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

Smriti Bhatt, Farhan Patwa and Ravi Sandhu
Institute for Cyber Security and Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas, USA
bhattsmriti1@gmail.com, farhan.patwa@utsa.edu and ravi.sandhu@utsa.edu

***Abstract*—Role-Based Access Control (RBAC) has been the dominant access control model in industry since the 1990s. It is widely implemented in many applications, including major cloud platforms such as OpenStack, AWS, and Microsoft Azure. However, due to limitations of RBAC, there is a shift towards Attribute-Based Access Control (ABAC) models to enhance flexibility by using attributes beyond roles and groups. In practice, this shift has to be gradual since it is unrealistic for existing systems to abruptly adopt ABAC models, completely eliminating current RBAC implementations.**

In this paper, we propose an ABAC extension with user attributes for the OpenStack Access Control (OSAC) model and demonstrate its enforcement utilizing the Policy Machine (PM) developed by the National Institute of Standards and Technology. We utilize some of the PM's components along with a proof-of-concept implementation to enforce this ABAC extension for OpenStack, while keeping OpenStack's current RBAC architecture in place. This provides the benefits of enhancing access control flexibility with support of user attributes, while minimizing the overhead of altering the existing OpenStack access control framework. We present use cases to depict added benefits of our model and show enforcement results. We then evaluate the performance of our proposed ABAC extension, and discuss its applicability and possible performance enhancements.

***Keywords*—Policy Machine; Attribute-Based Access Control; OpenStack; Authorization Engine;**

I. Introduction

Role Based Access Control (RBAC) model [2]–[4] is the most widely used access control model in industry. Many different applications and platforms use some customized form of RBAC based on their needs and requirements. Major cloud computing platforms such as OpenStack [5], AWS [6], and Microsoft Azure [7] utilize RBAC as their authorization foundation. Besides many well-known advantages of RBAC [8], it also has some well-known limitations [9] such as role explosion and role-permission explosion.

In RBAC, access control policies can only be defined on basis of roles which restricts access control flexibility. On the other hand, Attribute-Based Access Control (ABAC) provides great flexibility to express fine-grained access control policies in a simple and more powerful way based on attributes of users, subjects, and objects [10]–[12]. With the advancements of ABAC and its capabilities, there is an inevitable need for implementing ABAC models in real-world applications and systems. However, it is difficult for existing systems to instantaneously adapt to attribute-based access control policies since they require a well-defined and robust attribute and access control management system for their implementation. We believe the transformation from RBAC to ABAC policies has to be gradual but we will eventually see wide adoption and implementation of ABAC models in the industry. In particular combining ABAC with roles is one promising transition path.

NIST has identified three different ways of combining RBAC and ABAC effectively adding attributes to role-based access control policies. First is *dynamic roles* which uses user and context attributes to dynamically assign roles to users. This is similar to attribute-based user-role assignment [13]. Second is *attribute-centric* where roles are just another attribute of users with no special semantics. The third is *role-centric* which constraints the permissions of a role based on user attributes [14], [15].

In this paper, we propose a role-centric ABAC model for OpenStack by adding user attributes to the core OpenStack Access Control (OSAC) model [1]. This allows us to extend existing RBAC in OpenStack with attributes in order to combine the advantages of both models. We demonstrate the enforcement of our model in OpenStack using a general-purpose attribute-based access control framework developed by NIST, known as the Policy Machine (PM) [16], [17]. PM enables expression and enforcement of different types of access control policies through its policy configuration points. It provides a unifying framework to define, administer and enforce commonly known access control policies as well as new access control policies.

To facilitate communication between OpenStack and PM, we implemented a RESTful service called the Authorization Engine (AE). AE is a proof-of-concept implementation which acts as an authorization component by getting information and permissions from a PM server and evaluating authorization decisions. These decisions are then returned to OpenStack services where the policy is enforced. We also present use cases to show how existing RBAC and proposed role-centric ABAC policy would work in an organization-specific environment. Finally, we present performance evaluations followed by a discussion and analysis of our ABAC extension with user attributes to OpenStack and the AE enforcement architecture.

The rest of the paper is organized as follows. Section 2 briefly discusses related work and presents background on OpenStack access control model, and the Policy Machine. Our role-centric ABAC extension for OpenStack is presented and defined in Section 3. Section 4 discusses the enforcement and implementation details. Section 5 presents use cases of access

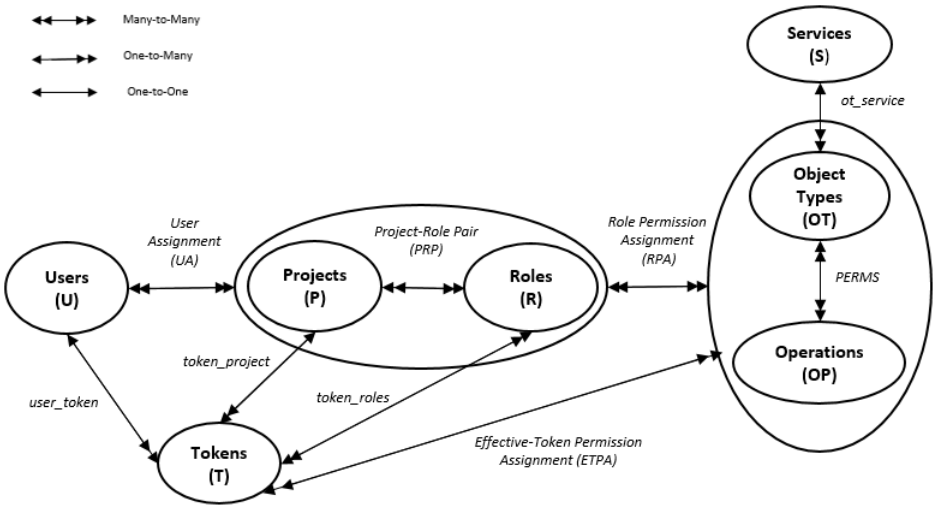

Figure 1: Simplified OpenStack Access Control (OSAC) Model (Adapted from [1])

control policies defined in PM and enforced in OpenStack. The performance evaluation, followed by a discussion and analysis, is presented in Section 6. Section 7 gives our conclusions and thoughts on future work.

II. Related Work and Background

In this section, we briefly discuss prior efforts on adding attributes to the OpenStack access control and authorization framework. The applicability of ABAC in OpenStack has been studied in different scenarios. Approaches to include attributes in OpenStack for cloud federation and federated identity management are discussed by Chadwick et al [18] and by Lee and Desai [19]. Pustchi and Sandhu [20] discuss the application of ABAC to enable collaboration between tenants in a cloud IaaS platform. In contrast our ABAC extension is focused on authorization in OpenStack within a single tenant.

Jin et al. [12] presented a unified ABAC model that can be configured to do mandatory, discretionary and role-based access control, and demonstrated an OpenStack implementation [21] as a replacement for the native OpenStack RBAC. Our approach is specifically designed to incorporate OpenStack's existing RBAC with an ABAC extension. A formal role-centric attribute-based access control (RABAC) model has been proposed by Jin et al. [15], along with XACML [22] profiles for this model. XACML is a general-purpose access control policy language for managing access to resources.

Our role-centric ABAC extension for OpenStack, in contrast, includes only user attributes as a first step leaving object attributes for future work. In cloud environments the objects are created on-demand, hence cannot be specified in the policy during policy configuration [1]. Besides this, we are utilizing the PM, a different attribute-based access control framework than XACML. PM is an open-source and freely available software by NIST [23] that facilitated us to build a customized authorization engine component for our ABAC extension for OpenStack. PM's flexibility and capabilities made it a desirable choice for our enforcement framework.

We now present background on the OpenStack Access Control (OSAC) model, and the Policy Machine (PM).

A. Simplified OpenStack Access Control (OSAC) Model

A core OSAC model was presented by Tang and Sandhu [1], based on the OpenStack Identity API v3 and Havana release. For ease of exposition, we simplify this model by removing **Groups** and **Domains**. The resulting simplified OSAC model is shown in Figure 1 and is defined as follows.

Definition 1. Simplified OSAC model has the following core and derived components.

1.1 Core Components:

- U, P, R, S, OT, OP and T *are finite sets of users, projects, roles, services, object types, operations and tokens respectively*
- $PRP = P \times R$, *is the set of project-role pairs*
- $PERMS = OT \times OP$, *is the set of permissions*
- $UA \subseteq U \times PRP$, *is a many-to-many user to project-role assignment relation*
- $RPA \subseteq PERMS \times R$, *is a many-to-many permission to role assignment relation*
- $ot\_service : OT \rightarrow S$, *is a function mapping an object type to its associated service*
- $user\_tokens : U \rightarrow 2^T$, *is a function mapping a user to a set of tokens; correspondingly,* $token\_user : T \rightarrow U$, *is a mapping of a token to its owning user*
- $token\_project : T \rightarrow P$, *is a function mapping a token to its target project*

1.2 Derived Components:

- $token\_roles : T \rightarrow 2^R$, *is a function mapping a token to its set of roles, formally,* $token\_roles(t) = \{r \in R \mid (token\_user(t), (token\_project(t), r)) \in UA\}$

- $ETPA : T \rightarrow 2^{PERMS}$, *is a function specifying the permissions available to a user through a token, formally,* $ETPA(t) = \bigcup_{r \in token\_roles(t)} \{perm \in PERMS \mid (perms, r) \in RPA\}$.

Simplified OSAC comprises seven entities: **users, projects, roles, services, object types, operations,** and **tokens**. **Groups** and **domains** are removed in this simplified model since groups are mere collection of users, and users in OpenStack belongs to a single domain or tenant (where they are created) and can be seen and managed only by the domain owner or administrator. The scope of simplified OSAC is within a single domain, thus the administrative boundary of domains is not relevant in this context.

Users are individuals authenticated to access cloud resources, **projects** are resource containers through which users get access to specific cloud resources such as virtual machines (VMs), storage, etc. **Roles** are global entities used to associate users with any of the projects inside a domain. **Permissions** are assigned to role-project pairs and are used to specify access levels of users to services in specific projects, with specific roles. Role-permission assignments are defined by a cloud administrator. **Object types** are different resources in cloud services such as virtual machines (VMs), images, swift-objects etc. **Operations** are access methods on these object types owned by **services** in the cloud.

Each authenticated user receives a **token** from the identity service Keystone, which defines the scope of resources that a user is allowed to access. A token is equivalent to a subject and has information of the user, its roles and its associated projects [24]. The derived components are derived from the core components such as **token_roles** and **Effective Token Permission Assignment (ETPA)**. ETPA is based on the token a user presents during access requests, and permissions are identified based on roles present in a token for a specific user, in specific projects. This is a RBAC model and has limitations as discussed earlier. To overcome these limitations, we extend the simplified OSAC model by adding user attributes. The extended model and its details are presented in Section 3.

B. The Policy Machine

Policy Machine (PM) [16] is a general-purpose attribute-based access control framework which can express and enforce arbitrary, organization specific, attribute-based access control policies. It is a mechanism to define access control policies in terms of a standardized and generic set of relations and functions that can be reused. The main objective of PM is to provide a unifying framework to support a wide range of attribute-based policies or policy combinations.

The PM has eight core elements or entities: **users, objects, user attributes, object attributes, operations, processes, access rights** and **policy classes**. The policy classes, user attributes and object attributes are containers for policies, users and objects respectively.

The PM has four types of relations: **assignment, association, prohibition** and **obligation**, and two sets of functions: **access control decisions** and **policy enforcement**. PM also supports hierarchical relations through containment.

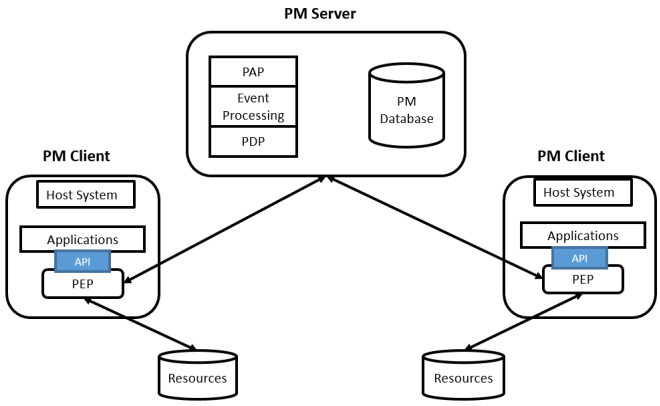

Figure 2: Policy Machine Architecture (Adapted from [16])

In this paper, we utilize only two types of relations *assignment*—for specifying policies, users, and user attributes, and *association*–for making association between user attributes and object attributes or objects through some operations.

The general PM architecture is shown in Figure 2. It comprises of PM server, PM clients and Resources repository. PM server includes a PM Database (Active Directory), a Policy Decision Point (PDP), a Policy Administration Point (PAP) and an event processing module. The PM clients are host systems where the policies are enforced. They encapsulate the application programming interfaces (API) and PM-aware applications. In current PM implementation, the Policy Enforcement Point (PEP) is implemented as a kernel simulator. The PM client or the user environment is the context in which the user's PM processes run. These processes are similar to subjects. A PM client could be an operating system, an application (e.g., a database management system), a service in a service oriented architecture, or a virtualized environment. The resources are the repositories for different types of objects such as files, records, directories, etc. [17].

Policy Machine supports a rich set of capabilities for defining customized access control policies. It allows to specify deny or prohibition relations and apply constraints, and also allows to combine different access control policies specified in PM using the Admin tool. PM Admin tool is a GUI based tool, used to define access control policies in PM by creating policy classes, users, user attributes, objects, object attributes, and setting operations sets and permissions between user attributes and object attributes and objects. All this data, information and relations are stored in Active Directory (the PM Database). Users request access to objects through PM clients which in turn communicate to the PM server for access control decisions and enforce these decisions on host systems.

III. An ABAC Extension for OpenStack

In this section, we propose a role-centric ABAC model for OpenStack by extending the simplified OSAC model.

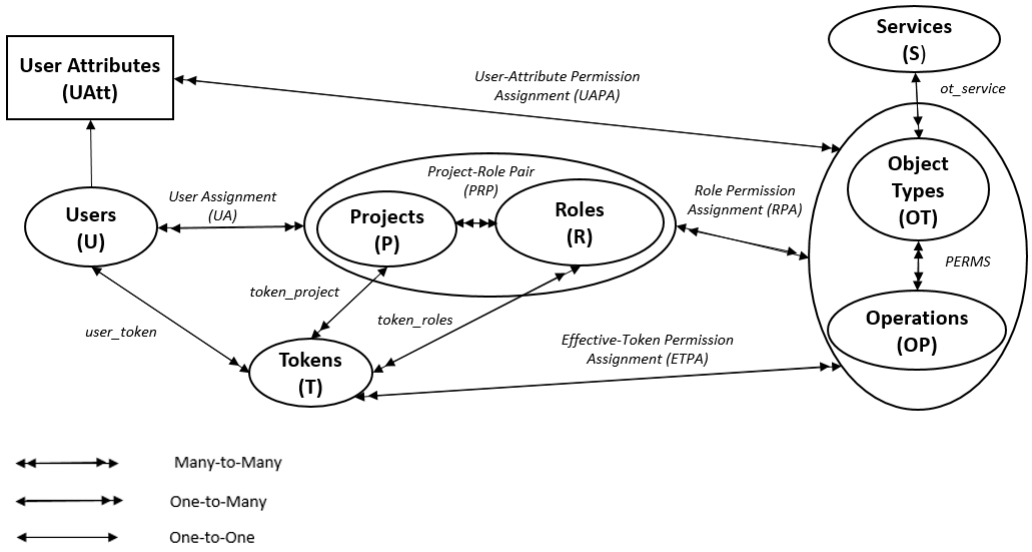

Figure 3: User-Attribute Enhanced OSAC in Single Tenant

This model is called the User-Attribute Enhanced OSAC model and is depicted in Figure 3. It has all the core components and derived components of simplified OSAC model along with some newly added elements and relations. **User attributes** are added to the **Users** and a new relation **UAPA** is introduced for user-attribute value and permission assignment. This model is a role-centric ABAC [14] model in the sense that it incorporates the existing RBAC framework of OpenStack, keeping all its advantages, and adds in the flexibility of ABAC model by introducing user attributes. A user's roles determine the maximum permissions which are further reduced by user attribute permission assignments. The newly added components are defined as follows.

Definition 2. User-Attribute Enhanced OSAC model has the following additional and modified components besides simplified OSAC model.

2.1 Additional Core Components

- $UAtt$ is a finite set of user attribute functions
- $Range(uatt)$ where $uatt \in UAtt$, is a finite set of atomic values defined for each user attribute function in $UAtt$
- For each $uatt$ in $UAtt$, $uatt : U \rightarrow Range(uatt)$, is a mapping of each user to an atomic value in $Range(uatt)$
- $UAPA \subseteq PERMS \times \{\langle uatt, v\rangle \mid uatt \in UATT; v \in Range(uatt)\}$, is a many to many permission to attribute-value assignment relation

2.2 Modified Derived Component

- $ETPA : T \rightarrow 2^{PERMS}$, is a function specifying the permissions available to a user through a token and user-attribute value assignment, formally $ETPA(t) = \bigcup_{r \in token\_roles(t)} \{perm \in PERMS \mid (perm, r) \in RPA\} \cap \bigcup_{uatt \in UAtt} \{perm \in PERMS \mid (perm, \langle uatt, uatt(token\_user(t))\rangle) \in UAPA\}$

User Attribute is a function which takes a user and returns a specific value from its range, where the **range** of an attribute is a finite set of atomic values defined for each attribute function. In general attributes are of two types—*atomic valued* attribute returns only one value from its range; and *set valued* attribute returns some subset of the range. User attributes represents characteristics or properties of the users, some examples are Department, Clearance, and Specialization [12], [15]. The User-Attribute Enhanced OSAC model only allows atomic-valued attributes for users. **UAPA** is a set of permissions associated with user attributes and their assigned values. **ETPA** has been modified to include permissions from user attributes, in addition to permissions from roles.

Scope and Assumptions

In the User-Attribute Enhanced OSAC model, we require that each user attribute is atomic valued. Every user is associated with a finite set of user attribute functions whose values are assigned and modified by security administrators. The details of this process are outside the scope of this paper. For any user, all the associated user attributes are always active during their lifetime in any defined policy.

IV. Enforcement and Implementation

This section presents the enforcement and implementation details of enforcing user-attribute enhanced OSAC in OpenStack utilizing the PM. OpenStack is a rapidly changing open-source cloud platform that provides an architecture to use or enhance its services as per the requirements. OpenStack and PM, both being open-source, provide the capability to

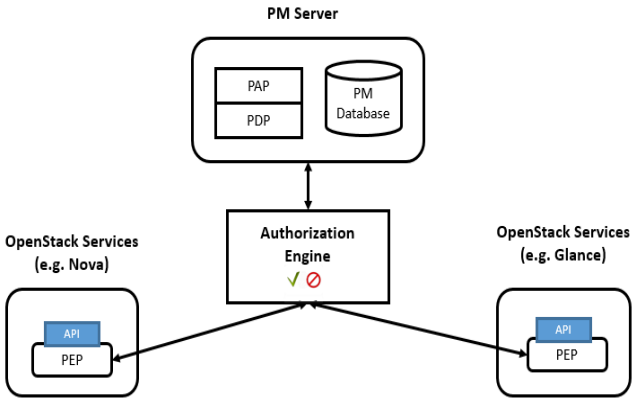

Figure 4: An ABAC Enforcement Architecture for OpenStack using PM

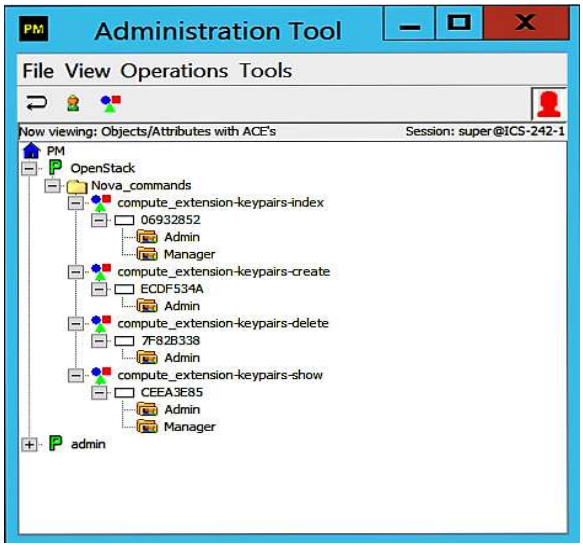

Figure 5: OpenStack Authorization using AE and PM

integrate our own implementation in the OpenStack authorization framework. In order to enforce our ABAC extension, user-attribute enhanced OSAC model, on OpenStack using PM, we need a customized authorization engine. Thus, an authorization engine (AE), which is a RESTful service, has been implemented as a proof-of-concept which acts as an interface between OpenStack and PM. In the enforcement framework, OpenStack Kilo release with Identity API version 2 and PM version 1.5 have been used. As per discussions with NIST PM team, a new version—PM 1.6 is soon to be released with new features and better performance. The newer release addresses the performance issues observed in PM 1.5.

A. Enforcement Architecture

The enforcement architecture is shown in Figure 4 and has been adapted from PM architecture in Figure 2. In this architecture, we utilize the PM server as a centralized policy administration point that returns a set of user permissions on objects based on the policy definitions. It is connected to an active directory (AD), a back-end database for PM that stores all the users and their associated user attributes. We also assume that OpenStack uses the same AD as its user identity back-end in order to store all users related information, including attributes. Due to dynamic nature of cloud objects, we consider OpenStack commands as objects in the policy defined in PM. The commands are specific to each service in OpenStack such as Nova, Glance, Cinder, etc. The policy definitions, typically listed in OpenStack policy file have been defined in PM Admin Tool in an "OpenStack" policy class. One instance of OpenStack policy class, from "Objects/Attributes with ACE's" view, is shown in Figure 5.

This is a sample policy defined for Nova *keypair* commands in OpenStack. These commands are used to generate ssh keys for a user which in turn are used while creating VMs. In Figure 5, *Admin* and *Manager* are roles (depicted as user attributes in PM), and *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show* are Nova commands (depicted as objects in PM). The user attributes and objects in PM are associated with a specific operation set, e.g., *"06932852"* is an operation set with "read" permissions defined between user attributes—*Admin* and *Manager*, and object—*compute_extension-keypair-index*. That means that attribute values *Admin* and *Manager* have *read* permissions on *compute_extension-keypair-index*, so that a user with any of these roles is authorized to do this operation in OpenStack.

The enforcement architecture utilizes a server-client architecture where PM server and AE has server-client relationship, and similarly AE, itself, acts as a server for different services of OpenStack. OpenStack services communicate, through a RESTful API, to AE which in turn communicates to PM server for making authorization decisions.

B. Authorization Engine

The authorization engine (AE) is an interface for communication between PM server and different OpenStack services. It is written in Java using a RESTful API and acts as a RESTful server for OpenStack services. For any operation to be performed by a **user** in OpenStack, AE initially verifies the **project** in the target and the token. Then, it connects to PM server and queries it via different PM commands to make access control decisions based on our ABAC extension for OpenStack.

AE replaces the existing policy engine in OpenStack and is responsible for evaluating the policy defined in PM and returning access decisions to OpenStack services. OpenStack Services (S) are the policy enforcement points (PEP) that enforces the access decisions returned by AE and responds to the users with appropriate results.

A sequence diagram presenting authorization in OpenStack using AE and PM is shown in Figure 6. It shows the sequence of actions involved in the authorization process. Currently, AE works with two types of policies, a RBAC policy (same as

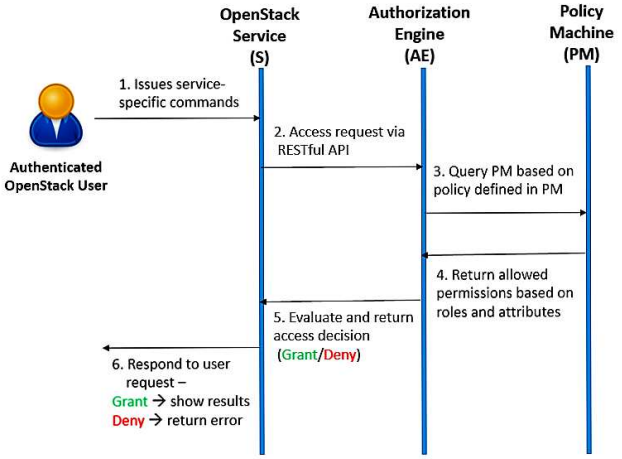

Figure 6: OpenStack Authorization using AE and PM

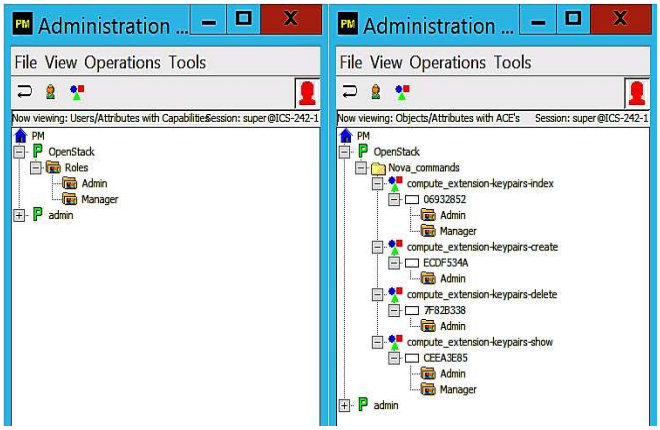

Figure 7: A Role-Based Access Control Policy in PM

OpenStack's current access control policy), and a role-centric ABAC policy with user attributes (our user-attribute enhanced OSAC), and can be easily extended to enforce other types of access control policies as required.

This is a proof-of-concept implementation and is mainly designed to show the applicability and feasibility of our proposed model in OpenStack cloud platform. However, it has not been optimized for performance. AE can be designed to be a more general and independent component which could be used with any policy-configuration tool, like PM, and be applicable to other cloud platforms besides OpenStack.

V. Use Case

Here, we present two use cases—first with only roles, and second with roles and user attributes. These use cases respectively illustrate how a simplified OSAC model and a user-attribute enhanced OSAC model can be enforced in OpenStack using PM and AE.

A. A Simplified OSAC RBAC Policy

In this first use case, we present a sample RBAC policy, equivalent to simplified OSAC policy in OpenStack, with two roles *Admin* and *Manager*, and four $Nova\_commands$: *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show*. These are example commands that we choose for the use case. Similar to these commands, we can define authorization policies for other commands in different services of OpenStack. The permissions are determined by the roles that a user has been assigned in a specific project. The Nova *keypair* commands are used to generate ssh keys for a user. These keys are used while creating VMs. A user can create, delete, list, and show details of keypairs using these commands. The authorization rules for each command, for a generic user *u* are given below.

Roles: {*Admin, Manager*}

Commands (c): *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show*

Authorization rules for any user u:

- $compute\_extension\text{-}keypair\text{-}create \rightarrow Role(u) = Admin$
- $compute\_extension\text{-}keypair\text{-}delete \rightarrow Role(u) = Admin$
- $compute\_extension\text{-}keypair\text{-}index \rightarrow (Role(u) = Admin \vee Role(u) = Manager)$
- $compute\_extension\text{-}keypair\text{-}show \rightarrow (Role(u) = Admin \vee Role(u) = Manager)$

Here, the rules state that a user must have an *Admin* role to create or delete keypairs, whereas *compute_extension-keypair-index* and *compute_extension-keypair-show* are authorized to be performed by an *Admin* role or *Manager* role. To enforce this authorization policy, we defined an equivalent policy to authorize each of these commands in PM. Figure 7 shows the policy defined in PM. There are two roles defined *Admin* and *Manager* as shown on the left side, and associations between commands and roles via operation sets (alpha-numerically named by default) are shown on the right in Figure 7. As shown on the right side, *compute_extension-keypair-index* can be done by *Admin* or *Manager*, whereas *compute_extension-keypair-create* and *compute_extension-keypair-delete* can be done by *Admin* only. On OpenStack end, we defined two roles *Admin* and *Manager* and assigned a single role or a combination of these roles to some users in a test tenant, for example *user1* is an *Admin* and *user2* is a *Manager*. The policy was tested in OpenStack by executing different commands for these users. A screenshot of the authorization results for few commands are shown in Figure 8.

B. A Role-Centric ABAC Policy

Here, we present a role-centric ABAC policy with user attributes only. This is an example of our user-attribute enhanced OSAC policy. Besides roles and commands, there is a user attribute—**Department** that can be assigned only one value from its **Range** = $\{IT, OPS\}$ for any user. User attributes are atomic valued unlike roles, which implies that a user can have

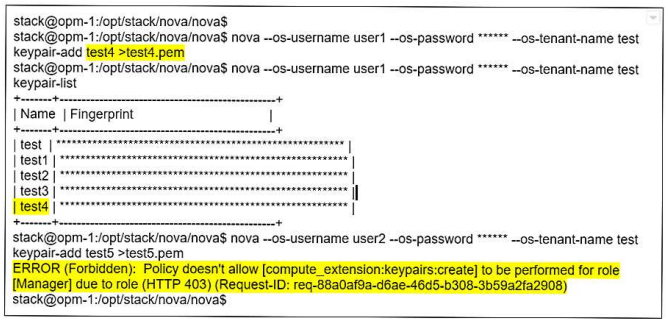

Figure 8: OpenStack Enforcement Results

multiple roles assigned but it can only be in one department, either *IT* or *OPS*. For any user, accesses are defined based on their roles and their associated user attributes. In a real organization, there are multiple roles and user attributes, and permissions are assigned based on roles and user attributes.

The authorization policy is defined based on roles and user attributes and is given below. As per the first rule, a user u is authorized to *compute_extension-keypair-create* only if the user has been assigned role *Admin* and the user is in department *IT*. Both of these requirements need to be true, otherwise, a user without the specified role and user-attribute value will be denied access. The second rule is similar. In the third rule, a user u needs to have an *Admin* or *Manager* role along with user-attribute value as *IT* or *OPS*. This is a role-centric policy which means that first the role is checked, then only user-attribute value will be checked. If a role check fails, then access is denied and user-attribute value is not checked.

Roles: $\{Admin, Manager\}$
Department: $\{IT, OPS\}$
Commands (c): *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show*
Authorization rules for any user u:

- $compute\_extension\text{-}keypair\text{-}create \rightarrow (Role(u) = Admin \wedge Dep(u) = IT)$
- $compute\_extension\text{-}keypair\text{-}delete \rightarrow (Role(u) = Admin \wedge Dep(u) = IT)$
- $compute\_extension\text{-}keypair\text{-}index \rightarrow ((Role(u) = Admin \vee Role(u) = Manager) \wedge (Dep(u) = IT \vee Dep(u) = OPS))$
- $compute\_extension\text{-}keypair\text{-}show \rightarrow ((Role(u) = Admin \vee Role(u) = Manager) \wedge (Dep(u) = IT \vee Dep(u) = OPS))$

Policy definition in PM is shown in Figure 9. Unlike the previous use case, there is an additional PM container *Attributes* which contains user attribute *Department* with two values *IT* and *OPS*. Associations shown on the right side also include user attributes besides the roles. In OpenStack, we defined a number of users with different roles and user attribute values

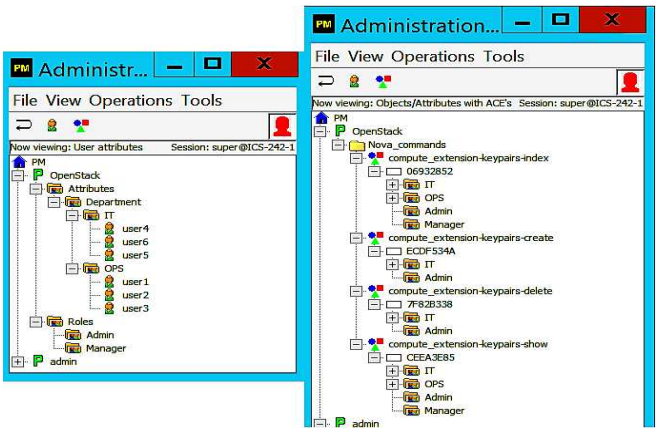

Figure 9: A User-Attribute Enhanced OSAC Policy in PM

and tested access for them against desired results. For example, a user *user1* is defined with role *Admin* and user attribute department as *OPS*, and another user *user4* is defined with role *Admin* and user attribute department as *IT*. In this case, *user4* has access to execute command *compute_extension-keypair-create*, whereas *user1* is denied access due to the department attribute value *OPS*. A screenshot of results in OpenStack is presented in Figure 10.

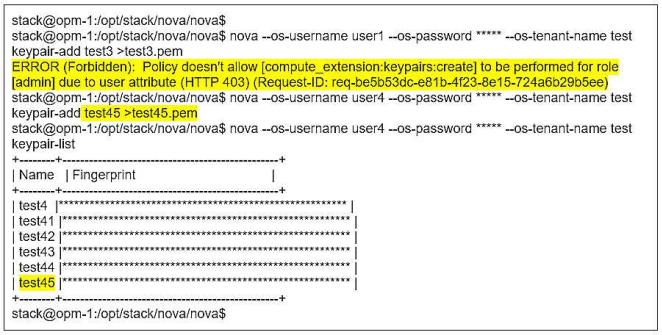

Figure 10: OpenStack Enforcement Results

VI. Evaluation

In this section, we present the details of our experiments carried out for performance evaluation. We discuss our results and analyze the applicability of our enforcement model with possible performance enhancements. The experiments were performed on two types of policies discussed in the use cases. We created scripts to test performance of Nova commands and compute the time taken for a set of number of requests (Nova command). The graph in Figure 11 shows the overall time for a set of requests executed by an OpenStack user. Here, our main objective is to evaluate the time taken for authorization decisions in existing OpenStack access control framework and an ABAC extended OpenStack access control framework utilizing the PM and AE.

The results showing only the policy check time in OpenStack are depicted in Figure 12. These graphs contain three

curves, one for a RBAC policy in OpenStack without any modifications (OS_RBAC), second for the same RBAC policy in OpenStack with modified policy engine (i.e. AE) and the PM (OS_PM_RBAC), and third for our user-attribute enhanced role-centric policy in OpenStack with AE and PM (OS_PM_ABAC). The overall request-response time for these three cases are quite similar, however, the policy check time for each of these cases significantly differ from each other. A centralized policy administration point PM, and a RESTful service AE in between PM and OpenStack adds latency in policy evaluation time.

Discussion and Analysis

There is always some trade-off between performance and enhanced functionality or capability. In this paper, our main goal is to enforce our proposed ABAC extension for OpenStack in an OpenStack platform. This model incorporates all the benefits of existing OpenStack access control mechanism and adds great flexibility by adding user attributes. These user attributes included in the policy allows a user to be assigned least possible permissions and allows to define more fine-grained access control policies avoiding problems such as role explosion and role-permission explosion.

Generally, an admin user is supposed to have maximum permissions but in a real enterprise, there are various departments and there could be an admin for each department. Similarly, there are many employees working in a department having different skill levels. Thus, creating a role for each of these combinations result in role explosion. However, having user attributes such as department and skills along with role admin in a policy avoids such problems and significantly enhances the capability and flexibility of access control policy.

Performance evaluations of our ABAC extension against existing OSAC gives an idea of the cost of applying it in real systems. However, there are a number of reasons for the performance latency. First, our implementation is a proof-of-concept implementation and hasn't been optimized for performance yet. It was rather driven by applicability of the model in a real platform, i.e., OpenStack. PM, the tool utilized to enforce this model, is mainly designed as a general attribute-based access control framework but needs some performance enhancements to be used in a real production environment. Also, there is network latency contributing to the time for each request from OpenStack to a centralized PM server which is currently residing on a node in different subnet.

We identify here a few techniques to achieve performance improvements: i) using performance enhanced server to host PM and AE in order to improve policy evaluation time, ii) caching policy evaluation results locally on the services, and iii) having PM, AE, and OpenStack services installed on an isolated subnet, similar to a typical OpenStack installation in a production environment.

VII. Conclusion and Future Work

In this paper, we proposed an ABAC extension for OpenStack and enforced it utilizing the PM and AE. We presented

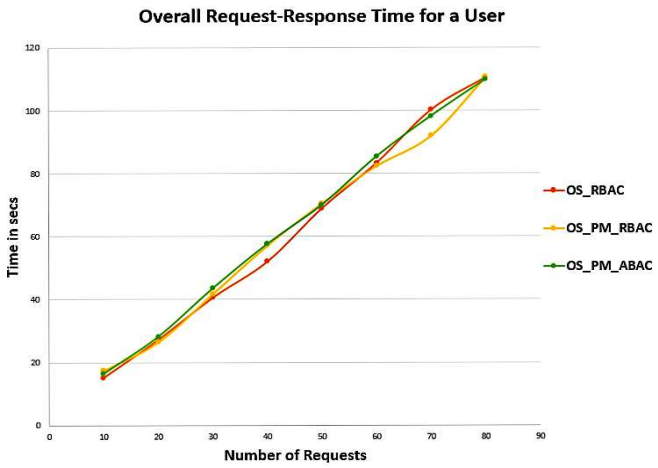

Figure 11: Overall Time Taken in Requests-Response

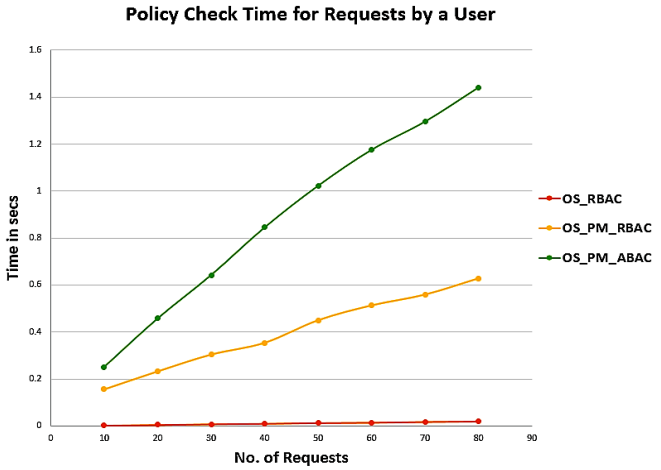

Figure 12: Policy Check Time for Requests by User

use cases to depict applicability and benefits of our ABAC extension with user attributes. We also investigated the cost and feasibility of this model and its enforcement architecture. This is an initial attempt towards applying a combination of ABAC model and RBAC model in a widely used cloud computing platform—OpenStack.

We believe this work will facilitate the transition towards ABAC models and will open prospective avenues to apply ABAC in real world applications using the PM. There are many interesting capabilities of PM that can be explored as extensions to our model such as applying combination of different access control policies defined in PM, or incorporating deny relations and constraints in the policies. Attribute and Role hierarchy is one of the possible extensions that is supported in PM and can be incorporated in our model. Besides these, the future work includes applying above discussed performance enhancements to the enforcement framework.

ACKNOWLEDGMENT

The authors would like to thank NIST and the Policy Machine team, especially Dr. David Ferraiolo, Serban I. Gavrila, and Gopi Katwala for their continuous support and useful suggestions. This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481, and DoD ARL Grant W911NF-15-1-0518.

REFERENCES

[1] B. Tang and R. Sandhu, "Extending OpenStack access control with domain trust," in *International Conference on Network and System Security*. Springer, 2014, pp. 54–69.

[2] R. Sandhu, E. J. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[3] R. Sandhu, "Role-based access control," *Advances in Computers*, vol. 46, pp. 237–286, 1998.

[4] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[5] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[6] "Amazon Web Services (AWS) - Cloud Computing Services." [Online]. Available: https://aws.amazon.com

[7] "Microsoft Azure." [Online]. Available: https://azure.microsoft.com

[8] L. Fuchs, G. Pernul, and R. Sandhu, "Roles in information security–a survey and classification of the research area," *Computers & Security*, vol. 30, no. 8, pp. 748–769, 2011.

[9] Q. M. Rajpoot, C. D. Jensen, and R. Krishnan, "Integrating attributes into role-based access control," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2015, pp. 242–249.

[10] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control." *IEEE Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[11] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (ABAC) definition and considerations," *NIST Special Publication 800-162*, 2014.

[12] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2012, pp. 41–55.

[13] M. A. Al-Kahtani and R. Sandhu, "A model for attribute-based user-role assignment," in *Proceedings of 18th Annual Computer Security Applications Conference, 2002.* IEEE, 2002, pp. 353–362.

[14] D. R. Kuhn, E. J. Coyne, and T. R. Weil, "Adding attributes to role-based access control," *IEEE Computer*, vol. 43, no. 6, pp. 79–81, 2010.

[15] X. Jin, R. Sandhu, and R. Krishnan, "RABAC: role-centric attribute-based access control," in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer, 2012, pp. 84–96.

[16] D. Ferraiolo, V. Atluri, and S. Gavrila, "The Policy Machine: A novel architecture and framework for access control policy specification and enforcement," *J. of Sys. Architecture*, vol. 57, no. 4, pp. 412–424, 2011.

[17] D. Ferraiolo, S. Gavrila, and W. Jansen, "Policy machine: features, architecture, and specification," *National Institute of Standards and Technology Internal Report 7987*, 2014.

[18] D. W. Chadwick, K. Siu, C. Lee, Y. Fouillat, and D. Germonville, "Adding federated identity management to OpenStack," *Journal of Grid Computing*, vol. 12, no. 1, pp. 3–27, 2014.

[19] C. A. Lee and N. Desai, "Approaches for virtual organization support in OpenStack," in *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2014, pp. 432–438.

[20] N. Pustchi and R. Sandhu, "MT-ABAC: A multi-tenant attribute-based access control model with tenant trust," in *International Conference on Network and System Security*. Springer, 2015, pp. 206–220.

[21] X. Jin, R. Krishnan, and R. Sandhu, "Role and attribute based collaborative administration of intra-tenant cloud IaaS," in *IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2014, pp. 261–274.

[22] "XACML." [Online]. Available: https://en.wikipedia.org/wiki/XACML

[23] "Policy Machine." [Online]. Available: http://csrc.nist.gov/pm/

[24] Y. Zhang, F. Patwa, R. Sandhu, and B. Tang, "Hierarchical secure information and resource sharing in OpenStack community cloud," in *IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 2015, pp. 419–426.