Adopting Provenance-Based Access Control in OpenStack Cloud IaaS

Dang Nguyen, Jaehong Park, and Ravi Sandhu

Institute for Cyber Security, University of Texas at San Antonio
`ytc141@my.utsa.edu, {jae.park, ravi.sandhu}@utsa.edu`

Abstract. **Provenance-based Access Control** (PBAC) has recently risen as an effective access control approach that can utilize readily provided history information of underlying systems to enhance various aspects of access control in a computing environment. The adoption of PBAC capabilities to the authorization engine of a multi-tenant cloud Infrastructure-as-a-Service (IaaS) such as OpenStack can enhance the access control capabilities of cloud systems. Toward this purpose, we introduce tenant-awareness to the $PBAC_C$ [14] model by capturing tenant as contextual information in the attribute provenance data. Built on this model, we present a cloud service architecture that provides PBAC authorization service and management. We discuss in depth the variations of PBAC authorization deployment architecture within the OpenStack platform and implement a proof-of-concept prototype. We analyze the initial experimental results and discuss approaches for potential improvements.

1 Introduction

Digital provenance data captures history information of system events. The utilization of provenance in computing platforms has demonstrated many benefits in different computing fields [4, 6, 7, 18]. In computer security, the utilization of provenance information facilitates the achievement of many security goals including intrusion detection and insider-threats detection. Harnessing provenance data to provide enhanced access control in different systems and platforms has been the basis of many recent works [2, 3, 8, 17]. The **Provenance-based Access Control** (PBAC) approach, as outlined in [12, 14, 15], captures application-specific provenance data and uses the information to enable enhanced access control in the underlying application system.

PBAC can effectively be employed in a multi-tenant[1] Infrastructure-as-a-Service (IaaS) cloud environment. Here, users (e.g., Virtual Machine

[1] We define a tenant from the perspective of a Cloud Service Provider (CSP), as an independent customer of the CSP responsible for paying for services used by that tenant. Payment is the norm in a public cloud while in a community cloud there often will be other methods for a tenant to obtain services. From the perspective of

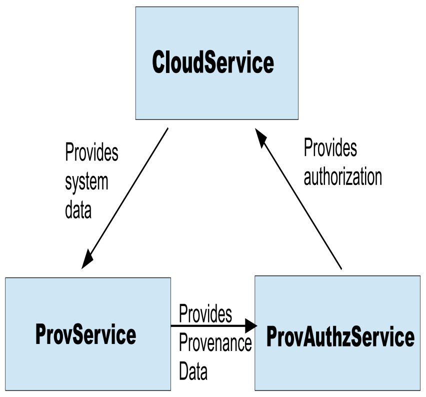

Fig. 1. A provenance-aware cloud architecture overview

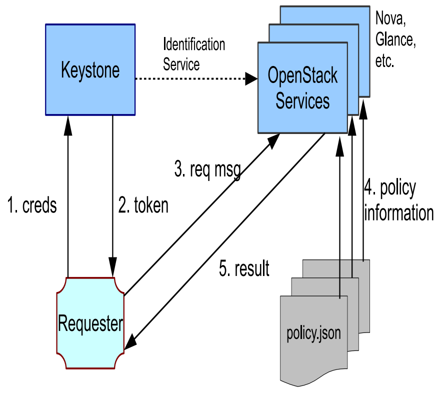

Fig. 2. An Overview of OpenStack Authorization

(VM) creators) and data objects (e.g., VM images, VM snapshots, VM instances) are involved in multiple tenants that are being configured with different authorization settings. The utilization of PBAC within a multi-tenant environment under multiple controlling principals can serve to elevate the authorization capabilities of cloud IaaS infrastructures, including but not limited to secure information flow control and prevention of privileges abuse. For example, a VM resource can be created in one tenant, shared and potentially modified in another tenant and then saved as an image for later use. Tenant administrators can specify access control on the shared VM resource based on its provenance data capturing its pedigree in the original tenant. In order to achieve these authorization goals with PBAC, it is essential to enable tenant-awareness in PBAC, a topic we discuss in this paper.

In this work, we focus our study in the IaaS layer of the cloud computing paradigm. Our contributions include a centralized-service architecture that enables provenance-awareness as well as cross-tenant utilization of provenance data for authorization. We proceed to describe the components, and their interactions, of the three service types depicted in Figure 1. We also identify a variety of architecture approaches the services can be deployed in the context of a cloud environment with and without cross-service provenance data sharing. We include several design criteria that can impact the choice of deployment approaches.

the tenant, a tenant could be a private individual, an organization big or small, a department within a larger organization, an ad hoc collaboration, and so on. This aspect of a tenant is typically not visible to the CSP in a public cloud.

2 Preliminaries

Cloud computing paradigm has recently risen as a popular approach that allows efficient utilization of computing resources that can simultaneously minimize related costs and achieve massive scalability at the same time. The concept has real-world practicality and development efforts are heavily invested by both academic and industrial sectors [5]. One of the important properties of cloud computing is multi-tenancy [10], where resources within a physical system are allocated and divided between tenants. The notion of tenants allows organized and secure administration of resources and management of privileged users/consumers of the resources.

In IaaS platforms, we focus on a multi-tenant single-cloud like a private cloud rather than a multi-cloud environment that is depicted in a hybrid cloud model [10]. In single-cloud environment, all services and associated resources within a cloud are provided by a single provider. Essentially, from the perspective of the tenants, there is a central provenance-aware authorization service that can normalize access control configurations and resolve conflicts across tenants. We adopt this setting in the incorporation of PBAC into cloud IaaS platforms as it naturally allows PBAC to be deployed without much cross-tenant complications.

In multi-cloud environment, individual cloud provider may permit controlled resources movement across cloud boundaries. In this case, it is a little bit more complicated to deploy PBAC as some forms of provenance data sharing are required. In our previous work [13], we proposed several approaches in addressing these concerns through the use of sticky provenance data and cascading sub-queries across tenants. In this paper, we do not explore these issues in depth.

3 Tenant-aware Provenance-based access control

We start with an overview of Provenance-based Access Control (PBAC) models as presented in [12, 14, 15].

Base PBAC Model ($PBAC_B$): The base PBAC Model ($PBAC_B$) [15] is an access control model which bases the authorization decision on provenance data. In order to effectively utilize provenance information, the $PBAC_B$ model makes use of a specific provenance data model which captures provenance data in directed-acyclic graph format [11]. Such capturing format allows effective ways to extract information through graph traversal queries.

Contextual PBAC Model ($PBAC_C$): The mechanism provided by the $PBAC_B$ model lays a solid foundation for access control that utilizes provenance information. An extended model, $PBAC_C$ [14], further

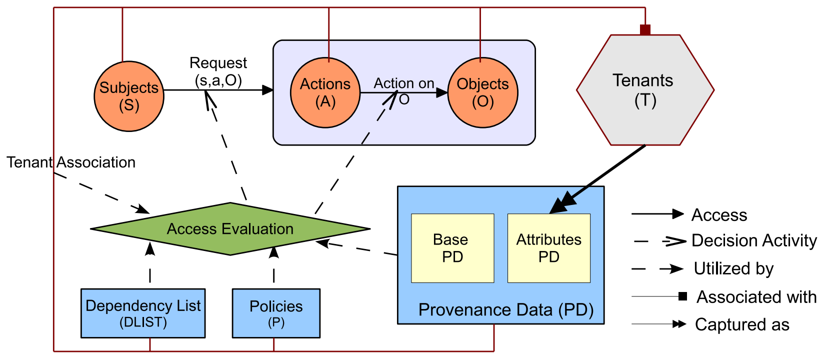

Fig. 3. A Tenant-aware $PBAC_C$ Model

enhances $PBAC_B$. Specifically, the extended model can capture and utilize contextual information associated with the primary entities of system events as attributes and store these as additional provenance data.

As depicted in Figure 3, the primary components of $PBAC_C$ can be briefly described as follows. **Subjects** represent human users interacting with a system. **Actions** represent the type of possible interaction a subject can perform in the system. **Objects** (or **Resources**) represent the type of data entities that exist within a system that require authorization protection for security goals. To interact with a system, a human user, through associated subjects, initiates **Requests** that will be evaluated based on **Policies** to determine the access decision (granted or denied). **Provenance Data** contains information on past system events as results of granted access requests and includes two types.[2] *Base* provenance data captures primary component-information of granted and executed access requests while *Attribute* provenance data captures the contextual information associated with the executed access requests.

In order to adopt tenant-awareness, we take the straightforward approach to view tenant as a special type of contextual information that can also be captured as attribute provenance data, as modeled in $PBAC_C$. We then use the relation type "associated with" to capture the semantic relations between tenants and other components such as: *Subjects*, *Actions*, *Objects*, *Dependency Lists*, *Policies*, and *Provenance Data*. Essentially, a set of atomic, or "base", application-specifically defined causality dependency edges between provenance graph vertices can be expressed with regular-expression based patterns. The graph vertices represent model components that constitute the primary entities of a system such as users or resources. This approach allows more expressive capture of relations between the model components. Meaningful combinations of dependency

[2] While provenance data can capture access requests that are not granted, for simplicity, we assume only granted accesses are stored in provenance data.

path expressions can be captured with abstract dependency-name constructs which represent more abstract application-specific semantics of the underlying system. An example is a dependency name "wasOriginallyUploadedBy" which captures any combination of dependency path expressions of actions, which can be multiple instances of modify or copy and ultimately upload, on a particular virtual image. Further application of attribute edges on the upload action instance can reveal the cloud user who originally uploaded the virtual image. These constructs can also be used for PBAC policy specifications. When an access request is generated, the *access evaluation* module extracts the request information to locate the appropriate policies for evaluation. When an access request is granted, the current contextual information is stored as provenance data. This contextual information is uniquely anchored to the action instance of the access transaction in provenance data.

4 Provenance-aware Access Control Cloud Architecture

In this section, we discuss a provenance-aware architecture that can enable PBAC capabilities in cloud environment. Specifically, we describe the main components and their interactions, and how the services can be deployed given various design criteria.

4.1 Architecture Overview

An overview of the architecture of our approach is depicted in Figure 1. We identify the three major types of services within this architecture as follows:

Cloud Service (CS) essentially provides a particular IaaS service to client tenants. The types of services include computing (management of virtual resources), authorization, virtual networks, and so on. Examples of the services can be *Amazon Web Services Elastic Compute*, OpenStack *Nova*, etc. These services essentially provide the functionality of the cloud.

Provenance Services (PS) is an IaaS service we propose with the purpose of capturing and managing provenance data that can be generated from any other typical cloud service. The provenance data captures the history information of system events occurring within the cloud services and can be utilized for many purposes. Our focused usage is on PBAC.

PBAC-enabled Authorization Services (PBAS) is an IaaS service we propose with the purpose of providing authorization capabilities to all other cloud services that require authorization. The authorization service is capable of providing PBAC features, but at the same time it can also provide other forms of access control including Role-based Access Control, Attribute-based Access Control, etc. Our focus in this paper is on the provision of PBAC capability for the authorization service.

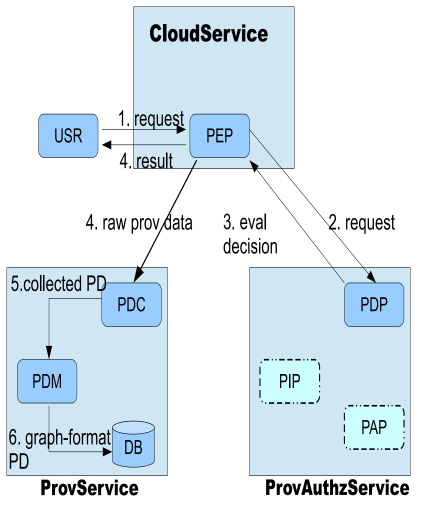

Fig. 4. A Provenance Service for Cloud IaaS

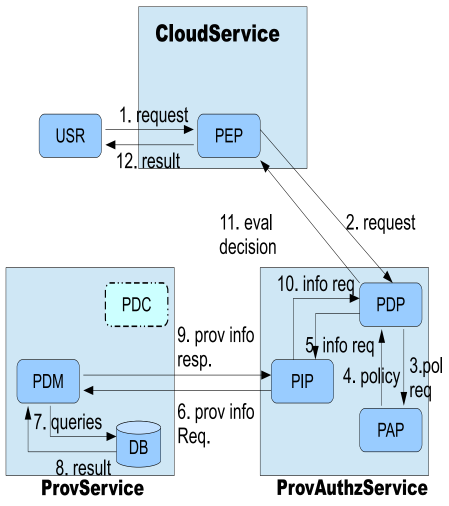

Fig. 5. A PBAC-enabled Authorization Service for Cloud IaaS

In summary, the three service types altogether establish an infrastructure that enables PBAC in a IaaS cloud. Specifically, the Cloud Service provides the PS with raw system events that PS selectively stores at provenance storage. The stored provenance data is then used to provide *PBAS* which enhances the security to the Cloud Service. While this work mainly focuses on the scenario where access requests are granted, we also note that it is possible to capture and store the information relating to access requests being denied. This information can allow additional control capability in a system. For example, if the provenance data of an object reflects that there exists three consecutive instances of request denial for a particular action type within certain recent request interval, it may lock the object from any future access or raise a flag indicating a potential threat or vulnerability within the system and request immediate attention with appropriate countermeasures. In this paper, we do not consider denied events as part of provenance data for simplicity and leave it for future study.

4.2 Conceptual Architecture

In this subsection, as shown in Figures 4 and 5 we identify and describe the interaction between the logical architectural components of the three service types. These components establish the fundamental and functional aspects of our architecture approach and can be applied to whichever deployment methods that are discussed later in the paper.

Components Any regular cloud service includes:

- Policy Enforcement Point (PEP): is responsible for receiving and enforcing an access request from a user. The enforcement is based on the evaluation results of that access request generated from the authorization service.
- A User is able to generate a request to the cloud service through any forms of interfaces such as web browsers (e.g., OpenStack Dashboard) or command-line interfaces (e.g., OpenStack Nova pythonclient).

The provenance service includes:

- Provenance Data Collector (PDC): is responsible for receiving raw system events data captured from a granted service action request being executed within individual services and potentially performing some filtering to select necessary data only.
- Provenance Data Manager (PDM): is responsible for transforming the collected raw data received from the PDC into provenance graph data format as well as managing the resulting provenance data. The management responsibilities include storing and loading provenance data in and from a database, as well as forming and executing provenance graph queries, and formatting and returning query results thereafter.
- Database (DB): represents persistent storage.

The PBAC-enabled authorization service includes:

- Policy Administration Point (PAP): is responsible for managing access control policies by enabling policies specification, storage and retrieval.
- Policy Information Point (PIP): is responsible for looking up relevant information that is necessary for making an access decision. In regard to PBAC, the PIP is tasked with delivering responses to provenance data requests to the relevant provenance service.
- Policy Decision Point (PDP): serves as the main computing process in deciding how a request should be resolved. In particular, the PDP receives the requests from the PEP, looks up the policy from the PAP, and requests information from the PIP to make decisions, which are then returned to the PEP.

Interactions We proceed to describe how the services perform whenever an access request comes in, as illustrated in Figure 4 and Figure 5. When a request is initiated by a user through any user client interface, the PEP receives the request and proceeds to verify the request with the authorization service by forwarding the request with relevant content to the PDP that resides in the PBAS service. In Figure 4, this interaction is abstracted in steps 2 and 3. It is further elaborated in Figure 5 through steps 2-11. Figure 4 demonstrates what happens after the access request evaluation process is completed. Essentially, if a request is granted, the PEP will enforce the execution of the requested action. The corresponding system event is then captured and sent to the PDC component of the provenance service where certain filtering can be performed to remove unnecessary data. The filtered data is then passed to the PDM for formatting into appropriate provenance data graphs for storage for later use. This completes a functional cycle in the context of an access request being directed at a cloud service.

In Figure 5, upon receiving the request from the PEP (2), the PDP proceeds to perform the evaluation procedure which includes, in sequential order, retrieving the correct policies through the PAP (3,4), searching for information required for policy rules evaluation through the PIP (5,6,9,10), and computing the actual evaluation decisions and returning the final results back to the PEP for enforcement (11). In our architecture, the PIP is responsible for looking up relevant provenance information in the provenance service for carrying out PBAC-related policy rules. The PIP performs this task by communicating with the PDM component of the provenance service by sending query templates. The PDM loads provenance data from its storage, forms appropriate queries and executes them to extract necessary provenance information to return to the PIP.

We proceed to demonstrate the above process with an example. Suppose a user Alice requests to delete a particular virtual machine, "vm1", in a tenant. The policy states that only a user who creates and stops a virtual machine instance can delete it. The PEP receives the request from Alice and delivers necessary information to the PDP. The PDP parses the request information, matches the request to the correct policy through the PAP to extract appropriate rules for the action "delete". The PIP is then sending information including "vm1" and dependency path patterns, e.g "wasVMCreatedBy", that express the semantics of creating and stopping users of a virtual machine instance to the PDM. The PDM forms appropriate queries using the provided information, executes the queries and returns the results back to the PDP. As Alice is shown to be the user who created and stopped "vm1", the PDP sends the approval to the PEP which starts the enforcement of the action. Upon completion, the PEP sends the events information to the PDC. The PDM can then at least generate provenance data which captures the fact that Alice performed "delete" on "vm1".

4.3 Deployment Architecture in OpenStack systems

Given the above logical architecture discussion, we shift the focus to how the services can be deployed in a cloud IaaS OpenStack system.

Most extant OpenStack cloud services often embed their own authorization service components that can enable authorization mechanisms including RBAC and ABAC. [3] In order to enable PBAC authorization mechanism for the extant OpenStack services, we identify several deployment architectures based on where PS and PBAS are implemented, which presents their own strengths and weaknesses.

First, similar to the current deployment of authorization components, these services can be integrated as structural components of an extant cloud service. In a cloud environment where sharing provenance data in multiple services is not a necessity, this integrated services deployment can significantly reduce the communication decision latency that takes place. Current standalone services, such as Nova and Glance, communicates over HTTP REST interface that can introduce expensive latency. Communication between components within the same service, as either inter-process or intra-process, is much less expensive in comparison. However, the extant cloud service will have more computing load to deal with as it will be required to maintain its own PBAS and PS components. Essentially, the integrated service has to collect, store and manage its own provenance data. This can also reduce the ease of services integration as it becomes more difficult to update changes to any of the embedded services.

Furthermore, in a cloud environment where cross-service provenance data sharing is necessary for purposes such as PBAC, a deployment of integrated PBAC-enabling services is required to employ provenance data sharing mechanisms. As each service stores and manages its own provenance data, PBAC decisions of a service require the provenance data of a different service. The requiring service has to initiate a request to the different service, therefore introduces communication decision latency over HTTP channels. In order to mitigate decision latency, each service can take the approach of maintaining duplicate provenance data of all relevant services. However, this introduces the necessity to synchronize all provenance data storage, and results in synchronization latency over HTTP channels. In scenarios where immediate synchronization is vital to the correctness of a PBAC decision, synchronization latency will affect decision runtime even if the evaluation process is done locally to the extant service. In scenarios where periodic synchronization is acceptable, optimal decision latency can be achieved.

Individual cloud service management of locally maintained provenance data can be complicated. The complications can be alleviated with the standalone deployment method with the cost of communication latency. Essentially, a standalone provenance service enables central prove-

[3] The Swift component utilizes a different form of authorization than most other OpenStack services.

nance data storage and management, which facilitates duplication and synchronization.

In addition, several variety of these two deployment methods, which we term hybrid deployment, can be employed to alleviate some of the issues faced by the above two deployment approach. For example, since not all provenance data is required for PBAC uses, only PBAC-relevant provenance data is necessarily duplicated in individual regular cloud services. Other provenance data, which can be used for auditing, can be stored and managed by standalone provenance service. In this paper, we use the standalone architecture for our OpenStack implementation and experiments.

5 An OpenStack Implementation

In this section, we will focus on the application of our approach on the open-source cloud management platform of OpenStack.

5.1 Overview of OpenStack Authorization Architecture

At the IaaS layer, OpenStack comprises several components that provide services to enable a fully functional cloud platform. Each of these components controls access to specific resources through locally maintained JSON policy files. At the IaaS layer, the resources to be protected are composed of API functions and virtual resources such as virtual machine images and instances.

Figure 2 captures a simplified view of the authorization as similarly performed by most OpenStack components. While the solid arrows denote information flow, the fine-dashed arrow indicates the Keystone component is responsible for providing identity service to other OpenStack components. This also provides authorization-required information indirectly using a token. When an access request is made, (1) authentication credentials need to be submitted to Keystone for validation. Once validated, (2) Keystone returns a token which contains necessary authorization information such as roles. (3) The token is then included in the request that is sent to the specific service component. (4) Authorization information is extracted from the token and used in evaluating the rules specified in the policy file native to that service. (5) The final evaluation and/or enforcement result is then returned to the requester.

Policy rules can be specified as individual rules of each criteria or a combination of rules. For Grizzly release, OpenStack authorization engine supports two types of rules: RBAC [16] where decisions are based on role field, ABAC [8] where decisions are either based on the value of a specific field or the comparison of multiple fields' values.

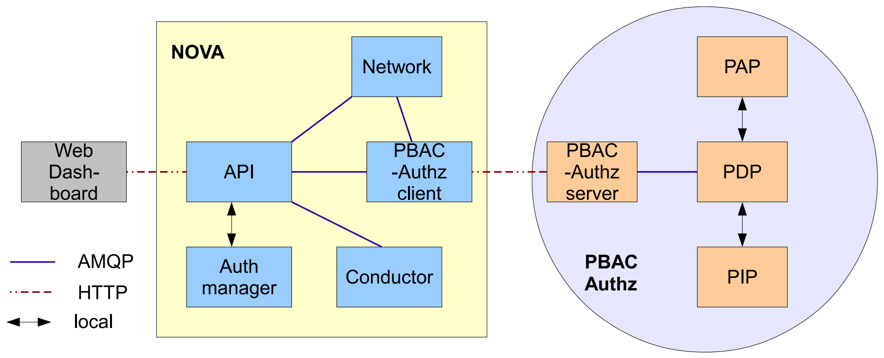

Fig. 6. Nova Implementation Architecture with PBAS Service

The authorization engine of OpenStack is evolving as additional blueprints and feature proposals are raised and delivered by the open-source community on a daily basis. Currently, OpenStack does not possess or support any variations of PBAC in its authorization schemes. As our demonstration and discussion of PBAC's usefulness in a multi-tenant cloud IaaS exhibit, it is useful to incorporate PBAC mechanisms into the OpenStack authorization platform for history-based, dynamic and finer-grained access controls.

5.2 Implementation and Evaluation

In this section, we describe and discuss our implementation of a proof-of-concept prototype that realize the above proposed architecture for enabling a PBAC-enabled authorization service within the OpenStack platform. Specifically, we will demonstrate how the OpenStack Computing (Nova) service can utilize the PBAC-enabled authorization for making access control decisions in addition to the current authorization schemes Nova is employing. [4] A similar process can be applied on the other services in OpenStack. In this paper, we implement our solution for Nova and Glance components and evaluate the performance runtime for each component under different experiments. Afterward, we provide an analysis of the runtime results.

OpenStack Nova Architecture First, we describe the current implementation approaches in the OpenStack Nova architecture.[5] As depicted

[4] An implementation of the provenance service is also available. However, since the emphasis is more on the PBAC aspect, we do not discuss the details of this component and the associated evaluation. Henceforth, Figure 6 does not depict the PS components.

[5] Similar architecture applies to Glance, the VM image repository in OpenStack.

in Figure 6, the Nova components include:[6] *Web Dashboard* is the potential external component that talks to the API, which is the component that receives HTTP requests, converts commands and communicates with other components via the queue or HTTP. *Auth Manager* is a python class component that is responsible for users, projects and roles. It is used by most components in the system for authentication purposes. *Network* is responsible for the virtual networking resources. *Conductor* is responsible for manage database operations.

There are two methods of communication between the service components: intra-service communication is done via AMQP mechanisms while inter-service communication is done via *HTTP REST* mechanisms. These are represented in Figure 6 as continuous lines and dashed lines respectively. Communication between sub-components of the same service can be done locally, such as invocation of the *Auth Manager*.

PBAC-enabled authorization implementation In order to incorporate and enforce PBAC-enabled authorization service, the following components were implemented to extend Nova.

– *PBAC Authorization Client*: a Python class that implements an authorization method that can be invoked whenever an API/ Scheduler/ Network/ Compute method is invoked. The client sends HTTP requests to the PBAC authorization server.
– *PBAC Authorization Server*: a Python class that resides on the PBAC-enabled authorization service. Receives HTTP requests from the PBAC authorization client, forwards the requests to and receives the decisions from the PDP, and returns the decisions to the PBAC authorization client.
– PDP, PIP, and PAP Python implementation for the corresponding architecture components.

Since all access control policies are specified in JSON, we implemented a policy parser class that can interpret policy statements specifying PBAC rules in JSON.
Evaluation and Discussion In order to evaluate our proof-of-concept prototype, we created and ran the provenance service and the PBAC-enabled authorization service in a Devstack installation of the OpenStack platform. [7] The Devstack is under the OpenStack Grizzly release and is deployed on a virtual machine that has 4GB of memory and runs on Ubuntu 12.04 OS installation. We generated mock provenance data that

[6] This is a partial list of Nova components.
[7] We note that in this paper we do not provide experiments for measuring provenance data update processes after granted action request enforced. Having such results can further enrich our insights and belongs in our future line of work.

simulates life cycles of VM images and instances across tenants and is stored in Resource Description Format (RDF)[9] with the Python RDFlib library [1]. We measured the execution performance of the following experiments.

Experiment 1 (e1): The execution time of a Glance command and a Nova command that require checking the associated policy for RBAC requirements (the original DevStack system).

Experiment 2 (e2): The execution time of the commands with the presence of an authorization service which evaluates the RBAC policy the service maintains and additionally PBAC policy where the authorization service also manage provenance service operations.

Experiment 3 (e3): The execution time of the commands with the independent presence of both a provenance service and a PBAC-enabled authorization service. The PBAC-enabled authorization service performs both normal RBAC requirements as well as PBAC requirements, where necessary provenance information is obtained from the provenance service.

For each experiment and each command, we performed 10 runs and took the average run-time. As noted in [14], the size and shape of the underlying provenance graph pose significant impact on query run-time. In this work we evaluated PBAC queries that require depth traversal. Specifically we chose to test the two scenarios where graph traversal takes distances of 20 and 1000 edges. These edge parameters were selected to respectively reflect a normal and an extreme use case of a VM image and instance within a cloud IaaS environment. The mock provenance data captures a VM image that is uploaded and modified multiple times and used to create a VM instance, which is suspended, resumed, and taken as a snapshot by multiple cloud users. The policy is specified following the informal grammar provided in [15]. A sample policy rule can specify that a user is only allowed to resume a VM instance if and only if he suspended that instance or a user is only allowed take a snapshot of a VM instance if he uploaded the VM image that instance is created from. The performance results are given in Table 1.

Based on the shown results, we make the following observations. First, compared to the regular execution (**e1** approach) of Glance or Nova commands, the incorporation of PBAC services (either **e2** or **e3** approaches) introduces some overhead for traversal distance of 20 edges, specifically between the 10 to 40 percent range. We also observe that the deploy-

Traversal Distance	Glance(e1)	Glance(e2)	Glance(e3)	Nova(e1)	Nova(e2)	Nova(e3)
No PBAC	0.55	-	-	0.75	-	-
20 Edges	-	0.607	0.642	-	0.902	1.062
1000 Edges	-	0.788	0.852	-	3.620	4.102

Table 1. Evaluation Runtime (secs)

ment of separate PBAC and Provenance services also introduces some overhead, specifically between the 5 and 18 percent range due to the additional communication time between provenance service and PBAC service. We observe that for the case of 1000 edges distance, the additional overhead is as expected as depth traversal require recursive implementation. However, the overhead is much more expensive for the Nova command in comparison to the Glance command. The reason for this lies in the authorization implementation of the Nova command. Specifically, the execution of the Nova command generates several authorization calls in contrast to only one from the Glance command. As the number of edges increases, this additional cost increases exponentially.

We propose a potential approach to improve on these performance results. Essentially, it is possible to reduce the performance cost associated with the increase in traversed edges by using meaningful, abstract edges that can equivalently capture the semantics of many base edges. This can help reduce the number of edges and thus produce, for example, a 20 edges run-time for a 1000 edges case.

6 Conclusions and Future Work

In this paper, we identified the potentials of adopting PBAC into the cloud Infrastructure-as-a-Service. To achieve practical deployment of PBAC, we proposed several variations of a centralized provenance and PBAC-enabled authorization services architecture. We demonstrated the architectural implementation in the context of the OpenStack cloud management platform. We implemented, evaluated and analyzed the performance runtime of our proof-of-concept prototype for the Nova and Glance components. From the results and our analysis, our work in this paper constitutes a strong foundation for enhanced authorization in cloud platforms. In order to further consolidate the validity of our approach, we are working on designing and performing more experiments. In addition, our prototype is still in preliminary stage and in need of additional development. We plan to release the prototype as an open source project in the near future.

Acknowledgement

This work is partially supported by NSF CNS-1111925.

References

1. https://code.google.com/p/rdflib/.
2. OASIS, Extensible access control markup language (XACML), v2.0 (2005).

3. A. Bates, B. Mood, M. Valafar, and K. Butler. Towards secure provenance-based access control in cloud environments. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY '13, pages 277–284, New York, NY, USA, 2013. ACM.
4. U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *The 3rd USENIX Workshop on Hot Topics in Security*, USENIX HotSec, pages 1–5, Berkeley, CA, USA, July 2008. USENIX Association.
5. M. Creeger. Cloud computing: An overview.
6. R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS '07, pages 13–18, New York, NY, USA, 2007. ACM.
7. R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *Trans. Storage*, 5(4):12:1–12:43, Dec. 2009.
8. X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy*, DBSec'12, pages 41–55, Berlin, Heidelberg, 2012. Springer-Verlag.
9. G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004.
10. P. Mell and T. Grance. The NIST definition of cloud computing. Special Publication 800-145, 2011.
11. L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche. The open provenance model core specification (v1.1). volume 27, pages 743 – 756, 2011.
12. D. Nguyen, J. Park, and R. Sandhu. Dependency path patterns as the foundation of access control in provenance-aware systems. In *4th USENIX Workshop on the Theory and Practice of Provenance*, TaPP'12. USENIX Association, Jun. 2012.
13. D. Nguyen, J. Park, and R. Sandhu. Integrated provenance data for access control in group-centric collaboration. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 255–262, 2012.
14. D. Nguyen, J. Park, and R. Sandhu. A provenance-based access control model for dynamic separation of duties. In *11th Annual Conference on Privacy, Security and Trust*, PST 2013. IEEE, Jul. 2013.
15. J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *10th Annual Conference on Privacy, Security and Trust*, PST 2012. IEEE, Jul. 2012.
16. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
17. L. Sun, J. Park, and R. Sandhu. Engineering access control policies for provenance-aware systems. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY '13, pages 285–292, New York, NY, USA, 2013. ACM.
18. V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, and L. Moreau. Security issues in a SOA-based provenance system. In *In Proceedings of the International Provenance and Annotation Workshop*. Springer, 2006.