SDAC: A New Software-Defined Access Control Paradigm for Cloud-Based Systems

Ruan He[1], Montida Pattaranantakul[2,3], Zonghua Zhang[2,3(✉)], and Thomas Duval[1]

[1] Orange labs, Châtillon, France
{ruan.he,thomas.duval}@orange.com
[2] IMT Lille Douai, Institut Mines-Télécom, Paris, France
{montida.pattaranantakul,zonghua.zhang}@imt-lille-douai.fr
[3] CNRS UMR 5157 SAMOVAR, Paris, France

Abstract. A cloud-based system usually runs in multiple geographically distributed datacenters, making the deployment of effective access control models extremely challenging. This paper presents a novel software-defined paradigm, called SDAC, to achieve scoped, flexible and dynamic access control. In particular, SDAC enables the tenant-specific generation of *access control model and policy* (*SMPolicy* in short), as well as their dynamic configuration by the cloud-hosting applications. To achieve that, SDAC uses an access control meta-model to initiate and customize different SMPolicies. Also, SDAC is decoupled into control plane and policy plane, allowing the global *SMPolicy* generated at the control plane to be efficiently propagated to the policy plane and enforced locally in different datacenters. As such, the local *SMPolicy* of a tenant can be synchronized with its global *SMPolicy* only when it's necessary, e.g., a user or a role cannot be identified. To validate the feasibility and effectiveness of SDAC, we implement a prototype in a carrier grade datacenter. The experimental results demonstrate that SDAC can achieve the desirable properties, maintain the throughput at a reasonable level regardless of the varying number of tenants, users, and datacenters, highly preserving scalability and adaptability.

1 Introduction

In the distributed cloud systems, one tenant can provision the resources from different cloud infrastructures. Considering the multi-tenancy, extremely dynamic and heterogeneous cloud environments, each tenant is expected to protect its users and resources with an effective access control model. However, the best practice of access control for cloud-based systems usually relies on the pre-definition of the access control models, e.g., Mandatory Access Control (MAC), Role Based Access Control (RBAC), while the tenant-specific and user-customized access control model remains unavailable.

S. Qing et al. (Eds.): ICICS 2017, LNCS 10631, pp. 570–581, 2018.
https://doi.org/10.1007/978-3-319-89500-0_49

Ideally, an access control paradigm for cloud-based systems should provide a mechanism for defining access control model on demand, as well as the dynamic specification of its policies at large scale. More importantly, the scope of an access control model and associated policies should be limited to an individual tenant instead of the whole system, while the tenant user is given the privilege to customize the most appropriate access control model and specify the corresponding policies. Then the policy of the tenant can be enforced at the multiple cloud infrastructures through a distributed access control framework.

To achieve these objectives, we propose a Software-Defined Access Control paradigm, called SDAC, which consists of an access control meta-model and a distributed framework: the meta-model is used for dynamically creating the access control model and policy (*SMPolicy*), while the distributed framework is used to enforce the policies in multiple datacenters. Thanks to the meta-model, SDAC is enabled with programmability, flexibility, and adaptability, allowing a tenant owner to define and customize his own access control model and specify the policies that are enforced inside the tenant. Also, the distributed framework makes SDAC scalable, enabling the access control policies to be effectively enforced at the tenant-level which is deployed across multiple datacenters.

To evaluate the feasibility and effectiveness of SDAC, we implement and deploy a prototype in a carrier grade cloud datacenter and conduct a set of experiments to validate its performance in terms of the desirable metrics, which is reported in Sect. 4. The design details of SDAC is presented in Sect. 3, covering the access control meta-model and the distributed framework. In Sect. 2, we briefly investigate the related work.

2 Related Work

Multi-tenancy is one of the salient features of cloud computing, which refers to the fact that the cloud infrastructure and its applications can be divided into the isolated sets of resources called tenants, according to different ownerships and requirements. As resources are distributed among several datacenters, their appropriate isolation and access control are key issues for both providers and users. However, as pointed out in [1,2], the conventional access control approaches are not suitable for the cloud, in which the resources are dynamically pooled and provisioned, the entities are heterogeneous, and the attributes are context-specific, resulting in sophisticated and fine-grained authorization requests.

It is well recognized that the access control policies needs to be created, configured or removed by the authorized users, so both DAC (Discretionary access control) and MAC allow to be specified the constrains on the assignment or the use of permissions [3]. For example, DAC allows the owner of an object to set up its access control list, while MAC supports the clearance and classification configuration by the administrator. However, these properties can not sufficiently meet all the requirements in the cloud, especially on the dynamic customization of access control models and the specification of policies. By leveraging the

software-defined approach, our SDAC provides a *generic* and *independent* policy customization approach that can *dynamically* create a specific access control model, together with its policy, for a group of objects.

The notion of *scope* in access control was firstly introduced in MAC as category [4], which refers to the validity of security lattice and provides a finer grained security classification. Also, OrBAC (Organisation-based access control) defines one security policy for one organization [5], while attribute-based access control encapsulates policy definition and decision-making algorithms into an independent unit [6]. Then in [7], the authors extend the notion of scope for multi-tenancy with RBAC and establishes trust relationship between the scopes. However, in the cloud, each tenant needs an access control policy to define the rules that authorize the users to access the appropriate resources. If we treat a tenant as a *scope*, a generic data model is expected to customize the access control model for handling the dynamically created scopes. Unfortunately, to the best of our knowledge, few of the available access control paradigms can achieve such an objective. For example, IBM [8] proposed Tivoli Access Manager to provide best practice of web-based access control solution for protecting tenant's cloud resources and supporting multi-tenant architectures. In [9,10], the authors presented distributed access control architectures for cloud computing, in which XML and XACML respectively, are used to formulate the access control policies. However, all of them have limited capability to generate different tenant-specific access control models and support distributed access control framework. In SDAC, we propose a *meta-model* to generate different access control models and policies, and present a four-layered framework to specify and enforce the tenant-specific access control policies in a *distributed* way, significantly improving the flexibility and dynamicity. In [16], a generic NFV based security management framework has been proposed, with an objective to orchestrating various security functions such as access control. However, the actual implementations have not been extensively discussed.

We have also seen tremendous efforts on designing access control schemes using cryptographic approaches [11], which mainly handle the access to the storage systems, while the protection scope of SDAC is all the computing resources (e.g., compute, storage, network) associated with the tenant. In [1], the authors pointed out that the user-role and role-permission assignments should be separately constructed using policies applied on the attributes of users, roles, the objects and the environment. Also, the attribute-based user-role and role-permission assignment rules should be applied in real-time in order to enforce access control decisions. Our SDAC successfully fulfills such requirements and provides a flexible way for the tenant owners to specify, configure, and manage access control model and policies on the fly according to the particular needs.

3 SDAC: Software-Defined Access Control Paradigm

In this section, we firstly identify the requirements on developing SDAC. We then specifically present the paradigm, which consists of a distributed framework and an access control meta-model.

3.1 Design Requirements

- **Tenant-specific and model generation on demand.** The protection scope should be limited to the tenants, so that the entities of the access control model (*e.g.*, subject, object, role, and related information) need to be specified based on particular tenant domain. As a tenant can be dynamically created or removed, the access control model and associated policies (Scoped Model and Policy (*SMPolicy*)) need to be generated or removed in real time.
- **Programmability of *SMPolicy*.** To facilitate the dynamic generation of SMPolicy, a meta-model needs to be developed through which each tenant can customize its access control model and policy set according to its specific security needs. As a tenant can be across multiple cloud infrastructures, the SMPolicy allows arbitrary update of any parts of the access control model instead of duplicating one complete access control model for each cloud.
- **Integration of diverse features.** The emerging access control models usually integrate diverse features to include more semantic information. For example, *session* in RBAC [12] can temporally activate or deactivate a user-role assignment, while the hierarchy role enables privilege inheritance from one role to another, and the *delegation* sets up a temporal trust relation between two users. UCON [13] provides continuous control through obligations and conditions, and it also enables attribute mutability as a supplementary instruction before or after an access decision [14].
- **Centralized control yet distributed enforcement.** To ensure the consistency and efficiency, a user should be able to create or configure a SMPolicy in a centralized way by describing all its datasets, while the policies need to be enforced in a distributed manner, covering all the datacenters that the tenant is deployed. More importantly, the policy updates should be delivered to the related enforcement components in the distributed infrastructure.

To meet the above requirements, we develop Software-Defined Access Control Paradigm (SDAC), which contains an access control meta-model and a distributed framework. In particular, the meta-model is used to initiate different access control models and policies (*SMPolicy*) for a scope, and the distributed framework is used to enforce the generated scoped policies. Specifically, the major operations of SDAC include: (1) creating a scope and identity entities involved in this scope; (2) generating an access control model; (3) specifying an access control policy based on the model, and; (4) enforcing the policies.

3.2 SDAC Framework

We propose SDAC framework to abstract and centralize policies into a control plane instead of distributing them to the local datacenters. As shown in Fig. 1, SDAC framework is composed of four planes, which are described as follows,

- **Application plane,** which hosts the applications that invoke PAP (Policy Administration Point) to customize *SMPolicy*. In [15], the feasibility of this access control framework has been demonstrated through a NFV orchestrator, which can be treated as an application.

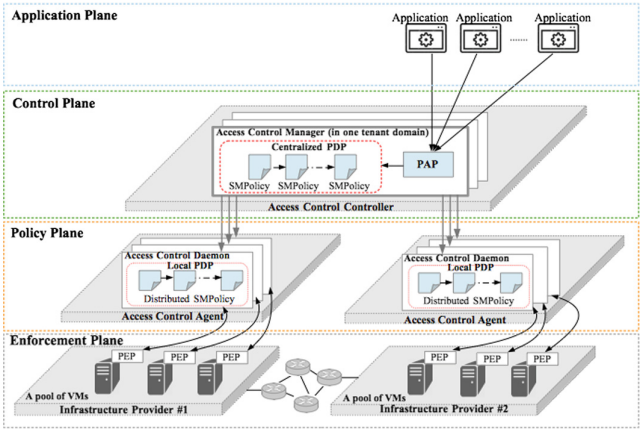

Fig. 1. Design architecture of SDAC

- **Controller plane:** an *Access Control Controller* containing multiple *Access Control Managers*, each of which is in charge of one tenant and has a global view about all the authorization related information and rules for that tenant. In another word, the *Access Control Manager* defines *SMPolicies* for each tenant. As in XACML [17], a PAP and a PDP (Policy Decision Point) are used for authorization.
- **Policy plane:** *Access Control Agents* are deployed in each local datacenter, serving as the agent of *Access Control Controller*. Each *Access Control Agent* holds a set of *Access Control Daemons*, which contain all the related information about the local usage of one tenant. The global *SMPolicies* are defined in the control plane, while the local *SMPolicies* are stored in the local PDP of each access control daemon. Thus, the users only need to customize the *SMPolicies* in the control plane, eventually leading to the automated customization of *SMPolicies* in the policy plane.
- **Enforcement plane:** the PEPs (Policy Enforcement Points) are installed into cloud-based systems for sending authorization requests to the corresponding local PDP running in the *Access Control Daemon* and finally enforces the decision from that distributed PDP.

3.3 Access Control Meta-model

The purpose of access control meta-model is to instantiate an access control model, *e.g.*, DAC, MAC, RBAC for a specific scope (one tenant in the cloud). To do that, we use attribute-based specification, which has potential to cover various access control models [18]. In particular, the attributes are those security-related properties such as role, domain, group, and type. Then a policy can define a set of rules based on the values of these attributes. To develop the meta-model, two notations are given as follows,

- Entity $E = \{e_i\}$, which can be used as *subjects* or *objects* in the access control model, and they can be either users or cloud resources.
- Information I: the related security properties of each entity, *e.g.*, user roles, file types. One entity can be assigned with several types of properties, called categories, and each of which has a set of values, *e.g.*, $I =$ *InfoCategory* $\times$ *CatScope*, where *InfoCategory* is a set of the types of security properties, and *CatScope* is a set of potential values for each category.

We suppose that each *SMPolicy* (Meta-model based Access Control Model and Policy) specified by the SDAC meta-model consists of an access control model (ACM) and an access control policy (ACP), which are described as follows.

Access Control Model: $ACM = (MD, MR)$, which includes a meta-data (MD) and a meta-rule (MR). In particular, MD defines a schema to instantiate an access control model, i.e., $MD = (SubjectMD, ObjectMD, ActionMD)$, where $SubjectMD, ObjectMD, ActionMD \subseteq InfoCategory$. The MR defines the schema to create the rules, which involve information categories based on which an authorization-related instruction can be executed, i.e.,

$SubjectCategory \times ObjectCategory \times ActionCategory \rightarrow Instruction$; where

$SubjectCategory \subseteq SubjectMD$, $ObjectCategory \subseteq ObjectMD$, $Action\ Category \subseteq ActionMD$, $Instruction : \{AuthzDecision, PolicyUpdate, Policy\ Chain\}$

For instance, the rule can be a decision to grant or deny an authorization request, update the policy itself, or redirect to another policy in the policy chaining. Thus, by extending the conventional access control policy with generic instructions, the meta-model is able to instantiate different access control models and advanced control features, and integrate them within one policy chain.

Access Control Policy: $ACP = (D, R, P, EDAss)$, which creates an access control policy for the access control model that is applied to a particular scope *e.g.*, one tenant in the cloud. The policy specifies potential values called data (D) for each category, establishes rules (R), identifies perimeter (P) and assigns values to each entities ($EDAss$) of this policy. Specifically,

- Data (D) is a complete set of values for each category for subjects, objects, actions, i.e., $D = (SubjectD, ObjectD, ActionD, Instruction)$, where $SubjectD \subseteq SubjectMD \times CatScope$, $ObjectD \subseteq ObjectMD \times CatScope$, $ActionD \subseteq ActionMD \times CatScope$
- Rule (R) specifies user privileges by using category values of subjects, objects, and actions to determine the instructions to be triggered, i.e., $R : SubjectD \times ObjectD \times ActionD \rightarrow Instruction$. As aforesaid, three types of instruction are identified, (1) authorization decision (grant or deny); (2) policy update to modify the category values of an entity; (3) policy chain to route the request to another policy.
- Perimeter (P) is a set of entities (subjects, objects, actions) to be protected. As each SMPolicy is applied to one particular scope, we need to define its

perimeter by identifying the entities that are involved in this scope, i.e., $P = (S, O, A)$, where $S, O, A \subseteq E$.
- Entity-Data Assignment ($EDAss$) is used to establish many-to-many relationship between related data and entities by assigning category value to each entity. Formally, $EDAss = (SubjectDataAss, ObjectDataAss, ActionDataAss)$, where $SubjectDataAss \subseteq S \times SubjectD$, $ObjectDataAss \subseteq O \times ObjectD$ $ActionDataAss \subseteq A \times ActionD$.

Policy Decision Algorithm, which is based on the values of categories. When an access request arrives, the algorithm fetches category values of subject, object and action to interpret the request through the value assignment of each entity. Then the algorithm checks whether these values match some rules in the policy. That says, access request (s_i, o_j, a_k) may trigger an instruction $inst_l$ if:

$$(\{s_i\} \times SubjectDataAss, \{o_j\} \times ObjectDataAss, \{a_k\} \times ActionDataAss, inst_l) \subseteq R$$

where $\{e\} \times DataAss$ means fetching all attributes of the entity e from its entity-data assignment.

3.4 SMPolicy Programmability

To generate an access control model, we use object-oriented approach. In particular, in access control model, an *object* is usually the resource that can be manipulated by a *subject*. If we model the *subjects*, *objects*, *attributes*, and the *relations* between them as *objects*, the authorized users then can manipulate them through a customization interface, *e.g.*, creating, modifying, and removing any parts of an access control model. As relations are also treated as *objects* that can be manipulated, users can modify attribute assignment as well. As shown in Fig. 2, SDAC meta-model enables dynamic creation of access control model and policy through a policy customization interface, which allow any SMPolicy defined by meta-model to be programmed and configured.

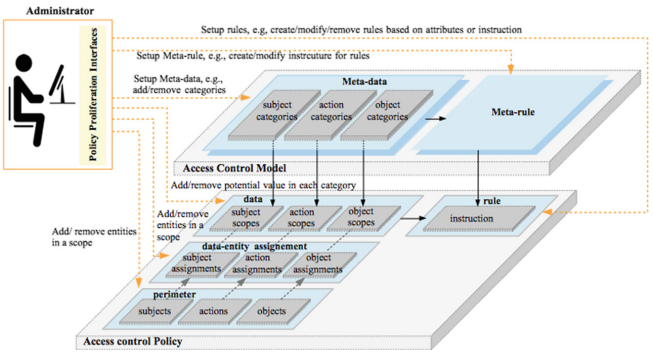

Fig. 2. SDAC data model: key components and customization

For a particular tenant, a corresponding *SMPolicy* is created as follows,

1. Through the meta-data interface, the admin creates categories for *subjects, objects* and *actions*, then defines *meta-rules*, which specify categories to be used to build rules. The meta-data (MD), together with meta-rule (MR), constructs a customized access control model.
2. The admin creates an access control policy based on this access control model by specifying the values for each category, and creating rules (*instructions*) based on these values and the *meta-rule*;
3. The admin identifies *subjects*, *objects* and *actions* that need to be protected by this *SMPolicy*, and finally assigns values to each category *subjects*, *objects* and *actions*.

To illustrate the creation of *SMPolicy*, an implementation of MLS (Multi-Level Security) with *SMPolicy* is given here. In particular, MLS sets up security levels for subjects and objects, authorization is then granted through their comparison. By applying SDAC meta-model, we can define the *SMPolicy* of MLS as follows,

- E = $\{user_0, user_1, user_2, vm_0, vm_1$, start-vm, stop-vm$\}$
- InfoCategory = (subject-security-level, object-security-level, action-type)
- CatScope = ((subject-security-level, [low, medium, high]), (object-security-level, [low, medium, high]), (action-type, [vm-action, storage-action]))

Meta-data MD:

- SubjectMD = (subject-security-level)
- ObjectMD = (object-security-level)
- ActionMD = (action-type)

Meta-rule MR:

- SubjectCategory = (subject-security-level)
- ObjectCategory = (object-security-level)
- ActionCategory = (action-type)
- Instruction = (AuthzDecision)

Data D:

- SubjectD = (subject-security-level, [low, medium, high])
- ObjectD = (object-security-level, [low, medium, high])
- ActionD = (action-type, [vm-action, storage-action])

Rule R:

- r = ((subject-security-level, [high]), (object-security-level, [medium]), (action-type, [vm-action], (instruction, [grant]))
- r = ((subject-security-level, [high, medium]), (object-security-level, [low]), (action-type, [vm-action], (instruction, [grant]))

Perimeter P:

- S: $\{user_0, user_1\}$
- O: $\{vm_0, vm_1\}$
- A: {start-vm, stop-vm}

Entity-Data Assignment $EDAss$:

- SubjectDataAss = (($user_0$, high), ($user_1$, medium))
- ObjectDataAss = ((vm_0, medium), (vm_1, low))
- ActionDataAss = ((start-vm, vm-action), (stop-vm, vm-action))

In this MLS, a user can start or stop a VM if and only if his or her security level is higher than that of VM. For example, $user_0$ can manipulate vm_0 and vm_1, while $user_1$ can only manipulate vm_1.

3.5 SMPolicy Chaining

The basic idea of policy chaining is to combine and route several *SMPolicies* together. For example, the authors of [6] have applied this idea to attribute-based access control for grid computing. For the cloud-based systems, it is well recognized that developing a generic access control model meeting the diverse requirements of all the tenants is mission impossible. We therefore propose to chain several feature-specific *SMPolicies* together rather than develop a generic one for implementing the integrated access control semantics. In doing so, an existing sophisticated access control policy with advanced features can be decomposed into a set of atomic SMPolicies, each of which can implement either a basic access control model or a particular advanced feature, e.g., session, delegation.

Formally, we define policy chain as a set of ordered SMPolicies, each of which is atomic that contains all the dataset about its model and policy. By introducing the concept of *instruction* in the SDAC meta-model, we extend an access control model to be more sophisticated to specify advanced control features, such as session in RBAC or continuous control in UCON. That says, a *SMPolicy* can be used to realize either a basic access control policy or an advanced feature. When a request occurs, the *SMPolicy* firstly fetches all the information related to this request. Based on their category values, it then decides whether to launch one or several instructions. For a basic access control policy, an access control policy makes a decision for triple-request (subject, object and action), where a subject (user) intends to take an action on an object. The resulting instruction of this *SMPolicy* is an authorization decision like grand or deny, based on the available information. If a *SMPolicy* specifies a control feature like session, obligation, the implication of triple-request (subject, object, action) refers to the fact that modifying (action) an attribute (object) of an entity (subject). The resulting *instruction* is then to update the *SMPolicy*. Similarly, if the *instruction* involves forwarding the request to another *SMPolicy*, then the triple-request (subject, object, action) means sending (action) the request (subject) to another *SMPolicy* (object). The three fields of an *instruction* allow to modify the *SMPolicy* and its dataset, route the request and/or dataset to another *SMPolicy* and finally validate the request.

4 Experiments

Our SDAC prototype is deployed into three HA (High-Availability) OpenStack clusters, one serves as master platform, while another two run as slave platforms. Each one is equipped with 5 servers (Intel E5-2680 with 48 cores/251G RAM), of which 3 are controller nodes and 2 are compute nodes. The 6th server is set up as a security node running SDAC. Specifically, our SDAC is implemented based on a micro-service architecture, which means that both *Access Control Manager* in the control plane and *Access Control Daemon* in the policy plane are implemented through a set of containers.

Throughput of the Policy Engine. One of the key metrics for evaluating the capability of access control policy engine (PDP) is *throughput*, *e.g.*, the number of authorization requests per second that it can handle. In this evaluation, we set *SMPolicy* as a basic *RBAC*, which has 10 users, 5 roles and 10 objects. We gradually increased the number of requests to observe the throughput of the policy engine. As shown in Fig. 3, the average throughput arrives its limit (4.1 requests per second) when the request frequency was adjusted from 1 to 20 requests per second. It is worth nothing that, thanks to the micro-service architecture, one identical *SMPolicy* container will be automatically launched when the number of request is beyond the throughput of the policy engine.

SMPolicy Chaining Overhead. Our SDAC allows several *SMPolicies* to be chained together to meet specific policy requirements. This apparently will incur certain overhead. In this experiment, we configure one tenant (10 users, 5 roles and 10 objects) with a purpose to comparing the authorization overhead between (1) $RBAC_0$ implemented by one policy only; and (2) $RBAC_0$ that is realized by chaining 2 SMPolicies together. The results is shown in Fig. 4. In case (1), the throughput was around 4 requests per second, while in case (2), the throughput was 2.9 requests per. It can be concluded that the extra overhead introduced by policy chaining is around 32%.

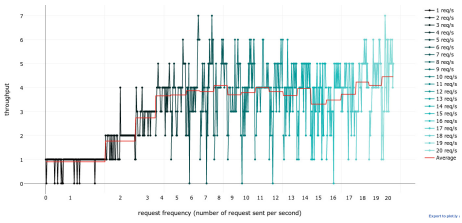

Fig. 3. Throughput: # of requests per second

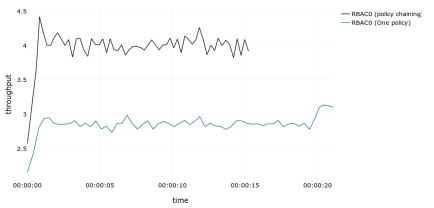

Fig. 4. SMPolicy chaining overhead

Scalability with the increasing number of users. We set *SMPolicy* as a basic *RBAC* for one tenant, which has 5 predefined roles and 10 VMs (as objects). We then increased the number of users and observed that the throughput remained stable as 5.9 requests per second when there were 50 users. Then the throughput

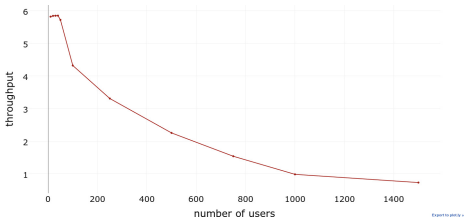

Fig. 5. # of users varying from 10 to 1500

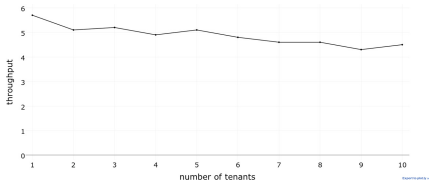

Fig. 6. # of tenants varying from 1 to 10

decreased dramatically when the number of users got larger than 50, as shown in Fig. 5. The worst case was 0.5 requests per second when the number of users reached to 1500.

Scalability with the varying number of tenants. To evaluate its scalability, we configured each tenant which has 10 users, 5 predefined roles and 10 VM objects. As shown in Fig. 6, the throughput of policy engine varied from 5.7 requests per second (one tenant) to 4.5 requests per second (10 tenants), showing slight degradation. The reason is that SDAC is implemented using the micro-service architecture, in which *SMPolicies* of each tenant run in a dedicated and independent containers, enabling SDAC to scale freely with multiple tenants.

5 Conclusion

This paper proposed a software-defined access control paradigm called SDAC for cloud-based systems, which requires the access control to be dynamic, adaptive, fully distributed and easily managed. Specifically, SDAC is featured with a distributed framework and a meta model. The distributed framework allows the access control models and policies to be distributed to the multiple tenants in different could datacenters, while they can be managed and updated in a centralized way. The meta-model provides a generic customization interface to generate different access control models on demand. The meta-model also extends conventional access control to be more sophisticated by integrating instruction, through which an access control policy can implement other operations in addition to grant and deny. More interestingly, some advanced features like session and delegation can be enabled and integrated through the policy chaining mechanisms in PDP. It is worth mentioning, however, that the dynamic access control model and policy customization can potentially introduce novel security vulnerabilities, and the policy chaining may cause conflicts. Thus, our future work will be focused on validating the correctness of access control model generation and chaining. Trust relationship between different tenants in the cloud will be also considered in the policy chaining.

References

1. Meghanathan, N.: Review of access control models for cloud computing. Comput. Sci. Inf. Technol. **3**, 77–85 (2013)
2. Ngo, C., Demchemko, Y., de Laat, C.: Multi-tenant attribute-based access control for cloud infrastructure services. J. Inf. Secur. Appl. **27**, 65–84 (2016)
3. Sandhu, R.S., Samarati, P.: Access control: principle and practice. IEEE Commun. Mag. **32**(9), 40–48 (1994)
4. Sandhu, R.S.: Lattice-based access control models. Computer **26**(11), 9–19 (1993)
5. Kalam, A.A.E., Baida, R.E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: POLICY 2013, pp. 120–131 (2003)
6. Lang, B., Foster, I., Siebenlist, F., Ananthakrishnan, R., Freeman, T.: A flexible attribute based access control method for grid computing. J. Grid Comput. **7**, 169–180 (2009)
7. Calero, J.M., Edwards, N., Kirschnick, J., Wilcock, L., Wray, M.: Toward a multi-tenancy authorization system for cloud services. IEEE Secur. Priv. **8**(6), 48–55 (2010)
8. IBM: Best practices for access control in multi-tenant cloud solutions using Tivoli Access Manager, May 2011. https://www.ibm.com/developerworks/cloud/library/cl-cloudTAM/index.html
9. Almutairi, A.A., Sarfraz, M.I.: A distributed access control architecture for cloud computing. IEEE Softw. **29**(2), 36–44 (2012)
10. Decat, M., Lagaisse, B., Van Landuyt, D., Crispo, B., Joosen, W.: Federated authorization for software-as-a-service applications. In: Meersman, R., Panetto, H., Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., De Leenheer, P., Dou, D. (eds.) OTM 2013. LNCS, vol. 8185, pp. 342–359. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41030-7_25
11. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: IEEE INFOCOM 2010, pp. 1–9 (2010)
12. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. **4**(3), 224–274 (2001)
13. Park, J., Sandhu, R.: The UCONABC usage control model. ACM Trans. Inf. Syst. Secur. **7**(1), 128–174 (2004)
14. Park, J., Zhang, X., Sandhu, R.: Attribute mutability in usage control. In: Farkas, C., Samarati, P. (eds.) DBSec 2004. IIFIP, vol. 144, pp. 15–29. Springer, Boston, MA (2004). https://doi.org/10.1007/1-4020-8128-6_2
15. Pattaranantakul, M., Tseng, Y., He, R., Zhang, Z., Meddahi, A.: A first step towards security extension for NFV orchestrator. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 598–605 August 2016
16. Pattaranantakul, M., He, R., Meddahi, A., Zhang, Z.: SecMANO: towards network functions virtualization (NFV) based security management and orchestration. In: ACM International Workshop on SDN-NFVSec 2017, pp. 25–30, March 2017
17. XACML:3.0: eXtensible access control markup language (XACML) Version 3.0, OASIS Standard (2013). http://portal.etsi.org/NFV/NFV_White_Paper.pdf
18. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering DAC, MAC and RBAC. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 41–55. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31540-4_4