Object-to-Object Relationship Based Access Control: Model and Multi-Cloud Demonstration

by

Tahmina Ahmed, Farhan Patwa and Ravi Sandhu
Department of Computer Science
University of Texas at San Antonio
7/29/2016
Outline

• Introduction
• Background and Motivation
  • Relationship in OSN
  • ReBAC beyond OSN
  • Existence of Object Relationship independent of user
  • Limitations of Existing ReBAC Models
• Model Characteristics
• OOReBAC Model
• OOReBAC: Application
• Implementation

World-Leading Research with Real-World Impact!
User to user relationships in a sample social graph [UURAC, Cheng et al. 2012]

User to user, user to resource and resource to resource relationships in a sample social graph [URRAC, Cheng et al. 2012]
A sample Relationship Graph for Organizational Environment [RPPM, Crampton et al., 2014]
Most of the ReBAC for OSN considers only user to user relationship

OSN has very specific types of resources – photos, notes, comments. Which only makes sense along with users.

Even though some ReBAC models consider general computing system they still need users/subjects existence in relationship graph.
Object Relationship in Object-Oriented System (Inheritance, Composition and Association)

History of a Git Project (Version Control System) is a DAG
Limitations of Existing ReBAC Models

- Cannot configure relationship between objects independent of user.
- Cannot express authorization policy solely considering object relationship.

Considering these limitations we are proposing an object-to-object relationship based access control model.
An Object to Object Relationship Based Access Control

ACL(o₁) = {u₁}
ACL(o₂) = {}
ACL(o₃) = {u₂}

Relationship
ACL
Access Request

Policy Level Example

policyLevel(a₁,o₁) = 2
policyLevel(a₂,o₁) = 0
policyLevel(a₁,o₂) = 1
policyLevel(a₂,o₂) = 0
policyLevel(a₁,o₃) = 3
policyLevel(a₂,o₃) = 2
policyLevel(a₁,o₄) = 2
policyLevel(a₂,o₄) = 0
OOReBAC: Model Components and Definition

- U is a set of users
- O is a set of objects
- R \subseteq \{ z \mid z \subseteq O \wedge |z| = 2 \}
- G = \langle O, R \rangle is an undirected relationship graph with vertices O and edges R
- A is a set of actions
- P^p(o_1) = \{ o_2 \mid there exists a simple path of length p in graph G from o_1 to o_2 \}
- policyLevel: O \times A \rightarrow N
- ACL: O \rightarrow 2^U which returns the Access control List of a particular object.
- There is a single policy configuration point. Authorization Policy. for each action a \in A, Authz_a(u:U,o:O) is a boolean function which returns true or false and u and o are formal parameters.
- Authorization Policy Language:
  Each action “a” has a single authorization policy
  Authz_a(u:U,o:O) specified using the following language.
  \[
  \phi := u \in PATH_\delta \\
  PATH_\delta := ACL(P^0(o)) \cup \ldots \cup ACL(P^i(o)) \quad where \quad i = \min(|O| - 1, policyLevel(a,o))
  \]
  where for any set X, ACL(X) = \bigcup_{x \in X} ACL(x)

Constraints
Configuration:

- $A = \{\text{read, write}\}$
- $\text{Authz}_{\text{read}}(u; U, o; O) \equiv u \in P_{\text{policyLevel}(\text{read}, o)}$
- $\text{Authz}_{\text{write}}(u; U, o; O) \equiv u \in P_{\text{policyLevel}(\text{write}, o)}$

Sequence of operations and its outcome:

- $U = \{u_1, u_2, u_3\}$
- $O = \{o_1, o_2, o_3, o_4\}$
- $R = \{\{o_1, o_2\}, \{o_2, o_3\}, \{o_3, o_4\}\}$
- $\text{ACL}(o_1) = \{u_1\}$
  - $\text{ACL}(o_2) = \{u_3\}$
  - $\text{ACL}(o_3) = \{u_2\}$
  - $\text{ACL}(o_4) = \{u_3\}$
- $\text{policyLevel}(\text{read}, o_1) = 2$
- $\text{policyLevel}(\text{write}, o_1) = 0$
- $\text{policyLevel}(\text{read}, o_2) = 2$
- $\text{policyLevel}(\text{write}, o_2) = 1$
- $\text{policyLevel}(\text{read}, o_3) = 0$
- $\text{policyLevel}(\text{write}, o_3) = 0$
- $\text{policyLevel}(\text{read}, o_4) = 2$
- $\text{policyLevel}(\text{write}, o_4) = 1$

- $\text{read}(u_1, o_3)$, $\text{write}(u_1, o_3)$ are denied
- $\text{read}(u_2, o_1)$ is allowed, $\text{write}(u_2, o_1)$ is denied
- $\text{read}(u_1, o_4)$, $\text{write}(u_1, o_4)$ are denied
An OOReBAC Instantiation

- \( U = \{ u_{pp}, u_{gs}, u_{od}, u_{op}, u_{pp}, u_{rp} \} \)
- \( O = \{ mr_{pp}, mr_{gs}, mr_{od}, mr_{op}, mr_{od}, mr_{rp} \} \)
- \( R = \{ mr_{pp}, mr_{gs}, mr_{od}, mr_{op}, mr_{od}, mr_{rp} \} \)
- \( ACL(mr_{pp}) = \{ u_{pp} \} \)
- \( ACL(mr_{gs}) = \{ u_{gs} \} \)
- \( ACL(mr_{od}) = \{ u_{od} \} \)
- \( ACL(mr_{op}) = \{ u_{op} \} \)
- \( ACL(mr_{od}) = \{ u_{od} \} \)
- \( ACL(mr_{rp}) = \{ u_{rp} \} \)
- \( Action = \{ read, write \} \)
- \( policyLevel(read, mr_{pp}) = \infty, policyLevel(write, mr_{pp}) = 0, \)
  \( policyLevel(read, mr_{gs}) = \infty, policyLevel(write, mr_{gs}) = 0, \)
  \( policyLevel(read, mr_{od}) = \infty, policyLevel(write, mr_{od}) = 0, \)
  \( policyLevel(read, mr_{op}) = \infty, policyLevel(write, mr_{op}) = 0, \)
  \( policyLevel(read, mr_{od}) = \infty, policyLevel(write, mr_{od}) = 0, \)
  \( policyLevel(read, mr_{rp}) = \infty, policyLevel(write, mr_{rp}) = 0 \)
- \( Authorization policy: \)
  \( Auth_{read}(u, o) \equiv u \in \mathcal{P}_{\text{policyLevel}(read, o)} \)
  \( Auth_{write}(u, o) \equiv u \in \mathcal{P}_{\text{policyLevel}(write, o)} \)

Sequence of Operations and Outcomes

1) \( read(u_{rp}, mr_{pp}) \) : authorized
2) \( read(u_{od}, mr_{op}) \) : authorized
3) \( write(u_{rp}, mr_{op}) \) : authorized
4) \( write(u_{rp}, mr_{pp}) \) : denied
5) \( write(u_{rp}, mr_{pp}) \) : denied

© Tahmina Ahmed

World-Leading Research with Real-World Impact!
Implementation: Openstack Object Storage (Swift)

Relationship

<table>
<thead>
<tr>
<th>SourceFileName</th>
<th>TargetFileList</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁@cloud₁:account₁:container₁</td>
<td>{f₂@cloud₄:account₄:container₄, f₃@cloud₂:account₂:container₂}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ACL

<table>
<thead>
<tr>
<th>Filename</th>
<th>UserList</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁@cloud₂:account₂:container₂</td>
<td>{u₄@cloud₄:account₄, u₅@cloud₄:account₄}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

PolicyLevel

<table>
<thead>
<tr>
<th>FileName</th>
<th>Policy Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₂@cloud₄:account₄:container₄</td>
<td>(download,2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Functional Specification:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Conditions</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Administrative Actions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateRelationship</td>
<td>admin ∈ role(u) ∧ cloud(filename1) = cloud(u) ∧ filename1 ∉ RelationshipSet(filename2) ∧ filename2 ∉ RelationshipSet(filename1)</td>
<td>RelationshipSet[filename1] := {filename2}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeleteRelationship</td>
<td>admin ∈ role(u) ∧ cloud(filename1) = cloud(u) ∧ filename1 ∈ RelationshipSet(filename2) ∧ filename2 ∈ RelationshipSet(filename1)</td>
<td>RelationshipSet[filename1] := {filename2}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IncludeUserInACL</td>
<td>Role(u) ∈ Admin ∧ cloud(filename1) = cloud(u) ∧ username1 ∉ ACLSet[filename1]</td>
<td>ACLSet[filename1] := {username1}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExcludeUserFromACL</td>
<td>Role(u) ∈ Admin ∧ cloud(filename1) = cloud(u) ∧ username1 ∈ ACLSet[filename1]</td>
<td>ACLSet[filename1] := {username1}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConfigurePolicyLevel</td>
<td>Role(u) ∈ Admin ∧ cloud(filename1) = cloud(u) ∧ num ≤</td>
<td>O</td>
</tr>
</tbody>
</table>

| Operational Command      |            |                                   |
| download                 | u ∈ U ∧ authorize(u, filename1, O) | allow user u to download file filename1 |
Conclusion and Future Work

• OOReBAC is the first attempt towards using object relationship independent of user in authorization policy specification and can only do where single type symmetric relationship is used.

• Limitations of OOReBAC:
  – Version Control system uses asymmetric relationship.
  – Object oriented Programming needs multiple Type asymmetric relationships.

We need to extend this model to accommodate multiple type asymmetric relationships to configure version control and object oriented system.
Questions?