A Role Based Administration Model
For Attribute

Xin Jin, Ram Krishnan, Ravi Sandhu

SRAS, Sep 19, 2012
Motivation

Background

Proposed Approach

Discussion and Future Work

World-Leading Research with Real-World Impact!
Attribute Based Access Control provide flexibility and scalabilities for distributed access control

- Role based trust management (RT)[Oakland02]
- Unified ABAC model (ABAC-alpha) [DBSEC12]
- Attribute based Encryption (ABE) [CCS06]
- Usage control model (UCON) [TISSEC]
- Logical based framework for ABAC [FMSE04], etc.

What is user attribute(credential, certificate , etc.)?
- Name, address, country…
- ID, clearance…
Much effort in how to use and store user attributes.
- Authorization policy evaluates attribute of subject and object in the request \((s, o)\) (ABAC-ALPHA, UCON, Logical based framework for ABAC)
- Attribute Encryption (ABE, PKI)
- Industry Implementation Standard (XACML, SAML, etc.)

Not enough on user attribute management
- Who is authorized to assign user attributes?
  - Who is authorized to assign value for Salary(Alice)?
  - Can Bob say that Salary(Alice)=3000?
- How should permissions be assigned to administrators?
  - Manually one by one? No!
Motivation

- Effective user attribute administration model needed
  - Ease of Administration one of desired features

- Role based Access Control
  - Good candidate for user attribute Administration
    - Proved ease of administration
    - Efficient safety analysis
    - Sizable literature, etc.

Example:
- E.g. ProgramManager
- E.g. Assign Alice’s salary to 3000, 6000 9000
User role assignment model (URA97)

Can_assign( administration role, prerequisite role, role range notation )

Example: Pr programmanager Employee ∨ projectmanager {prj1leader, proj2leader}

Can_revoke( administration role, role range notation )

Example: Pr programmanager {prj1leader, proj2leader}

User attributes

- Atomic: User can only have one value for this kind of attribute. E.g. user id, clearance, etc.
- Set: User can be assigned multiple values. E.g. role, phonenumbers, etc.
- Range(ua), attType(ua)
Proposed Approach

- Generalize Role as one of user attributes
  - Difference between role and user attribute
    - Role represents permissions while attributes do not
      (Alice’s role is Program Chair VS Alice’s age is 23)
    - Role has hierarchy while attribute may not
      (Role: CEO is senior role of employee. Attribute: Address)

- Difference between atomic user attributes and set-valued user attributes
  - Assign new value to atomic user attribute automatically remove old value
  - Add new values to set-valued attribute does not guarantee permissions to delete existing attribute values.
Proposed Approach

- Permissions to each administrative role
  - Add value to set-valued user attributes
  - Delete value from set-valued attributes
  - Assign value to atomic-valued attributes

- Specify the three permission sets for each role
  - Add value to set-valued user attributes
  - Delete value from set-valued attributes
  - Assign value to atomic-valued attributes
Proposed Approach (GURA₀)

- Add values to set-valued user attributes:

\[ \forall sua \in SUA. \text{can\_add}_{sua} \subseteq AR \times \text{EXPR}(sua) \times 2^{\text{Range}(sua)} \]

**AR**: set of administrative roles  
**SUA**: set of set-valued user attributes  
**EXPR(sua)**: logical expression composed of user attribute *sua*
Proposed Approach (GURA₀)

Delete values from set-valued user attributes:

\[ \forall sua \in SUA. \ can\_delete\_sua \subseteq AR \times EXPR(sua) \times 2^{Range(sua)} \]

**AR:** set of administrative roles  
**SUA:** set of set-valued user attribute  
**EXPR(sua):** logical expression composed of user attribute \( sua \)
Proposed Approach (GURA₀)

- Assign value to atomic-valued user attributes:

\[
\forall sua \in AUA. \, \text{can\_assign}_{a ua} \subseteq AR \times \text{EXPR}(a ua) \times 2^{\text{Range}(a ua)}
\]

**AR**: set of administrative roles  
**AUA**: set of atomic-valued user attributes  
**EXPR(aua)**: logical expression composed of user attribute \( aua \)
Common Expression Language

\[ \varphi ::= \varphi \land \varphi \mid \varphi \lor \varphi \mid (\varphi) \mid \neg \varphi \mid \exists x \in \text{set.}\varphi \mid \forall x \in \text{set.}\varphi \mid \text{set compare set} \mid \text{atomic} \in \text{set} \mid \text{atomic} \notin \text{set} \mid \text{atomic atomic compare atomic} \]

atomic compare ::= < | = | \leq | \neq
set compare ::= \subset | \subseteq | \not\subseteq

Example: set-valued attributes in GURA_0

EXPR(SUA) is specified using an instance of the above language where

\[ \text{set ::= sua(u) | constantSet} \]
\[ \text{atomic ::= constantAtomic} \]
Example

Users And User Attributes

- InvolvedProject
- Employee
- Salary

Administrative Roles

- PrjManager
- PrjLeader1
- PrjLeader2
- HumanResourceManager

Operations:
- Add Employee to prj1 and prj2 but not both
- Add Employee who is not involved in prj2 to prj1
- Add Employee who is not involved in prj1 to prj2
- Promote the salary of employee whose salary is less than 2000 to 3000, 6000 or 9000
**Example**

*can_add(Pr jManager, prj1 \notin Involved Project(u), involved Project, prj2) can_add(Pr jManager, prj2 \notin Involved Project(u), Involved Project, prj1)*

**Users And User Attribute**

- InvolvedProject
- Employee
- Salary

**Administrative Roles**

- PrjManager
- PrjLeader1
- PrjLeader2
- HumanResourceManager

*Add Employee who is not involved in prj2 to prj1*

*Add Employee who is not involved in prj1 to prj2*

*Promote the salary of employee whose salary is less than 2000 to 3000, 6000 or 9000*
**Example**

**Users And User Attribute**

- **InvolvedProject**
- **Employee**
- **Salary**

**Administrative Roles**

- **PrjManager**
- **PrjLeader1**
- **PrjLeader2**
- **HumanResourceManager**

**can_add** (Pr jManager, prj1 \(\notin\) Involved Project(u), involved Project, prj2)  
**can_add** (Pr jManager, prj2 \(\notin\) Involved Project(u), Involved Project, prj1)

Add Employee who is not involved in prj1 to prj2:

Promote the salary of employee whose salary is less than 2000 to 3000, 6000 or 9000:
**Example**

**Users And User Attribute**

- **InvolvedProject**
  - **Employee**
  - **Salary**

**Administrative Roles**

- **PrjManager**
- **PrjLeader1**
- **PrjLeader2**
- **HumanResourceManager**

- `can_add(Pr jManager, prj1 \notin Involved Project(u), involved Project, prj2)
  can_add(Pr jManager, prj2 \notin Involved Project(u), Involved Project, prj1)`
- `can_add(Pr jLeader1, prj2 \notin Involved Project(u), Involved Project, prj1)`
- `can_add(Pr jLeader2, prj1 \notin Involved Project(u), Involved Project, prj2)`

Promote the salary of employee whose salary is less than 2000 to 3000, 6000 or 9000.
Example

**Users And User Attribute**

- **InvolvedProject**
  - **Employee**
  - **Salary**

**Administrative Roles**

- **PrjManager**
  - **PrjLeader1**
  - **PrjLeader2**
  - **HumanResourceManager**

**Constraints**

- $can\_add(Pr\ jManager, prj1 \notin Involved\ Project(u), Involved\ Project, prj2)$
- $can\_add(Pr\ jManager, prj2 \notin Involved\ Project(u), Involved\ Project, prj1)$
- $can\_add(Pr\ jLeader1, prj2 \notin Involved\ Project(u), Involved\ Project, prj1)$
- $can\_add(Pr\ jLeader2, prj1 \notin Involved\ Project(u), Involved\ Project, prj2)$

$can\_assign(HR, salary(u) < 2000, salary, \{3000, 6000, 9000\})$
Proposed Approach (GURA1)

- Limited Expressive Power
  - Expression (ua) ONLY composed using the user attribute in context is not enough.
    Example: PrjLeader1 is only authorized to add employee whose experience with Java is more than 3 years and whose skills include C++.

- Least Privilege is not achieved
  - Expression (ua) CAN NOT restrict users to the least set. Need other user attributes to compose the expression.
Proposed Approach (GURA₁)

➢ Further Generalization

∀ sua ∈ SUA. can_add_{sua} ⊆ AR × EXPR(sua) × 2^{Range(sua)}

EXPR(UA): logical expression composed of all user attribute ua in UA
Proposed Approach (GURA₁)

➢ Further Generalization for GURA₁:

\[ \forall \text{ sua} \in \text{SUA. can\_delete}_{\text{sua}} \subseteq AR \times \text{EXPR} (UA) \times 2^{\text{Range}(\text{sua})} \]

\[ \forall \text{ aua} \in \text{AUA. can\_assign}_{\text{aua}} \subseteq AR \times \text{EXPR} (UA) \times 2^{\text{Range}(\text{aua})} \]
Example using GURA

Table 5: Examples

can_add relation for attribute involvedprj:
(prj1leader, prj2  \notin involvedprj(u) \land trainingpassed(u) =
true \land clearance(u) > S \land C \in skills(u), \{prj1\})

(prj2leader, prj1  \notin involvedprj(u) \land trainingpassed(u) =
true \land clearance(u) > S \land C \in skills(u), \{prj2\})


can_add relation for attribute skill:
(secretary, NULL, \{C, C++, Java\})

can_delete relation for attribute involvedprj:
(prj1leader, prj1  \notin involvedprj(u), \{prj1\})
(prj2leader, prj2  \notin involvedprj(u), \{prj2\})

can_delete relation for attribute skill:
(secretary, NULL, \{C, C++, Java\})

can_assign relation for attribute trainingpassed:
(trainingmanager, NULL, \{true, false\})

can_assign relation for attribute clearance:
(humanmanager, NULL, \{TS, S, C, U\})
Discussions and Future Work

- Advantages
  - Advantages of RBAC inherited

- Limitations
  - Awkward for distributed administration of user attributes
  - For fine-grained user attribute administration, many roles with slight different set of permissions need to defined.

- Future Work
  - Delegation
  - Attribute based user attribute management
Any Questions?