An Attribute-Based Access Control Extension for OpenStack and its Enforcement Utilizing the Policy Machine

Prof. Ravi Sandhu
Executive Director and Endowed Chair

2nd IEEE International Conference on Collaboration and Internet Computing (CIC)
November 1-3, 2016

Smriti Bhatt, Farhan Patwa and Ravi Sandhu
Department of Computer Science

ravi.sandhu@utsa.edu
www.profsandhu.com
www.ics.utsa.edu
Outline

- Introduction
- RBAC and ABAC
- Simplified OpenStack Access Control (OSAC) Model
- An ABAC Extension for OpenStack
- Policy Machine (PM) and its Architecture
- ABAC Enforcement Architecture
- Authorization using AE and PM
- Use Cases
- Evaluation
- Discussion and Analysis
- Conclusion and Future Work
RBAC:
- Most dominant access control model
- Major cloud computing platforms:
  - OpenStack
  - AWS
  - Microsoft Azure

Limitations:
- Role explosion

ABAC:
- Access control based on attributes
- Enhanced flexibility and fine grained access control
- Implement ABAC models in real-world applications
- A gradual shift from RBAC to ABAC models

Authorization based on RBAC model
Three different ways of combining RBAC and ABAC by NIST –

- **dynamic roles**: uses user and context attributes to dynamically assign roles to users
- **attribute-centric**: roles just another attribute of users with no special semantics
- **role-centric**: constrains role permissions based on user attributes

- Proposed a role-centric ABAC extension for OpenStack
- Combining advantages of both RBAC and ABAC
Simplified OpenStack Access Control (OSAC) Model

Union of role permissions based on the token of a user

Union of role permissions


Groups and Domains removed

© Ravi Sandhu
User-Attribute Enhanced OSAC in Single Tenant

Union of attribute-value permissions

User-Attribute Permission Assignment (UAPA)

Role-Centric ABAC

Many-to-Many
One-to-Many
One-to-One

(Union of role permissions \( \cap \) Union of attribute-value permissions) based on the token of a user
An ABAC Extension for OpenStack

- Extended simplified OSAC with user attributes –
  - $U_{Att}$ – a finite set of user attribute functions
  - For each $uatt$ in $Uatt$, $\text{Range}(uatt)$ is a finite set of atomic value
  - $UAPA$ – user-attribute value permission assignment

- For any user –
  - maximum permissions determined based on roles in the token,
  - further constrained by permissions associated to its user-attribute values

- Currently, model designed for atomic valued attributes only
- Object attributes next challenge to explore
- Enforced this model in OpenStack utilizing the Policy Machine (PM)
General-purpose attribute-based access control framework

Express and enforce arbitrary access control policies

Provide a unified platform supporting:

- Commonly known and implemented access control policies
- Combinations of policies
- New access control policies

PM Core Elements
- Users
- Objects
- User Attributes
- Object Attributes
- Operations, Access Rights
- Processes
- Policy Classes

PM Relations
- Assignment
- Association
- Prohibition
- Obligation

Assignment — for specifying relationships between policies, users, and user attributes, objects, and object attributes

Association — for defining policies through associations between user attributes and object attributes or objects through some operations
Policy Machine Architecture (Adapted from [*])

An ABAC Enforcement Architecture for OpenStack using PM

Active Directory (AD): stores all the users and their associated user attributes in the system (the same AD also to be Identity backend for OpenStack)
Authorization Sequence Diagram

1. Issues service-specific commands

2. Access request via RESTful API

3. Query PM based on policy defined in PM

4. Return allowed permissions based on roles and attributes

5. Evaluate and return access decision (Grant/Deny)

6. Respond to user request – Grant → show results Deny → return error

OpenStack Authorization using AE and PM

© Ravi Sandhu

World-Leading Research with Real-World Impact!
Use Case I

A Simplified OSAC RBAC Policy

- Roles: \{Admin, Manager\}
- Commands (c):
  - compute extension-keypair-index
  - compute extension-keypair-create
  - compute extension-keypair-delete
  - compute extension-keypair-show
- Authorization rules for any user \(u\):
  - compute extension-keypair-create \(\Rightarrow\) \(\text{Role}(u) = \text{Admin}\)
  - compute extension-keypair-delete \(\Rightarrow\) \(\text{Role}(u) = \text{Admin}\)
  - compute extension-keypair-index \(\Rightarrow\) \((\text{Role}(u) = \text{Admin} \lor \text{Role}(u) = \text{Manager})\)
  - compute extension-keypair-show \(\Rightarrow\) \((\text{Role}(u) = \text{Admin} \lor \text{Role}(u) = \text{Manager})\)

A Role-Based Access Control Policy in PM
Use Case I

OpenStack Enforcement Results

Role = Admin

Role = Manager
Use Case II

- A Role-Centric ABAC Policy
  - Roles: \{Admin, Manager\}
  - Department: \{IT, OPS\}
  - Commands (c):
    - `compute extension-keypair-index`
    - `compute extension-keypair-create`
    - `compute extension-keypair-delete`
    - `compute extension-keypair-show`
  - Authorization rules for any user \(u\):
    - `compute extension-keypair-create \rightarrow (Role(u) = Admin \land Dep(u) = IT)`
    - `compute extension-keypair-delete \rightarrow (Role(u) = Admin \land Dep(u) = IT)`
    - `compute extension-keypair-index \rightarrow ((Role(u) = Admin \lor Role(u) = Manager) \land (Dep(u) = IT \lor Dep(u) = OPS))`
    - `compute extension-keypair-show \rightarrow ((Role(u) = Admin \lor Role(u) = Manager) \land (Dep(u) = IT \lor Dep(u) = OPS))`
A User-Attribute Enhanced OSAC Policy in PM
Use Case II

OpenStack Enforcement Results

Role = Admin and Department = OPS

Role = Admin and Department = IT
Network latency
Proof-of-concept implementation
Need for Optimization

Indications of cost of implementing our ABAC extension in OpenStack using PM
Advantages of an ABAC extension with user attributes:

- Define more fine-grained access control policies
- Significantly reduce number of roles required in a policy
- Avoid problems such as role explosion and role-permission explosion

Trade-off between performance and enhanced functionality/capability

Performance improvement techniques:

- High-performance server to host PM and AE to improve policy evaluation time
- Cache policy evaluation results locally
- Install PM, AE, and OpenStack services on an isolated subnet
Conclusion and Future Work

- Proposed an ABAC extension with user attributes for OpenStack
- Enforced our model in OpenStack utilizing the PM and AE
- An initial attempt to facilitate transition towards ABAC models in real world applications

Future Work:

- Explore other capabilities of PM – combination of access control polices, attribute and role hierarchy
- Apply performance enhancements to enforcement framework
- Include object attributes
Thank you!!
Questions??