Mining Attribute-Based Access Control Policies from RBAC Policies

Zhongyuan Xu  Scott D. Stoller

Department of Computer Science

Stony Brook University
Motivation

- Role-based access control (RBAC) is very widely used but has notable limitations, prompting a shift towards more advanced access control models.

- Attribute-based access control (ABAC) allows policies to be written in a more flexible and higher-level way, reducing the effort and cost of policy management.

- Migrating from RBAC to ABAC can be labor-intensive and expensive.

- Policy mining algorithms partially automate the construction of ABAC policies from RBAC policies and possibly other information, such as user and resource attributes.
Mining ABAC Policies from RBAC Policies

- Consistency
- Preserve RBAC Structure
- Maximize Policy Quality

RBAC Policy → Policy Mining Algorithm → ABAC Policy
Main Contributions

- **First formal definition** of the problem of mining ABAC policies from RBAC policies and attribute data.
- **First algorithm** specifically designed to mine an ABAC policy from an RBAC policy and attribute data.
- **Evaluation** of the algorithm on case studies.
RBAC Policy

- An RBAC policy consists of a user set, permission set, and role set
- user-role assignment
- role-permission assignment
- role inheritance relation

<table>
<thead>
<tr>
<th>User</th>
<th>member of</th>
<th>Role</th>
<th>has</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scott</td>
<td></td>
<td>Faculty</td>
<td></td>
<td>Read Class Schedule</td>
</tr>
<tr>
<td>Sekar</td>
<td></td>
<td>Dept Chair</td>
<td></td>
<td>Write Budget</td>
</tr>
<tr>
<td>Ari</td>
<td></td>
<td>Chair</td>
<td></td>
<td>Update Server Configuration</td>
</tr>
<tr>
<td>Brian</td>
<td></td>
<td>Web Master</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ABAC Policy

- An ABAC policy consists of a
  - user set, resource set, and operation set
  - user attribute data and resource attribute data
  - rule set

![Diagram showing ABAC policy with users, rules, and resources connected by constraints.]

- User id=Scott, teach={...}
- User id=Ari, teach={...}
- Resource id=CSE101, dept=CS, type=roster
- Resource id=CSE506, dept=CS, type=roster

related by constraint
Attribute Expression

- **Attribute expressions** specify the sets of users and resources associated with rules.

- **Example of a User Attribute Expression:**
  
  \[
  \text{department} = \text{CS} \land \\
  \text{position} \in \{\text{grad, ugrad}\} \land \\
  \text{classes} \supseteq \{\text{CS101, CS102}\}
  \]

- This expression is satisfied by
  - users in the CS department,
  - who are graduate students or undergraduate students, and
  - whose classes include CS101 and CS102 (and possibly other classes).
Constraint

- **Constraint**: expresses an equality, set membership, or subset relationship between a user attribute and a resource attribute.

- **Format**: `user attribute ∼ resource attribute`, where ∼ is a relation.

- **Examples**:
  1. department = department
     user’s department equals resource’s department
  2. courseTaught ⊇ course
     value of user’s courseTaught attribute contains value of resource’s course attribute
  3. specialties ⊇ topics
     value of user’s specialties attribute is a superset of value of resource’s topics attribute
ABAC Rule

- A rule consists of a
  - user-attribute expression (UAE)
  - resource-attribute expression (RAE)
  - operation set
  - constraint

- Example Rule:
  - UAE: position = faculty
  - RAE: type = gradebook
  - Operations: \{changeScore, assignGrade\}
  - Constraint: courseTaught \ni course

- This rule specifies that the instructor for a course can change scores and assign grades in the gradebook for that course.
A policy quality metric $Q_{\text{pol}}$ is a function from ABAC policies to a totally-ordered set. We use a measure of policy size.

Given an RBAC policy $P_{\text{RBAC}}$, attribute data, and a policy quality metric $Q$, find an ABAC policy $P$ such that

- $P$ is consistent with $P_{\text{RBAC}}$
  - $P$ and $P_{\text{RBAC}}$ grant each user the same permissions.
- $P$ preserves the structure of $P_{\text{RBAC}}$
  - Each rule in $P$ grants the same permissions as some set of roles in $P_{\text{RBAC}}$ after splitting (see next slide), with no redundancy (i.e., each role is covered by exactly one rule).
- $P$ has the best quality according to $Q$ among such policies.
ABAC Policy Mining Algorithm

1. **Split** each role in $P_{RBAC}$ into a set of roles (if necessary), so the set of permissions granted by each role is the Cartesian product of a set of resources and a set of operations.

2. For each resulting role $r$, **add** to the ABAC policy a rule consisting of
   - **user-attribute expression** satisfied only by $r$’s members
   - **resource-attribute expression** satisfied only by the resources in $r$’s permissions
   - **operation set** containing operations that appear in $r$’s permissions
   - **strongest constraint** that holds between every member of $r$ and every resource in a permission of $r$
3. **Improve the policy quality** by repeatedly applying the following transformations, which preserve consistency and preserve the structure of $P_{RBAC}$:
   - remove redundant rules
   - merge pairs of rules, when the merged rule is valid
   - simplify rules by removing conjuncts, values in conjuncts, constraints, etc.
Evaluation

▶ We wrote 3 case study policies (university records, health care, project management) in RBAC and ABAC, applied our algorithm to the RBAC policy and accompanying attribute data, and compared the generated ABAC policy with the manually written one.

▶ When all relevant attribute data is available, our algorithm produces an intuitive high-level ABAC policy from each RBAC policy, with only minor differences from the hand-written ABAC policy.

▶ When some relevant attribute information is unavailable, our algorithm produces an intuitive high-level ABAC policy that uses the available attribute data and uses role membership as a substitute for missing attribute data.
General View of Access Control Policy Mining

- Correctness Requirements
- Optimization Goal

Source Access Control Policy → Policy Mining Algorithm → Target Access Control Policy

Other Data
Mining ABAC Policies from ACL Policies

ACL Policy → Policy Mining Algorithm → ABAC Policy

Attribute Data

Consistency

Maximize Policy Quality
Mining Attribute-Based Access Control Policies from RBAC Policies

Our Other Work on Policy Mining

Mining ABAC Policies from ACL Policies

This problem is harder, because there are no roles to guide rule generation. We developed a 3-phase algorithm and applied it successfully to case studies and synthetic policies.

1. Use selected user-permission relations in the ACL policy as seeds for rules. Generalize each of these rules to cover more of the ACL policy by replacing conjuncts in attribute expressions with constraints.

2. Improve policy quality by merging and simplifying rules.

3. Select the highest-quality rules to form the final policy.
Mining Attribute-Based Access Control Policies from RBAC Policies

Our Other Work on Policy Mining

Minimize policy size, over-assignments, and under-assignments

Operation Logs → Policy Mining Algorithm → ABAC Policy

Attribute Data
Mining ABAC Policies from Logs

- This problem is **even harder**, because operation logs provide **incomplete** information about the current policy.
- When **generalizing** rules to cover more user-permission relations seen in the log, allow high-quality rules that grant permissions not seen in the log.
- These additional permissions are called **over-assignments**.
- A parameter controls the **trade-off** between **policy quality** and **over-assignments**.
- **Usage frequency** can help guide policy mining.