Role and Attribute Based Collaborative Administration of Intra-Tenant Cloud IaaS

(Invited Paper)

Xin Jin, Ram Krishnan and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio

October 22–25, 2014
10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing

World-Leading Research with Real-World Impact!
IT Infrastructure Operations

World-Leading Research with Real-World Impact!
Access Control

World-Leading Research with Real-World Impact!
Cloud Service Models

- **Software as a Service (SaaS)**
- **Platform as a Service (PaaS)**
- **Infrastructure as a Service (IaaS)**

**Network accessible software**

**App dev environment with cloud characteristics**

**Virtualized hardware infrastructure**
Equivalent policies should be configurable using cloud access control service.

With virtualization, cloud may provide more fine-grained access control.
Access Control in IaaS

**Cloud Root User Tasks:**
1. Manage virtual infrastructure
2. Create and manage tenants (e.g., create tenant root-user)

**Tenant Root User Tasks:**
1. Configure attributes of tenant’s users and cloud resources
2. Create and manage admin users
3. Manage attributes of admin users

**Tenant Administrative User Tasks:**
1. Create and manage tenant’s regular users
2. Manage attributes of regular users

**Tenant Regular User Tasks:**
1. Day-to-Day Operations
2. Add/Remove Capacity
3. Manage N/W
4. Backup, Snapshot, etc.
5. Manage attributes of tenant’s resources

IaaS Administrative Model

IaaS Operational Model

World-Leading Research with Real-World Impact!
Requirements: Intra-CSP

Tenant A
Access Control Requirement #1
config

Tenant B
Access Control Requirement #2
config

......

Tenant N
Access Control Requirement #n
config

CSP #1's Access Control Interface

IaaS CSP #1
Requirements: Inter-CSP

Communicate

Tenant A

Access Control Requirement

config

CSP #1's Access Control Interface

IaaS CSP #1

CSP #N's Access Control Interface

IaaS CSP #N

World-Leading Research with Real-World Impact!
Key Requirements

➢ Requirements
  ➢ Tenants’ full control over their access control design
  ➢ Simple yet flexible administrative policy
  ➢ Flexible operational model
  ➢ Strong formal foundations

➢ Existing Models
  ➢ Industry Models
    ➢ OpenStack and Amazon Web Service
  ➢ RBAC-based Models
    ➢ Using the legend RBAC model
  ➢ ABAC-based Models
    ➢ More details to follow
Limitations

- Tenant can not configure their own policy, uses cloud role instead
- Not able to configure tenant administrator
- Access control on operation level, no control on object level
  - Give `identity:createUser` permission to role r1, then r1 can create users in any tenant
  - Give `nova:stop` permission to role r1, r1 can stop any machine in the tenant
- Access control only based on role
AWS Access Control

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:AddUserToGroup",
                "iam:RemoveUserFromGroup",
                "iam:GetGroup"
            ],
            "Resource": "arn:aws:iam::123456789012:group/MarketingGroup"
        }
    ]
}
```
AWS Access Control

- Advantages over OpenStack
  - Tenant has full control over their own policy, by account root user
  - Flexible policy: groups, user id, time, address.
  - Control over resources and operations

- Limitations
  - No automation
  - Restricted set of attributes
  - Not flexible enough, group explosion
  - No extension available (e.g., can not include customized attributes)
  - No subject and user distinction
Related ABAC models

- **Formal Model**
  - UCON\textsubscript{ABC} (Park and Sandhu, 01): authorization, mutable attributes, continuous enforcement
  - Logical framework (Wang et al, 04): set-theory to model attributes
  - NIST ABAC draft (Hu et al, 13): enterprise enforcement

  No difference between user and subject (classical models can not be configured)
  No relationship of user, subject and object attributes.

- **Policy Specification Language**
  - SecPAL (Becker et al 03, 04), DYNPAL (Becker et al 09), Rule-based policy (Antoniou et al, 07), Binder (DeTreville 02), EPAL1.2 (IBM, 03), FAF (Jajodia et al 01)

- **Enforcement Models**
  - ABAC for web service (Yuan et al 06), PolicyMaker (Blaze et al 96)

- **Implementations**
  - XACML: authorization
  - SAML: pass attributes
  - OAuth: authorization

- **Attribute Based Encryption**
  - KP-ABE (Goyal et al 06), CP-ABE (Bethencourt et al 07)

**World-Leading Research with Real-World Impact!**
Proposed Model

- ABAC-alpha model \[1\] and GURA model \[2\]
- Flexibility
  - Covers DAC, MAC and RBAC
  - Potentials to covers various RBAC extensions
  - Resource-level fine-grained access control
- Automation
  - User attributes inherited by subject and further object, access control automatically added for newly created objects
- Ease in policy specification and administration
  - Attributes defined to reflect semantic meaning and policy specified with certain level of relationship to natural language

Different types of object may have different sets of attributes.
TReU, S and O represent finite sets of existing regular users, subjects and objects respectively.

UA, SA and OA represent finite sets of user, subject and object attribute functions respectively.

\[ \text{objType: } O \rightarrow OT. \text{ For each object, objType gives its type.} \]

\[ \forall t \in OT, O_t = \{ \text{obj} | \text{obj} \in O \land t = \text{objType(obj)} \}, \text{represents objects of type } t. \]

\[ \text{oaType: } OA \rightarrow 2^{OT}. \text{ For each object attribute, oaType gives its types.} \]

\[ \forall t \in OT, OA_t = \{ \text{oa} | \text{oa} \in OA \land t \in \text{oaType(oa)} \}, \text{represents object attributes of type } t. \]

SubCreator: \(S \rightarrow U\). For each subject SubCreator gives its creator.

For each \(att\) in \(UA \cup SA \cup OA\), \(SCOPE_{att}\) represents the attribute’s scope, a finite set of atomic values.

\[ \text{attType: } UA \cup SA \cup OA \rightarrow \{ \text{set, atomic} \}. \text{ It specifies attributes as set or atomic valued.} \]

PER represents finite set of operations.

Each attribute function maps elements in TReU, S and O to atomic or set values.

\[ \forall ua \in UA. ua : TReU \rightarrow \{ \begin{array}{ll} SCOPE_{ua} & \text{if attType(ua) = atomic} \\ 2^{SCOPE_{ua}} & \text{if attType(ua) = set} \end{array} \] \]

\[ \forall sa \in SA. sa : S \rightarrow \{ \begin{array}{ll} SCOPE_{sa} & \text{if attType(sa) = atomic} \\ 2^{SCOPE_{sa}} & \text{if attType(sa) = set} \end{array} \] \]

\[ \forall t \in OT. \forall oa \in OA_t. oa : O_t \rightarrow \{ \begin{array}{ll} SCOPE_{oa} & \text{if attType(oa) = atomic} \\ 2^{SCOPE_{oa}} & \text{if attType(oa) = set} \end{array} \] \]
**IaaSad Model**

### Part I. Basic Sets and Functions

CRU, TRU represent the cloud root user and tenant root user respectively.

TAU represents a finite set of tenant administrative users.

AR represents a set of administrative roles and UAR represent user-role assignment, i.e., UAR ⊆ TAU × AR.

### Part II. Operations

<table>
<thead>
<tr>
<th>Operations</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Operations for Cloud Root User</td>
<td></td>
</tr>
<tr>
<td>1.1 createTenant(req:CRU, tenant:NAME)</td>
<td>T' = T U {tenant}</td>
</tr>
<tr>
<td>1.2 createRootUser(req:CRU, u:NAME, tenant:T)</td>
<td>TRU = ∅, TRU = {u}</td>
</tr>
<tr>
<td>2. Operations for Tenant Root User</td>
<td></td>
</tr>
<tr>
<td>2.1 createUserAttr(req:TRU, ua:NAME, type: {set, atomic})</td>
<td>UA' = UA ∪ {ua}, attType(ua) = type</td>
</tr>
<tr>
<td>2.2 createSubAttr(req:TRU, sa:NAME, type: {set, atomic})</td>
<td>SA' = SA ∪ {sa}, attType(sa) = type</td>
</tr>
<tr>
<td>2.3 addSubConstr(req:TRU, policy:POLICY)</td>
<td>SubConstr' = SubConstr ∪ {policy}</td>
</tr>
<tr>
<td>2.4 createObjAttr(req:TRU, oa:NAME, type: {set, atomic}, oat:OT)</td>
<td>OA' = OA ∪ {oa}, attType(oa) = type, oaType(oa) = oat</td>
</tr>
<tr>
<td>2.5 addObjConstr(req:TRU, policy:POLICY)</td>
<td>ObjConstr' = ObjConstr ∪ {policy}</td>
</tr>
<tr>
<td>2.6 addAuthz(req:TRU, policy:POLICY)</td>
<td>Authz' = Authz ∪ {policy}</td>
</tr>
<tr>
<td>2.7 createAdminRole(req:TRU, adminrole:NAME)</td>
<td>AR' = AR ∪ {adminrole}</td>
</tr>
<tr>
<td>2.8 createAdminPolicy(req:TRU, policy:POLICY)</td>
<td>AdminPolicy' = AdminPolicy ∪ {policy}</td>
</tr>
<tr>
<td>2.9 addAdminUserRole(req:TRU, u:TReU, r:AR)</td>
<td>UAR' = UAR ∪ {(u, r)}</td>
</tr>
<tr>
<td>3.1 addUser(req:TAU, u:NAME)</td>
<td>TReU' = TReU ∪ {u}</td>
</tr>
<tr>
<td>3.2 add(req:TAU, u:TReU, att:UA, value:SCOPEatt)</td>
<td>att(u)' = att(u) ∪ {value}</td>
</tr>
<tr>
<td>3.3 delete(req:TAU, u:TReU, att:UA, value:SCOPEatt)</td>
<td>att(u)' = att(u) {value}</td>
</tr>
<tr>
<td>3.4 assign(req:TAU, u:TReU, att:UA, value:SCOPEatt)</td>
<td>att(u)' = value</td>
</tr>
</tbody>
</table>
Proof of concept in OpenStack

Requests from Cloud Users

- Swift (Object Store)
- Glance (Image)
- Nova (Compute)
- Cinder (Block Storage)
- Quantum (Network)
- Others

Keystone (Identity)
OpenStack Authorization for Nova

User

1. (user_name, password)
2. Verify (user_name, password).
   Sign user data and generate a token
   end_point = Nova.
   Return (Token, end_point)
3. (Request, token)
4. Request Token
   Revocation List
5. List of Revoked Token ID
6. Fetch Object Info
7. Return Object Info
8. Decode token and send user
   info, request, object info
9. Load Policy
10. Return Policy
11. Evaluate and return result
12. Respond to user request

Keystone
Nova PEP
Nova PIP
Nova PDP
Nova PAP

World-Leading Research with Real-World Impact!
ABAC Enforcement in OpenStack

Enhanced Keystone

Enhanced Nova

Enhanced Other Service

Enforcement Model

World-Leading Research with Real-World Impact!
Alternative Enforcement Models

Enhanced **Nova**

PEP

PIP

Objects and object attribute assignment

1. Forward request (s, o, op)

2. Return true or false

PDP

PAP

IaaSOp Policy + Object attribute constraint

PolicyEngine Service

(a) Enforcement Model II

Enhanced **Cinder**

PEP

PIP

Objects and object attribute assignment

Other Components

3. Forward request (s, o, op)

4. Return True or false

PDP

PAP

IaaSOp Policy + Object attribute constraint

PolicyEngine Service

(b) Enforcement Model III

Enhanced **Nova**

PEP

Object Attribute Store Service

Enhanced **Cinder**

PEP

Other Components

1. Retrieve object attribute

2. Return object attributes

PIP

Object attribute assignment and constraint policy

---

World-Leading Research with Real-World Impact!
Conclusion

Summary

- We illustrate the case of access control in cloud IaaS
- We summarize four core requirements of access control models
- Existing models fail to satisfy those requirements
- By connecting existing models with additional features, we proposed IaaSop and IaaSad models based on ABAC

Future work

- Different types of attributes: system wide, service-specific attributes.
- Various types of subject attributes constraints, object attribute constraints.
- Reachability analysis on IaaSop and IaaSad instance.
Thanks. Questions?