

Security Policy Frameworks

Scott D. Stoller

- **Problem:** Security policies are often enforced by
 - ◆ **Humans** (slow, subject to social engineering, etc.)
 - ◆ **Application code** in Java, C++, etc. (hard to understand, analyze, and maintain)
- **Goal:** Express security policies in **high-level** languages.
 - ◆ **Easier** to read, analyze, and maintain
 - ◆ **Enforced** efficiently and **automatically** (except for occasional manual override)
- **My research on policies:** design of policy **languages**, algorithms for **analysis** and **enforcement** of policies

What Security Policy Frameworks?

- **Access Control Lists (ACLs):** Widely used, but well suited only for **small, closed, static** systems.
- **Role-Based Access Control (RBAC):** Associate permissions with roles. Add users as members of roles. More **scalable** than ACLs. Easier to **administer**.
 - ◆ **Advanced Features:** role hierarchy, separation of duty
 - ◆ **Standards:** SQL: 1999, ANSI/INCITS 359-2004
- **Attribute-Based Access Control (ABAC):** Use **all attributes** of subject and resource. Role is one attribute.
 - ◆ **Examples:** dept./company affil., licenses, location
 - ◆ **Standard:** XACML

What Security Policy Frameworks? (2)

- **Trust Management:**
 - ◆ **Delegate** authority for **attribute** info.
 - **Example:** Trust HR for affiliation, AMA for licenses
 - ◆ **Delegate** authority for **authorization** decisions.
 - **Example:** CEO allows project leader to give employees (not contractors) access to project budget.
- **Information Flow:** Control indirect access to information
- **Administrative Policy:** Controls changes to security policy. Needed for every framework.
 - ◆ **Example:** Dept. admin can modify policy related to his/her dept.

My Research on Security Policies

- **Overall Research:** Multi-threaded and distributed systems, program analysis and optimization, software testing and verification, security. 62 publications. \$1,322K grants as PI. \$5,586K grants as co-PI. 24 program committees.
- **Specification and efficient implementation** of RBAC
 - ◆ Used general method to implement **sets and relations**
- Generate **efficient implementations** of **trust management**
 - ◆ Used general method for implementation of **rules**
- **Analysis of Security-Enhanced Linux** policies
 - ◆ Fine-grained mandatory access control policies
 - ◆ Use **logic programs** to query and analyze policies

My Research on Security Policies (2)

- **Analysis** of security policies and configuration
 - ◆ Use **deductive spreadsheets** [C.R. Ramakrishnan]
- **Analysis** of **attribute-based access control** policies
 - ◆ Logic-based verification tools: SAT solvers, BDDs, ...
 - ◆ Joint work with U.S. Naval Research Lab.
- **Analysis** of role-based **administrative policies** for RBAC
 - ◆ Based on **planning** algorithms in Artificial Intelligence
 - ◆ Analyze effect of **sequences of changes** by admins
- **Analysis** of **information flow** in programs
 - ◆ Algorithms to infer all **possible information flows**.

eXtensible Access Control Markup Language

- **Attribute-based access control**, delegation, policy admin.
- Developed by **OASIS**. XACML TC: BEA, BMC Software, BAH, Computer Associates, Entrust, IBM, Oracle, Sun
- Open-source implementation of key components by Sun.
- Defines XML formats for **requests, responses** (permit, deny, ...), and access control **rules** and **policies**.
- **Rule:**
 - ◆ **Target:** values of attributes of **subject, resource, action**
 - ◆ **Effect:** **permit** or **deny**
 - ◆ **Condition:** a **boolean expression** using the attributes
 - **Example:** `subject.age > 16`

XACML: Example Rule and Request

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

Target
Subject: any
Resource:
 namespace: med.com/record.xsd
Action: actionID: read
Effect: permit
Condition:
 target.subject.patientNumber = resource.patient.patientNumber

Request:
Subject:
 name: John Doe
 patientNumber: 1123
Resource:
 med.com/records/Bart Simpson.xml
Action: actionID: read

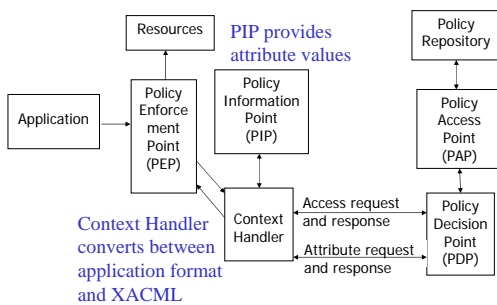
This is shorthand syntax.

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

XACML Architecture



Proposed Project, Part 1: Design and Implementation of PIP

- **Policy Information Point (PIP)** provides attribute values, by retrieving them from DBMSs, LDAP repositories, ...
- **Benefits of PIP driven by PDP:** avoid spurious denials due to omitted attributes, avoid changes to application code when policy changes.
- No existing open-source implementation of PIP.
- We propose to
 - ◆ Define format for **configuration files** that specify where to get info for each attribute
 - ◆ Develop a PIP that uses such configuration files to retrieve attributes from DBMSs and (if time permits) LDAP repositories.

Proposed Project, Part 2: Concrete Policy Analysis for XACML

- **Concrete policy analysis** requires access to attribute info.
- Support this simple but powerful policy analysis:
 - ◆ **Query:** a request template, i.e., a partially specified request
 - ◆ **Result:** all permitted instances
- Develop **policy analysis engine** by extending the PIP.
- Generate **queries** to retrieve attribute info from DBMSs, based on PIP config file, then **compute the analysis result** from the query result, based on the policy.
- Support LDAP repositories, if time permits. Would need to extend LDAP server with **reverse lookup**.

Follow-On Projects on Policy Analysis for XACML

- **Other policy analysis questions:** consistency of permit and deny rules, detection of dead rules, change impact analysis, comparison of strictness of policies, ...
- Analyze **delegation** and **policy administration**
 - ◆ Analyze chains of delegation
 - ◆ Analyze sequences of policy changes
- Concrete analysis contrasts with **symbolic policy analysis**.
 - ◆ **Result:** formula characterizing permitted instances
 - ◆ **Example:** Grading: subject=resource.course.instructor