable(goal)
for each rule r and substitution  $\sigma$  s.t.  $\sigma(r$ 's conclusion)=goal
for each premise p of r
  if derivable( $\sigma(p)$ ) continue; // try next premise
  else break; // this rule failed; try next rule
return true // we proved the goal using this rule
// no rule succeeded
return false
  
```

Proof of Compliance

- The goal-directed algorithm can easily be extended to provide a **proof** that the goal is derivable.
- A **proof** is a tree formed from instantiated rules, with facts from the policy at the leaves, and with the goal at the root.



Goal-Directed Algorithm: Tabling

- The simple goal-directed algorithm on previous slide may:
 - re-derive the same goal many times
 - **Example:** a12 and a21 could be the same.
 - diverge on recursive policies
- Goal-directed evaluation with **tabling**:
 - Cache all derived goals.
 - Look in the cache for an existing goal that unifies with the new goal before attempting to derive the new goal.

Outline

- Introduction and Motivation
- **Design Issues and Features**
 - Bounds on Re-Delegation
 - Proof Search
 - Credential Gathering
 - Policy Changes
 - Trust Negotiation
 - Constraints
 - External Data
 - Separation of Duty
 - Separation of Privilege
 - Roles
 - Global and Local Names
- Trust Management Frameworks
- Sample Application Domains
- Research Directions

Re-Delegation

- If A delegates a permission to B, can B re-delegate it to C?
- **Example:** Conference's policy for reviewing papers
- A PC member can submit a review of a paper.
- `allow(pcmem, Submit(Review(p))) :- PCmember(pcmem)`
- A PC member can designate a subreviewer.
- `allow(subrev, Submit(Review(p))) :- PCmember(pcmem), pcmem.subreviewer(subrev)`
- Can subreviewer S1 delegate to sub-sub-reviewer S2? No.
- `S1.subreviewer(S2)` doesn't work, because `Conf.PCmember(S1)` doesn't hold.
- S2 could write S1's review, though.

Re-Delegation

- Re-delegation is allowed if relations are defined recursively.
- **Conf:** `allow(rev, Submit(Review(p))) :- PCmember(rev)`
- If `rev` can submit review, `rev` can designate subreviewer.
- `allow(subrev, Submit(Review(p))) :- allow(rev, Submit(Review(p))), rev.allow(subrev, Submit(Review(p)))`
- This allows delegation chains of arbitrary length.
- To allow delegation chains up to a specified length, use a subreviewer relation parameterized by the allowed delegation depth.

In compliance checking, who searches for proof?

- **Resource owner**, i.e., the policy enforcement mechanism
 - Example: medical records database server
- **Requester** (needs resource owner's policy) [Bauer+05]
 - Appropriate for embedded devices
 - **Example:** Lock with Bluetooth on Mike's office door.
 - The lock's policy is: `allow(e, Open()) :- Mike.allow(e, Open(OfficeDoor))`
 - e's cell phone needs to present a proof of `Mike.allow(e, Open(OfficeDoor))`.
 - The cell phone can communicate with Mike and his delegates. The lock can't.

Credential Gathering

- **Credential:** signed certificate containing a policy statement, usually a fact (in some systems, a fact or rule).
- To **import** a credential `[iss.r(args)] signed by K`: if `K` is `iss`'s key, and the signature is valid, then add `iss.r(args)` to the policy; otherwise, the credential is invalid.
- Compliance checking requires credentials for all subgoals with remote issuers.
- **Example:**
`AcmeHospital.allow(doc, Read(EPR(pat))) :- AMA.doctor(doc), pat.consentToTreatment(doc).`

Where To Get Credentials?

- From issuer
 - **Example:** request `AMA.doctor(doc)` from AMA
- From requester
 - **Example:** request `AMA.doctor(doc)` from doc
 - doc may have it or may request it from AMA.
- From a location specified in the policy (instead of hard-coding the decision in the evaluation algorithm).
 - Details on the next slide.

Policy-Directed Credential Gathering

- Each **premise** is labeled with a **location** (e.g., an Internet address) where the credential should be obtained. Location can be a variable. Default location is issuer.
- **Notation:** `location@issuer.relation(args)`.
- **Example:** request AMA.doctor credential from doctor.
`AcmeHospital.allow(doc, Read(EPR(pat))) :- doc@AMA.doctor(doc), pat.consentToTreatment(doc).`
- **Example:** request AMA.doctor credential from AMA.
`AcmeHospital.allow(doc, Read(EPR(pat))) :- AMA@AMA.doctor(doc), pat.consentToTreatment(doc).`
- Can include both of these rules in the policy.

Policy Changes

- Derived facts may be cached locally and may be sent in credentials and cached at other sites, for efficiency and fault-tolerance.
 - Example:** Hospital caches `AMA.doctor` credential.
- These facts may become invalid due to:
 - Deletion** of facts or rules.
 - Addition** of facts or rules, if the policy language is **non-monotonic** (adding a fact or rule can invalidate facts), i.e., can express negation or equivalent.
- This is a standard problem with caching in distributed systems. Standard solutions, such as **expiration dates** or **revocation lists**, can be used.

ACSAC 2005

Scott Stoller, Stony Brook University

31

Trust Negotiation: Gradual Release of Sensitive Credentials

- Credentials may contain **sensitive** information.
- Example** [Bhargava+ 2004, Winsborough+ 2004]: On-Line University (OLU) gives a discount to veterans. Joe reveals his veteran status only to IRS-certified non-profits.
 - Joe** → **OLU**: `register(CS101)`
 - OLU** → **Joe**: `requestCredential: VA.veteran(Joe)`
 - Joe** → **OLU**: `requestCredential: IRS.nonProfit(OLU)`
 - OLU** → **Joe**: `IRS.nonProfit(OLU)`
 - Joe** → **OLU**: `VA.veteran(Joe) // OLU: give discount`
 - OLU** → **Joe**: `requestPayment: $1000`
 - Joe** → **OLU**: `Citigroup.creditCard(Joe,1234-5678-9012)`

ACSAC 2005

Scott Stoller, Stony Brook University

32

Trust Negotiation: Privacy Policy for Credentials

- Suppose we associate privacy policies with **credentials**.
Example [Winsborough+ 2004]:
 - Joe reveals his low-income credential only to non-profits.
 - Joe** → **E-Realty**: `request house listings`
 - E-Realty** → **Joe**: `requestCredential: IRS.lowIncome(Joe)`
 - Joe** → **E-Realty**: `requestCred.: IRS.nonProfit(E-Realty)`
 - E-Realty is not non-profit. Rich is not low-income.
 - Rich** → **E-Realty**: `request house listings`
 - E-Realty** → **Rich**: `requestCred.: IRS.lowIncome(Rich)`
 - Rich** → **E-Realty**: `nothing`
 - E-Realty infers (no proof) Joe is low-income, Rich isn't.

ACSAC 2005

Scott Stoller, Stony Brook University

33

Trust Negotiation: Privacy Policy for Attributes

- Associate a privacy policy with each **attribute**, regardless of whether user has that attribute or a credential for it.
 - Joe** → **E-Realty**: `request house listings`
 - E-Realty** → **Joe**: `requestCredential: IRS.lowIncome(Joe)`
 - Joe** → **E-Realty**: `requestCred.: IRS.nonProfit(E-Realty)`
 - Rich is not low-income but has the same privacy policy.
 - Rich** → **E-Realty**: `request house listings`
 - E-Realty** → **Rich**: `requestCred.: IRS.lowIncome(Rich)`
 - Rich** → **E-Realty**: `requestCred.: IRS.nonProfit(E-Realty)`
 - E-Realty learns nothing.

ACSAC 2005

Scott Stoller, Stony Brook University

34

Where to Get Privacy Policies for Attributes?

- Where** does Rich get the privacy policy for `IRS.lowIncome`? Perhaps Rich never heard of `IRS.lowIncome` before E-Realty asked about it.
- Issuers** should provide **standard privacy policies** for attributes, at well-known locations [Winsborough+ 2004].
- Example:** When Rich sees request for `IRS.lowIncome` credential, he contacts IRS policy server and obtains and uses the IRS-recommended privacy policy for attribute `IRS.lowIncome`.
- If Rich doesn't bother to do this, then other people probably won't do it for other attributes, which Rich might want to keep private.

ACSAC 2005

Scott Stoller, Stony Brook University

35

Trust Negotiation Strategies

- Trust negotiation **policy** determines when a credential **may** be released.
- Trust negotiation **strategy** determines which releasable credentials **are** released.
 - Eager Strategy:** at each step, send all releasable credentials.
 - Targeted Strategy:** at each step, send releasable credentials that help achieve the current goal.

ACSAC 2005

Scott Stoller, Stony Brook University

36

Trust Negotiation Strategies: Example

- **Eager Strategy** (fewer rounds of communication):
- **Joe** → NP-Realty: request house listings
- NP-Realty → Joe: requestCred.: IRS.lowIncome(Joe); IRS.nonProfit(NP-Realty), BBB.member(NP-Realty), ...
- **Joe** → NP-Realty: IRS.lowIncome(Joe)
- **Targeted Strategy** (fewer credentials sent):
- **Joe** → NP-Realty: request house listings
- NP-Realty → Joe: requestCred.: IRS.lowIncome(Joe)
- **Joe** → NP-Realty: requestCred.: IRS.nonProfit(NP-Realty)
- NP-Realty → Joe: IRS.nonProfit(NP-Realty)
- **Joe** → NP-Realty: IRS.lowIncome(Joe)

ACSAC 2005

Scott Stoller, Stony Brook University

37

Constraints

- **Constraint**: a premise that uses an externally defined relation on a data type. Common examples include:
- **Numerical inequalities**
 - ◆ **Example**: allow(empl,Read(file)) :- securityLevel(empl,m), securityLevel(file,n), $m \geq n$.
- **Prefix-of** relation on sequences (e.g., pathnames)
 - ◆ **Example**: allow(stu, Read(file)) :- Registrar.enrolled(stu, CSE101), /CSE101/exam/ prefix-of file.
- These relations can't be defined in Datalog.

ACSAC 2005

Scott Stoller, Stony Brook University

38

External Data

- Policy may depend on **external data**.
- **Example**: personnel database: employees, their department and rank
- **Example**: EHR database: author of each entry
- Storing this info in **policy database** would be inefficient.
- How does the policy access it?
 - ◆ Request **credentials** from the DBMS. This is inefficient and unnecessary, assuming DBMS is local and trusted.
 - ◆ Use a **connector** that makes the DBMS look like part of the policy database. Neat, because policy language and DBMS are both relational.

ACSAC 2005

Scott Stoller, Stony Brook University

39

External Data: Connector to DBMS

- Each **table** corresponds to a **relation**.
- Each **record** corresponds to a **fact**.
- Connector generates **SQL queries** to retrieve relevant data.
- **Example**: allow(e, read(Budget(dept)) :- deptSeniorPers(sp, dept), sp.allow(e, read(Budget(dept))).
- sp is **unbound** when deptSeniorPers is evaluated.
- If deptSeniorPers is external, the generated **SQL query** finds and returns all senior personnel of the department.
- If results from DBMS are **cached**, they must be **invalidated** if a **DBMS update** changes the relevant data.

ACSAC 2005

Scott Stoller, Stony Brook University

40

External Functions

- Manipulate data
 - ◆ **Example**: selectors for compound data structures
- Provide environment and context information
 - ◆ **Example**: allow(stu, Read(file)) :- Registrar.enrolled(stu,CSE101), /CSE101/exam/ prefix-of file, currentTime() > 09:00.7dec2005.
- Provide simple interface to external data (file, DBMS, ...).
 - ◆ Arguments must be ground (constants) at call time.
 - ◆ **Example**: Note: author is an external function.
allow(e, Update(rec)) :- employee(e), e=author(rec).

ACSAC 2005

Scott Stoller, Stony Brook University

41

Object-Based Separation of Duty

- Separation of duty limits the set of permissions of a single user. This helps prevent fraud, which requires collusion.
- **Example**: A single employee may perform at most 1 of the 3 steps involved in a purchase: issue purchase order, verify receipt of goods, issue payment.
- **Object-based separation of duty** allows an employee to perform at most 1 of these operations for a single purchase.
- **Example**: allow(e, IssuePayment(trans)) :- acctgClerk(e), e ≠ getPurchClerk(trans), e ≠ getRcvClerk(trans)
- getPurchClerk(trans): clerk who issued the PO for trans

ACSAC 2005

Scott Stoller, Stony Brook University

42

Separation of Privilege

- **Separation of privilege:** an action is permitted only if a specified number of authorized users request it.
- `issuer.allow2(principal1,principal2,operation(resource))` means `issuer` authorizes `principal1` and `principal2` jointly (together) to perform `operation` on `resource`.
- **Example:** `allow2(clerk, mgr, IssuePayment(amount)) :- AcctgClerk(clerk), AcctgManager(mgr), clerk ≠ mgr, amount < 1,000,000. allow(clerk, IssuePayment(amount)) :- AcctgClerk(clerk), amount < 10,000.`
- **Alternative:** Decompose the action into multiple actions.
- **Example:** `clerk: InitiatePayment, mgr: ApprovePayment.`

Roles

- **Parameterized role:** `r(args)`. Abbreviate `r()` as `r`.
- **Example:** `Manager(department), Guardian(patient)`
- “`p` is a member of `r(args)`” can be represented as
 - ◆ `r(p, args)` roles as relations
 - ◆ `member(p, r(args))` roles as values
- Permission-role relation is defined by rules like:
 - ◆ `permit(p, oper(resource)) :- r(p,args), ...`
 - ◆ `permit(p, oper(resource)) :- member(p,r(args)), ...`
- Roles as values allows variables that range over roles, but this can be simulated with roles as relations.

Role Hierarchy

- **Role hierarchy** can be expressed by rules for inheritance of membership.
- **Example:** `Manager ≥ Employee` is expressed by `member(e, Employee) :- member(e, Manager).`
- Role hierarchy could be expressed instead by rules for inheritance of permissions if we made the permission-role relation `PR(action,role)` explicit.

Global and Local Names

- **Local names:** names of attributes and relations include the name of a principal, called its `source` or `issuer`.
 - ◆ **Example:** `AMA.doctor(Dan), BMA.doctor(Dan)`
 - ◆ Statements about `src.r` may be issued only by `src`.
- **Global names:** shared namespace for attributes & relations.
 - ◆ Less structured, but more flexible.
 - ◆ **Example:** `AcmeHosp.member(Dan, Doctor(AMA))`
- **RT** [Li+ 2003]: local. **Cassandra** [Becker+ 2004]: global.
- An **ontology** can provide common meaning for names.
- Local names for **principals**, e.g., [SBU President]. Used in **SPKI/SDSI**. Can be simulated using parameterized roles.

Outline

- Introduction and Motivation
- Design Issues and Features
- **Trust Management Frameworks**
 - ◆ List of several proposed frameworks on next slide.
 - ◆ We'll discuss a few representative frameworks.
- Sample Application Domains
- Research Directions

Some Trust Management Frameworks

- **PolicyMaker** and **Keynote** [Blaze, Feigenbaum, et al., 1996-1999]
- **SPKI/SDSI** [Rivest, Lampson, Ellison, Frantz, Thomas and Ylonen, 1997-1999]
 - ◆ Simple PKI / Simple Distributed Security Infrastructure
- **Delegation Logic, RT** [Li et al., 2000-present]
- **SD3** [Jim 2001]
- **Binder** [DeTreville 2002]
- **TrustBuilder** [Seamons, Winslett, et al., 2002-present]
- **PeerTrust** [Nejdl, Olmedilla, et al., 2003-present]
- **Cassandra** [Becker and Sewell, 2004-2005]

PolicyMaker [Blaze, Feigenbaum, et al]

- PolicyMaker is a blackboard-based trust management **architecture**. The **blackboard** contains
 - ◆ requests (goals): action/statement to be authorized
 - ◆ acceptance records: issuer allows action/statement
- **Policy**: functions that read requests and acceptance records from the blackboard and write acceptance records.
 - ◆ Use any safe functional programming lang. (SafeAWK)
- PolicyMaker is **flexible** but offers **minimal functionality**.
 - ◆ Application gathers credentials, verifies signatures, etc.
 - ◆ AWK interpreter (or ...) evaluates policy functions

SPKI/SDSI [Rivest, Lampson, et al.] Name Certificates

- Local names for principals. The meaning of local names is given by name certificates that relate local names to each other and to global identifiers (public keys).
- **Format**: [K, name, subject, validitySpec] signed by K
- **Meaning**: local name K name refers to subject
- **subject** may be a name or a public key
- **Example**: [K-SBU-CS, Chair, K-Ari, exp. june 2007]
- A local name mapped to multiple keys is a group name.
- **Example**: [K-SUNY, Student, K-Joe, exp. may 2006]
[K-SUNY, Student, K-Mary, exp. may 2006]
- **validitySpec**: expiration date, CRL location, ...

SPKI/SDSI: Authorization Certificate

- **Format**: [K, subject, deleg, tag, validitySpec] signed by K
 - ◆ Not using official syntax.
- **Meaning**: issuer K gives permission tag to subject.
 - ◆ deleg indicates whether subject can delegate the permission (in addition to using it himself).
- **subject** may be a name (defined by other certificates), a public key, a threshold structure, or an object hash (ignore)
 - ◆ A **threshold structure** [{K1,K2,...}, n] means any n of the listed keys can together authorize the delegated action (separation of privilege).

SPKI/SDSI: Authorization Certif. Examples

- [K-SBU, [K-SBU Faculty], false, readDir, exp. 2006]
- **Example with delegation**:
 - [K-SBU, [K-SBU Faculty], true, (getRoster *), exp. 2006]
- **Name Certificate**: [K-SBU, Faculty, Scott, exp. 2009]
- [K-Scott, K-Seung, false, (getRoster CSE394), exp. 2006]
 - ◆ Seung is the TA. He can't delegate this permission.
- Authorization certificates define **one relation**: allows (delegates).
- Role-based policies can be expressed using groups

Limitations of SPKI/SDSI

- Delegation and authorization and not distinguished: a principal must have a permission in order to delegate it.
 - ◆ **Example** [De Treville 2002]: DMV must be a licensed driver in order to be authorized to license drivers.
- Only unary relations on principals, expressed as groups, are supported.
 - ◆ Can't express policies like: "Nurses in the workgroup treating a patient can access the patient's medical record", which uses relation is-treating(pat,grp)
- No **variables** (parameters) in tags. No **conjunction** of groups/attributes. No **trust negotiation**.

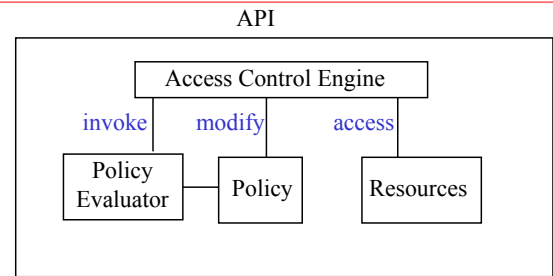
Binder [DeTreville 2002]

- Our simple policy language (without constraints) is very similar to Binder's.
- Authorization relation:
 - ◆ issuer says can(principal,operation,resource)
- **Nicely written** paper; recommended as an **introduction** to rule-based trust management
- Allows **communication of rules** (discussed later), although the details are unspecified
- Does not consider **trust negotiation**.
- http://research.microsoft.com/research/pubs/view.aspx?tr_id=545

Cassandra [Becker and Sewell, 2004-2005]

- Policy language: Datalog + constraints + aggregation
 - ◆ Aggregation allows non-monotonic policies
- Role-based
- Parameterized roles, parameterized actions (permissions)
- Global names (local names can be simulated using issuer and other parameters)
- Goal-directed evaluation with memoization (tabling)
- Policy-controlled credential gathering
- Role activation (but not sessions; details below)
- Trust negotiation
- External functions

Cassandra Architecture



- Policy: local policy + cached credentials
- Alternative architecture: return authorization decisions to the application

Cassandra Predicates

- **Syntax for Predicates:** `location@issuer.pred(args)`
- **Special Predicates** (special significance in the API):
- `permits(entity,action)`: entity is authorized to perform action
- `canActivate(entity, role)`: entity can activate (is a member of) role
- `hasActivated(entity,role)`: entity has activated role
 - ◆ Role activations are recorded as facts in the policy.
 - ◆ No explicit notion of sessions; implicitly, there is one session per Cassandra instance.

Facts as Role Activations

- Many kinds of facts are expressed as role activations. An entity says `fact(args)` by activating the "role" `fact(args)`.
 - ◆ This moderately simplifies the API.
- **Example:** A patient consents to treatment by Dr. Dan by activating the role `ConsentToTreatment(Dan)`.
- **Example:** The manager of department `dept` appoints an employee `e` in `dept` by activating `AppointEmployee(e,dept)`
- `canActivate(mgr, AppointEmployee(e, dept)) :- hasActivated(mgr, Manager(dept))`
- `canActivate(e, Employee(dept)) :- hasActivated(mgr, AppointEmployee(e, dept))`

Cassandra Predicates: canDeactivate

- `canDeactivate(entity,entity1,role)`: entity is authorized to deactivate entity1's activation of role
- **Example:**

```

canDeactivate(pat,pat,ConsentToTreatment(doc)) :- true.
canDeactivate(grdn,pat,ConsentToTreatment(doc)) :-
  hasActivated(grdn,Guardian(pat)).
            
```
- Cassandra does not consider administrative policy, so there is no notion of who is authorized to remove an entity from a role (by changing the policy).

Cassandra Predicates: isDeactivated

- `isDeactivated(entity,role)`: entity's activation of role is being deactivated. Used to trigger other role deactivations.
- **Example:** A user being removed from the `Employee` role should also be removed from the `Manager` role.


```

isDeactivated(e, Manager()) :-
  isDeactivated(e, Employee()).
            
```
- **Example:** A doctor being removed from "on duty at hospital" should also be removed from "attending doctor".


```

isDeactivated(doc, AttendingDoctor(pat)) :-
  isDeactivated(doc, OnDuty()).
            
```
- Could add premise `hasActivated(doc,AttendingDoctor(pat))`

Cassandra Predicates: canRequestCredential

- This predicate expresses **trust negotiation** policy.
- `canRequestCredential(entity, issuer.r(args))`: entity is authorized to request credentials that match `issuer.r(args)`.
 - ◆ Abbreviate as `canRequestCred`
- **Example**: OLU allows a registered student to request a credential showing this.
- `stu` is registered in semester `sem` ↔ `stu` has activated `Student(sem)`.
- `canRequestCred(stu,OLU.hasActivated(stu,Student(sem)))` :- `hasActivated(stu,Student(sem))`.
 - ◆ Response: “**here’s the certif**” or “**unauthorized request**”

Cassandra Predicates: canRequestCredential

- Continuing the example, consider an alternative rule:
- `canRequestCred(stu,OLU.hasActivated(stu,Student(sem)))` :- `true`.
 - ◆ Same effect as previous policy, except response to requests by non-student asking about own status is “**You are not registered as a student.**”
- OLU allows a student’s parent to request that credential.
- `canRequestCred(par,OLU.hasActivated(stu,Student(sem)))` :- `parentOf(par,stu)`.

Cassandra Predicates: canRequestCredential

- A student can delegate authority to get his Student credential to anyone (e.g., potential employer).
- **OLU’s policy**:
`canRequestCred(e,OLU.hasActivated(stu,Student(sem)))` :- `stu.canRequestOLUreg(e)`.
- **Joe Cool’s policy**: `JoeCool.canRequestOLUreg(Google)`.
- An entity can have a `canRequestCredential` policy for credentials (attributes) it does not have.
- **Example**: Every user can have the policy
- `canRequestCredential(c, IRS.lowIncome(y))` :- `IRS.nonProfit(c)`

Cassandra API: doAction, activate

S: site where the operation is invoked.

P: S’s policy.

P |- **fact**: **fact** is derivable from **P**.

e: the entity invoking the operation.

```
e:doAction(a)
  if P |- S.permits(e,a)
  execute action a.
```

```
e:activate(r)
  if P |- S.canActivate(e,r) and not P |- S.hasActivated(e,r)
  add S.hasActivated(e,r) to P
```

Cassandra API: deActivate

```
e:deactivate(e1,r)
if P |- S.hasActivated(e1,r) and P |- S.canDeactivate(e,e1,r)
  add S.isDeactivated(e1,r) to P
D = { [e2,r2] | P |- S.isDeactivated(e2,r2) }
// note: D contains (e1,r)
for [e2,r2] in D
  remove S.hasActivated(e2,r2) from P (if present)
  remove S.isDeactivated(e1,r) from P
```

deActivate: Example

Initial policy P:

```
isDeactivated(e, Manager()) :- isDeactivated(e, Employee())
hasActivated(Mike, Employee())
hasActivated(Mike, Manager())
canDeactivate(Charles, Mike, Employee())
Charles:deActivate(Mike, Employee())
Add isDeactivated(Mike, Employee()) to P.
Then P |- isDeactivated(Mike, Manager()).
So D = { [Mike, Employee()], [Mike, Manager()] }
```

Cassandra API: requestCredential

- may be invoked directly or via a remote premise.
- `iss.r(args)` must be ground. Ignore constraint creds here.

```
e:requestCredential(iss.r(args))
if P |- S.canRequestCredential(e, iss.r(args))
  if iss = S
    if P |- S.r(args) return S.r(args) signed by S
    else return "not S.r(args)"
  else // iss ≠ S. Forward cached credential, if any.
    if P contains iss.r(args) return iss.r(args) signed by iss
    else return "unauthorized request"
```

Cassandra: Constraints and Aggregation

- Constraints over integers, sets, other domains.
- Aggregation operators:
 - Group: collect the facts that match a pattern `S.r(args)` into a set
 - Count: count the number of facts that match a pattern `S.r(args)`
 - Variables in `S.r(args)` not used elsewhere in query may have any value
 - `S` is the local entity; otherwise, answer may be incomplete.

Non-Monotonic Policies

- Aggregation + constraints allow non-monotonic policies
- Example: Dynamic Separation of Duty: no user may have the Doctor and Patient roles active concurrently.
`canActivate(doc, Doctor()) :-`
 `AMA.Doctor(doc),`
 `COUNT(hasActivated(doc, Patient())) = 0`
- This is my own syntax. Cassandra's syntax is more general but harder to read.
- Non-monotonic because adding `hasActivated(Dan, Patient())` changes `canActivate(Dan, Doctor())` from true to false.

Example: Chinese Wall

- To avoid conflict of interest, a consultant can work on at most one project involving each industry sector.
- Example: can't work on projects for Intel and AMD, both in semiconductor sector.
- `industrySector(proj, sec)`: proj involves industry sector `sec`.
- Example: `industrySector(IntelReengg, Semiconductor)`.
- `employeeSector(emp, sec)`: employee `emp` is working on a project in sector `sec`.
- `employeeSector(emp, sec) :-`
 `hasActivated(mgr, AppointEmployee(emp, proj)),`
 `industrySector(proj, sec)`

Example: Chinese Wall

- `canActivate(mgr, AppointEmployee(emp, proj)) :-`
 `Manager(mgr, proj), industrySector(proj, sec),`
 `COUNT(employeeSector(emp, sec)) = 0.`

Outline

- Introduction and Motivation
- Design Issues and Features
- Trust Management Frameworks
- Sample Application Domains
 - Electronic Health Records (EHR)
 - Other application domains
- Research Directions

Electronic Health Records (EHR)

- Promising application for RBAC and trust management
- People and organizations with limited trust must share sensitive information: patients, doctors, nurses, hospitals, billing companies, insurance companies, government agencies (e.g., Medicaid, FDA), professional societies (e.g., AMA), medical researchers, etc.
- More interactive information sharing will increase the need for trust management.
- **Case study [Becker 2005]: Output Based Specification for Integrated Care Record Service**, version 2, 2003. Developed by the National Health Service (NHS) of the United Kingdom.

ACSAC 2005

Scott Stoller, Stony Brook University

73

EHR Policy

- **Spine**: a nationwide EHR system
 - ◆ one electronic health record (EHR) per patient
 - ◆ multiple items per record
- **Registration Authority (RA)**: issues credentials for clinicians, with name, affiliation, specialty, etc.
 - ◆ typically for one organization, but may be regional
- **Local health organizations**: hospitals, doctors' offices, etc.
 - ◆ one electronic patient record (EPR) per patient, with full data
- **Patient Demographic System (PDS)**
 - ◆ One nationwide PDS

ACSAC 2005

Scott Stoller, Stony Brook University

74

Registration Authority Policy

- **Main Role: RA-manager**
- A manager registers a clinician by activating **NHS-Clinician-cert(...)**.
- **canActivate(mgr,**
 NHS-clinician-cert(org, cli, spcty, start, end)) :-
 hasActivated(mgr, RA-manager()),
 hasActivated(y, NHS-health-org-cert(org, start2, end2)),
 start in [start2, end2],
 end in [start2, end2],
 start < end

ACSAC 2005

Scott Stoller, Stony Brook University

75

Spine Policy: Main Roles and Main Actions

- **Clinician**
 - ◆ request consent to treatment, read and update EHR
 - ◆ emergency access to EHR
 - ◆ refer a patient to another clinician
 - ◆ approve requests to seal items
 - ◆ appoint agents for patient (if patient is unable to)
 - ◆ conceal items from patient
- **Administrator**
 - ◆ register new patients, clinicians, and administrators
 - ◆ unregister old ones

ACSAC 2005

Scott Stoller, Stony Brook University

76

Spine Policy: Main Roles and Main Actions

- **Patient**
 - ◆ one-off (i.e., one-time) consent to policy
 - ◆ consent to treatment
 - ◆ appoint agents
 - ◆ seal items in EHR (express consent is required to read them)
- **Agent**: a parent, guardian, spouse, etc., authorized to perform actions on patient's behalf
- **Third party**: a third party whose consent is needed for an action. **Example**: Joe needs his father's consent to access item in Joe's EHR describing his father's cardiac disease.

ACSAC 2005

Scott Stoller, Stony Brook University

77

Spine Policy: Activate Spine-clinician Role

- A NHS-certified clinician can activate Spine-clinician role.
- **canActivate(cli, Spine-clinician(ra, org, spcty)) :-**
 ra.hasActivated(x, NHS-clinician-cert(org, cli, spcty, start,
 end)), // a similar rule has location ra for this premise
 canActivate(ra, Registration-authority()),
 no-main-role-active(cli),
 Current-time() in [start, end]
- **canActivate(ra, Registration-authority()) :-**
 NHS.hasActivated(x, NHS-registration-authority(ra, start,
 end)),
 Current-time() in [start, end]

ACSAC 2005

Scott Stoller, Stony Brook University

78

Spine Policy: Author Can Read Item

- The author of an EHR item can always read it, provided the patient has given one-off consent, even if the patient has sealed the item.
- `permits(cli, Read-spine-record-item(pat, id)) :-`
 - ◆ `hasActivated(cli, Spine-clinician(ra, org, spcty))`,
 - ◆ `hasActivated(x, One-off-consent(pat))`,
 - ◆ `Get-spine-record-org(pat, id) = org`,
 - ◆ `Get-spine-record-author(pat, id) = cli`

Spine Policy: Treating Clinician Can Read Item

- A treating clinician can read item if patient has given one-off consent, item is not sealed by patient, and the item's subjects are permitted for the clinician's specialty.
- `permits(cli, Read-spine-record-item(pat, id)) :-`
 - ◆ `hasActivated(cli, Spine-clinician(ra, org, spcty))`,
 - ◆ `hasActivated(x, One-off-consent(pat))`,
 - ◆ `canActivate(cli, Treating-clinician(pat, org, spcty))`,
 - ◆ `count-concealed-by-spine-patient(n, a, b)`,
 - ◆ $n = 0$, $a = (pat, id)$, $b = (org, cli, spcty)$,
 - ◆ `Get-spine-record-subjects(pat, id) ⊆ Permitted-subjects(spcty)`

Spine Policy: Activate Treating-clinician

- `cli` can activate `Treating-clinician(pat, org, spcty)` if
 - ◆ `pat` consented to treatment by `cli`, or
 - ◆ `pat` consented to treatment by workgroup containing `cli`, or
 - ◆ a clinician treating `pat` referred `pat` to `cli`, or
 - ◆ there is an emergency situation (audited later)

Patient Demographic System: Main Roles

- **PDS-manager**
 - ◆ Register and unregister people
- **Patient**
- **Agent**
- **Professional-user**
 - ◆ Clinicians, Caldicott guardians, etc.

Patient Demographic System: Activate Agent

- An agent can activate the `Agent(pat)` role if the agent is registered as a patient at the PDS, and the Spine confirms that he is an agent for `pat`.
- `canActivate(ag, Agent(pat)) :-`
 - ◆ `hasActivated(x, Register-patient(ag))`,
 - ◆ `no-main-role-active(ag)`,
 - ◆ `Spine@Spine.canActivate(ag, Agent(pat))`

Local Health Organization: Staff Roles

- **Clinician(spcty)**
 - ◆ as in Spine
- **Caldicott-guardian()**
 - ◆ patient advocate and ombudsman. can give consent on behalf of a patient and, in exceptional cases, override a patient's decisions.
- **HR-manager()**
 - ◆ Register and unregister patients and staff
- **Receptionist()**
 - ◆ Register patients

Local Health Organization: Non-Staff Roles

- Patient
- Agent
- Ext-treating-clinician
 - ◆ external clinician who needs access to patient's local EPR
- Third-party

Local Health Organization Policy: Activate Ext-treating-clinician

- An clinician can activate Ext-treating-clinician if patient has given consent and the clinician is certified by an RA certified by NHS.
- `canActivate(cli, Ext-treating-clinician(pat, ra, org, spcty)) :- hasActivated(ref, Consent-to-referral(pat, ra, org, cli spcty)), no-main-role-active(cli), ra@ra.hasActivated(y, NHS-clinician-cert(org, cli, spcty, start, end)), canActivate(ra, Registration-authority())`

Local Health Organization Policy: Deactivate Ext-treating-clinician

- Ext-treating-clinician is deactivated if patient (or patient's agent) withdraws patient's consent to the referral.
- `other-referral-consents(...)` holds if, e.g., an agent of the patient has given consent for the referreal.

`isDeactivated(cli, Ext-treating-clinician(pat, ra, org, spcty)) :- isDeactivated(x, Consent-to-referral(pat, ra, org, cli2, spcty)), other-referral-consents(0, x, pat, ra, org, cli, spcty)`

Other Potential Application Domains

- **Military**: cooperation with
 - ◆ other armed services (Army, Navy, Air Force, Marines)
 - ◆ government agencies
 - ◆ highly trusted coalition partners
 - ◆ less trusted coalition partners
- **Collaborative Engineering Design** [Bhargava+, 2004]
 - ◆ multi-vendor bids for large engineering contracts
 - ◆ collaborators need to share designs and design knowledge, status of technologies, etc.

Other Potential Application Domains

- **Supply Chain Management** [Bhargava+, 2004]
 - ◆ For tight integration, a company must give its suppliers, customers, and its customers' customers (to increase their confidence in its ability to deliver) some access to its order entry, order status, sales forecast, and production planning systems.

Outline

- Introduction and Motivation
- Design Issues and Features
- Trust Management Frameworks
- Sample Application Domains
- **Research Directions**
 - ◆ XACML for trust mgmt
 - ◆ State-dependent policies
 - ◆ Communication of rules
 - ◆ Administrative policy
 - ◆ Policy analysis
 - ◆ Trust for service provision

eXtensible Access Control Markup Language (XACML)

- Developed by OASIS, a consortium for information standards
- Supports attribute-based access control
- An XACML rule contains a
 - Target:** values for selected attributes of the subject, resource, and action
 - Effect:** permit or deny
 - Condition:** a boolean expression using the attributes
 - Example:** subject.age > 16
- A rule applies to a request if its target matches the request and its condition holds.

ACSAC 2005

Scott Stoller, Stony Brook University

91

XACML: Example

Rule: A patient may read his/her own medical record.

Target

Subject: any

Resource: MedRec.com/records

Action: read

Effect: permit

Condition:

```
target.subject.policyNumber =
getAttribute(resource,
"patientNumber")
```

Request:

Subject:

name: John Doe

patientNumber: 11231

Resource:

MedRec.com/records/
JohnDoe.xml

Action: read

This is my own syntax.

ACSAC 2005

Scott Stoller, Stony Brook University

92

XACML Policy

An XACML policy contains:

- Target**
- Set of **rules and policies**
- Combining algorithm**, to combine the effects from the rules and policies
 - Examples: permit overrides, deny overrides, first-applicable, only-one-applicable
- Obligation**, i.e., operations that the PEP should perform
 - Examples: write a log record, send a notification

ACSAC 2005

Scott Stoller, Stony Brook University

93

Evaluation of XACML Policy

- Evaluation of policy **pol** for request **req**:
 - If (**pol.target** matches **req**) then
 - evaluate the rules and policies in **pol** for **req**
 - combine their effects using the combining alg
 - return the resulting effect and the obligation
- XACML is not based on Datalog.
 - No inference of auxiliary facts.
- XACML policies are local: no credentials.

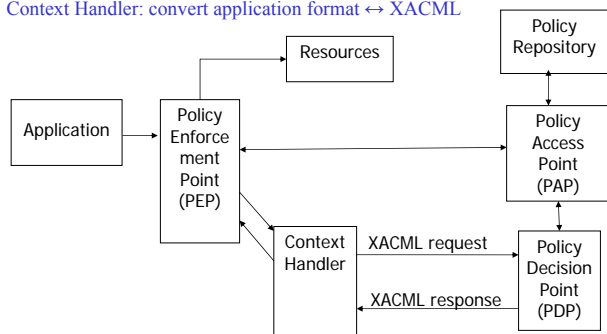
ACSAC 2005

Scott Stoller, Stony Brook University

94

XACML Architecture

Context Handler: convert application format ↔ XACML



ACSAC 2005

Scott Stoller, Stony Brook University

95

Using XACML for trust management

- Goals:**
 - Support Datalog-based trust management policies
 - Minimize changes to Policy Decision Point (PDP).
 - Change context handler (CH) instead.
- Each **atom loc@iss.r(args)** is represented as an XACML policy
 - add **Location** and **Issuer** elements to Policy
 - r(args)** is expressed as attributes of the subject, resource, and action. Details differ for different relations.

ACSAC 2005

Scott Stoller, Stony Brook University

96

XACML Representation of Facts and Rules

- Example: `loc@iss.permits(ent,act)` is represented as:
Location: `loc` **Effect:** `permit`
Issuer: `iss` **Obligation:** `none`
Target:
 Subject: `ent`
 Resource: `act`
 Action: `permits`
- Each Datalog rule `concl :- a1,a2,...` becomes an XACML policy that represents `concl` as above, and with obligations that are references to policies that represent `a1,a2,...`

Policy Evaluation

- **Goal-directed evaluation** is implemented in CH.
- CH sends request to PDP. PDP's reply contains obligations (subgoals).
- CH sends requests to evaluate each of them.
 - ◆ Request is sent to **local PDP** or **remote CH**, depending on the Location attribute.
 - ◆ We added **request broker** to XACML architecture to handle communication
- Limitations of current design and implementation
 - ◆ Does not fully support Datalog-style variables.
 - ◆ Does not put requester's name in requests to remote CH

State-Dependent Policies

- The state is normally **external data**.
- **Approach 1:** The application updates the state.
- **Example:** Teaching assistants (TAs) may change a student's grade for an assignment at most once.
`permit(TA, ChangeGrade(class, stu, assignment)) :-`
 TeachingAssistant(TA, class),
 numTAGradeChanges(class, stu, assignment) = 0.
- `numTAGradeChanges` is external data updated by the application. Grades are also external data.

State-Dependent Policies

- **Approach 2:** Specify updates as part of the policy.
- `allow(principal,operation(resource),effect)`: `principal` is authorized to perform `operation` on `resource` provided `effect` is executed at the same time.
- **Example:**
`permit(TA, changeGrade(class,stu,assignment),`
`increment(numTAGradeChanges(class,stu,assignment)) :-`
 TeachingAssistant(TA,class),
 numTAGradeChanges(course,student,assignment) = 0.
- DRM (digital rights mgmt) uses state-dependent policies.

Communication of Rules

- Policy evaluator may gather rules, as well as facts, from other sites. This can be more efficient.
- **Example:** SUNY students get discount at Textbooks.com.
- **Textbooks.com:**
`getDiscount(stu) :- SUNY@SUNY.student(stu).`
- **SUNY:** `student(stu) :- stu@SUNYSB.student(stu).`
`student(stu) :- stu@SUNYAlb.student(stu).`
- **JoeCool:** `SUNYSB.student(JoeCool).`
- Without rule communication: for each student, Textbooks.com asks SUNY which asks student.

Communication of Rules

- With rule communication: Textbooks.com imports rules from SUNY.
Textbooks.com: `SUNY.student(stu) :- SUNYSB.student(stu).`
`SUNY.student(stu) :- SUNYAlb.student(stu).`
- Only imported rules have conclusions with issuer \neq self.
- Textbooks.com asks each student for campus credential and applies an imported rule to infer SUNY credential.
- This reduces communication overhead and delays.
- Facts concluded using imported rules cannot be exported.
 - ◆ **Example:** `[SUNY.student(JoeCool)] signed by Textbooks.com` is an invalid credential.

Which Rules to Send?

- Textbooks.com asks SUNY for rules relevant to SUNY.student.
- **SUNY:** student(stu) :- SUNYSB.student(stu), approved(stu).
approved(stu) :- ...
- Which rules should SUNY send?
 - ◆ Rules with conclusion student(stu).
 - ◆ Rules with conclusion student(stu) and rules they depend on, recursively following dependencies.
- Some of the rules may have premises with remote issuers. Should all of them send relevant rules, too?

Which Rules to Send?

- **Binder** [DeTreville 2002] does not address this issue.
- **Secure Dynamically Distributed Datalog (SD3)** [Jim 2001] sends (in one step) all rules and facts that could be useful for answering the query. This minimizes communication delays but may send unnecessary rules and facts.
- **Open Issues:**
 - ◆ Privacy policies for rules
 - ◆ Policies that control which rules are sent, instead of a fixed algorithm.

Administrative Policy

- **Administrative policy:** security policy that controls changes to the security policy.
- Introduce **actions** that update the policy
 - ◆ addFact(fact), addRule(rule), removeFact(fact), removeRule(rule)
- Develop **authorization rules** for those actions.
- **Example:** allow(hrm, addFact(paymentMgr(e))) :- humanResourcesMgr(hrm), employee(e).
- **paymentMgr** may be internal (policy) data or external data.
- This extension to the API eliminates the need to encode such facts as role activations.

Administrative Policy: Static Separation of Duty

- **Separation of duty** limits the set of permissions of a single user. This helps prevent fraud, which requires collusion.
- **Example:** A single employee may perform at most 1 of the 3 steps involved in a purchase: issue purchase order, verify receipt of goods, issue payment.
- **Static separation of duty** allows an employee to be a member of at most 1 of the corresponding roles (purchasing clerk, receiving clerk, accounting clerk).
- **Example:** allow(so, addFact(member(e, PurchClerk))) :- COUNT(member(e, RcvClerk))=0, COUNT(member(e, AcctgClerk))=0.

Policy Analysis: Sample Analysis Questions

- Is a given principal allowed to perform a given action?
- Which principals are allowed to perform a given action?
- What is the effect of adding a given rule or fact?
 - ◆ i.e., what new actions can each principal perform?
- What is the effect of removing a given rule or fact?
 - ◆ i.e., what allowed actions does each principal lose?
- Is every principal that is allowed to perform a given action also allowed to perform another given action?
- Analysis algorithms for **SPKI/SDSI** [Jha+ 2004], **XACML** [Fisler+ 2005]

Policy Analysis with Administrative Policy

- Given:
 - ◆ a policy, including an administrative policy
 - ◆ a set of (less trusted) administrators
- Ask questions about the policies reachable from the current policy through changes those administrators can perform.
- Does **Q** hold for some such policy?
- Does **Q** hold for every such policy?
- **Q** is a yes/no question from the previous slide.

Policy Analysis with Administrative Policy: Example

- **Example:** Can an administrator for the CPU division give an employee access to resources in the RAM division?
- Such delegation of administrative control can be indirect.
 - ◆ **Example:** The RAM division allows employees on the company's re-engineering team to access RAM division resources. An administrator in the CPU division can appoint the CPU division's representative on the re-engineering team.
- Need to analyze possible delegation chains.
- This is computationally hard in general for rule-based languages [Li+ 2005, Sasturkar+ 2005].
- Currently looking for tractable cases of practical interest.

Trust for Service Provision

- **Access control:** Who do I trust to **access** this resource/service?
 - ◆ Usually a **boolean** answer is desired.
- **Service provision:** Who do I trust to **provide** this resource/service reliably?
 - ◆ Often desire a **quantitative evaluation** of providers
 - ◆ Final **decision** depends on trust level, cost, speed, etc.
 - ◆ Trust levels may be **discrete** (e.g., low/med/high) or **continuous** (e.g., a number between 0 and 1)
 - ◆ A **confidence level** for the trust evaluation can also be maintained.

Trust for Service Provision

- **Examples:** Trust in
 - ◆ Internet content ratings from different organizations
 - ◆ Internet storage providers (xdrive.com, idrive.com, netdrive.com, ...)
 - Availability, privacy, ...
- **Support for trust levels in rule-based policy languages**
 - ◆ Add trust level as a parameter of appropriate relations
 - ◆ **Example:** allowServiceProvider(service, provider, trustLevel)

Recap: Essential Features of Trust Management

- Each policy statement is associated with a principal, called its **source** or **issuer**.
- Each principal's policy specifies which **sources it trusts** for which kinds of statements, thereby **delegating** some authority to those sources.
- Policies may refer to domain-specific **attributes** of and **relationships** between principals, resources, and other objects.
- **Example:** AcmeHospital.allow(doc, Read(EPR(pat)) :-
AMA.doctor(doc),
pat.consentToTreatment(doc).

Thank You.
Any Questions?