ABAC Based Online Collaborations in the Cloud

Mohamed Amine Madani[1(✉)], Mohammed Erradi[1], and Yahya Benkaouz[2]

[1] Networking and Distributed Systems Research Group, SIME Lab, ENSIAS, Mohammed V University in Rabat, Rabat, Morocco
amine.madani@um5s.net.ma , mohamed.erradi@gmail.com
[2] LCS, Department of Computer Science, FSR, Mohammed V University in Rabat, Rabat, Morocco
y.benkaouz@um5s.net.ma

Abstract. Nowadays sharing data among organizations plays an important role for their collaboration. During collaborations, the organizations need to access shared information while respecting the access control constraints. In addition, most organizations rely on cloud based solutions to store their data (e.g. openstack). In such platform, data access is regulated by Access Control Lists (ACLs). ACL defines static access rules. It assumes the knowledge of the whole set of users and possible access requests. This make ACL unusable in collaborative context due to the dynamic nature of collaborative sessions. In this paper, we consider ABAC, a flexible and fine-grained model, as an access control model for cloud-based collaborations to overcome the ACL limitations. We provide an architecture that integrate ABAC in the storage level of a cloud platform.

Keywords: ABAC model · Swift · Collaborative session · Access control

1 Introduction

Nowadays, sharing information among multiple organizations plays an important role to ensure an optimal utilization of distributed resources to improve productivity and profits. In order to reach this objective, a tight collaboration among organizations should be established. Collaborative applications allow a group of users to collaborate, communicate and cooperate through distributed platforms in order to perform common tasks, such as document sharing.

As most organizations rely on cloud-based solutions to store their data, cloud platforms [1] provide a considerable convenience to support the collaboration as well as the information sharing [8]. In this direction, OpenStack cloud platform represents a very interesting solution. OpenStack [6] is an open source IaaS (Infrastructure as a Service) software adopted by many cloud service and technology providers such as Rackspace, IBM, Dell and RedHat.

During collaborations, the organizations need to access and use the information shared by other collaborating organizations. This information often contains sensitive data. It is meant to be shared only during specific collaborative sessions

F. Belqasmi et al. (Eds.): AFRICATEK 2017, LNICST 206, pp. 67–76, 2018.
https://doi.org/10.1007/978-3-319-67837-5_7

[5]. This arises the access control issue [4]: The organizations need strong access control model to permit or deny a specific request of other organizations.

Using OpenStack cloud platform, the collaborating organizations store their data and information as objects in the Swift storage [7] (an object storage service in OpenStack). Swift uses the access control lists (ACL) to manage the access permissions. However, ACL model is too simple, static, and a coarse-grained model that does not provide the rich semantics for the collaboration. During a collaborative session, users may intervene dynamically without a prior knowledge of which user will access which objects. Specifying access rules during a collaboration is a difficult even an impossible task to accomplish using ACL. A fine-grained access control model is mandatory to support the requirements of the collaborative systems [5].

In this direction, Attribute Based Access Control model (ABAC) is of a great interest. ABAC model [9,10] overcomes the limitations of the classical access control models (i.e., ACL, MAC and RBAC). This model is adaptive and flexible. ABAC is more suitable to describe complex, fine-grained access control semantics, which is especially needed for collaborative environments. In ABAC, access requests are evaluated based on the user attributes, the object attributes and the environment attributes. Therefore, in this paper, our main contributions are twofold: (1) Ensuring the access control dynamicity in collaborative session on the cloud based on the ABAC model. (2) Providing an architecture to integrate ABAC in the storage level of the cloud and providing an enforcement model.

The paper is organized as follows: Sect. 2 presents the background of this work. In Sect. 3, we present the related work. Section 4 describes the suggested architecture and the enforcement model. Section 5 discusses the implementation performance. Finally, we conclude in Sect. 6.

2 Background

This section aims to present the necessary background of this work. This section mainly focus on the presentation of the concept of Cloud based collaborative application. Then, it gives an overview of the OpenStack cloud platform. Finally, it presents the attribute based access control model.

Cloud based collaborative applications. Collaborative applications are among the services that can be provided by the cloud computing. They enable collaboration among users from the same or different tenants of a given cloud provider [2,3]. During collaborations, the participants need to access and use resources held by other collaborating users. These resources often contain sensitive data. They are meant to be shared only during specific collaborative session [5]. The collaborative session is an abstract entity, comprising a set of users, called members of the session, playing the same or different roles. These users may have concurrent access to shared objects in this session depending on the access control policies. As most organizations rely on cloud-based solutions to store their data, cloud platforms provide a considerable convenience to support

the collaboration. In this direction, OpenStack cloud platform represents a very interesting solution.

Openstack. OpenStack is a robust open-source IaaS software for building public, private, community or hybrid clouds. OpenSteck is adopted by many cloud providers such as Rackspace, IBM and RedHat. OpenStack contains the following components: Nova, Swift, Glance, Cinder, Keystone, and Horizon. Each component acts as a service which communicates with other sevices via message queues. Keystone provides authentication and authorization for all OpenStack services. In our work, we focus on the Swift object storage. Swift is a multi-tenant, highly scalable and durable software defined storage system designed to store files, videos, virtual machine snapshots and other unstructured data [7]. It allows building, operating, monitoring, and managing distributed object storage systems that can scale up to millions of users.

The Account Server is responsible for listings of containers, while Container Server is responsible for listings of objects. A container is a mechanism that stores data objects. An account might have many containers, whereas a container name is unique. A user represents the entity that can perform actions on the object in the account. Each user has its own account and is associated to a single tenant. Swift uses the access control lists (ACL) to manage the access permissions. In fact, the ACL model defines static access rules. It is not suitable for collaborative environment. In this direction, Attribute Based Access Control model (ABAC) is of a great interest.

ABAC Model. ABAC is an adaptive and a flexible access control model for the collaboration in the cloud. The core components of ABAC model [9] are:

- U, O and E represent finite sets of existing users and objects and environments respectively. A is a finite set of actions might be noted $A = \{create, read, update, delete\}$.
- $UATT$, $OATT$ and $EATT$ represent finite sets of user, object and environment attribute functions respectively.
- For each att in $UATT \cup OATT \cup EATT$, $range(att)$ represents the attribute's range, which is a finite set of atomic values.
- $attType : UATT \cup OATT \cup EATT \rightarrow \{set, atomic\}$, specifies attributes as set or atomic values.
- Each attribute function maps elements in U to an atomic value or a set
  - $\forall ua \subseteq UATT.\ ua : U \rightarrow Range(ua)\ if\ attType(ua) = atomic$
  - $\forall ua \subseteq UATT.\ ua : U \rightarrow 2^{Range(ua)}\ if\ attType(ua) = set$
- Each attribute function maps elements in O to an atomic value or a set
  - $\forall oa \subseteq OATT.oa : O \rightarrow Range(oa)\ if\ attType(oa) = atomic$
  - $\forall oa \subseteq OATT.oa : O \rightarrow 2^{Range(oa)}\ if\ attType(oa) = set$
- Each attribute function maps elements in E to an atomic value or a set
  - $\forall ea \subseteq EATT.ea : E \rightarrow Range(ea)\ if\ attType(ea) = atomic$
  - $\forall ea \subseteq EATT.ea : E \rightarrow 2^{Range(ea)}\ if\ attType(ea) = set$
- An authorization that decides on whether a user u can access an object o in a particular environment e for the action a, is a boolean function of u, o, and e attributes: Rule: $authorization_a(u, o, e) \rightarrow f(ATTR(u), ATTR(o), ATTR(e))$.

3 Related Work

In the Task based access control [13] (TBAC), the permissions are granted in steps that are related to the tasks progress. The TRBAC [14] model is constructed by adding task to the RBAC model. In TRBAC, the user has a relationship with permissions through roles and tasks. On the other hand, in the Team Access Control Model (TMAC) [12], the permissions are granted to each user through its role and the current activities of the team. These models enable fine-grained access control but they do not incorporate contextual parameters into security considerations and do not support dynamic collaboration during collaborative sessions.

Current access control models for cloud are built on role-based access control (RBAC). There have been very few works for implementing ABAC in cloud. Attributes based access control (ABAC) [10] model brings out many advantages over traditional identity or role based models. Jin et al. [15] present an ABAC framework for access control in cloud IaaS. This paper provides formal models for the operational and administrative aspects of this framework for cloud IaaS. Authors present the implementation of the models based on the open source cloud platform OpenStack. However, this ABAC framework is not dedicated for the swift environment.

Biswas et al. [16] proposes an extension of Swift Object Server where policies might be specified on a Swift object at the content level and let different users access different parts of it. Biswas et al. [17] presents an attribute based protection model for JSON documents. Security-label attribute values are assigned to JSON elements and authorization policies are specified based on these attribute values. This approach is specific to JSON documents, whereas our suggested architecture might be applied to any objects.

4 Architecture and Enforcement Model

In this section, we present the implementation of the ABAC model on the swift storage component. First, we describe the architecture of the extended ABAC module. Then, we present the enforcement model. Finally, we evaluate the implemented approach to demonstrate its feasibility.

4.1 System Architecture

We implemented the ABAC model on the swift storage component. This component acts as a service that communicates with other components (Nova volume, nova compute, nova network, glance and keystone) via message queues. These components are loosely coupled. Keystone is the identity service used by OpenStack for authentication and authorization. It provides a token signed by each users private key.

Let us consider a telemedicine scenario where the School Hospital (SH), the Emergency Medical Services (EMS), and the Home Hospital (HH) are three

collaborating organizations. These organizations share a common private cloud openstack. We consider that these organizations use the swift component for the storage service. In this use case, each organization is assigned to a swift account. (e.g. the accounts ACC_SH, ACC_EMS and ACC_HH represent the organizations SH, EMS and HH respectively).

This cloud provides a service of collaborative sessions for these organisations. This service allows a group of users, from different tenants, to collaborate in order to observe and treat a patient admitted in the Home Hospital (HH) emergency. In this example, we have a collaborative session CS1 of a telemedicine type. During a collaborative session, users may intervene dynamically without a prior knowledge of which user will access which object.

In order to support ABAC Model in the OpenStack Swift environment and overcome the limitations of Swift ACL, we propose to extend the Swift component by implementing a new ABAC Module (Fig. 1). The ABAC module is

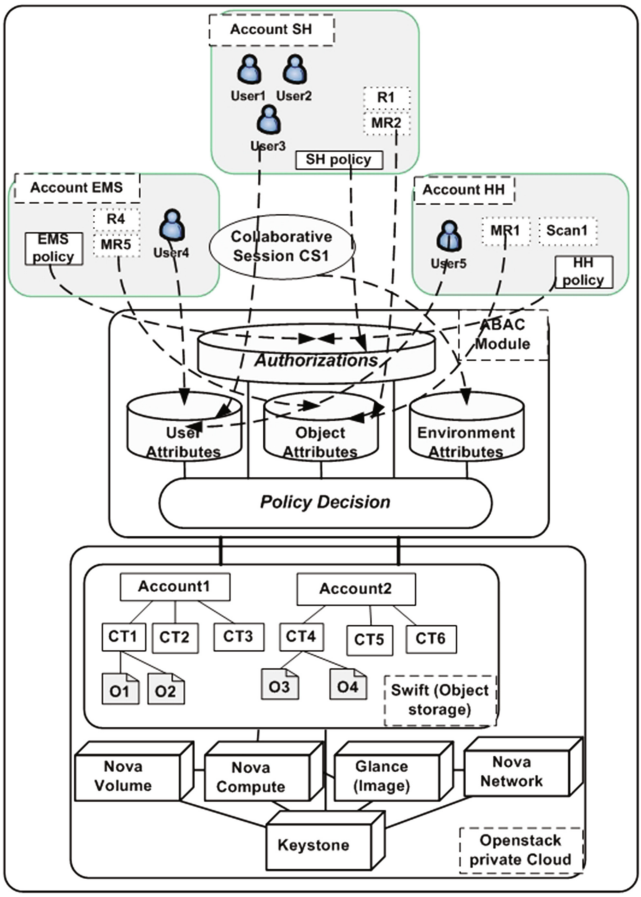

Fig. 1. The system architecture

composed of five components: User attributes, object attributes, environment attributes, authorizations and the policy decision component. In the following, we describe each of these components:

- **User attributes:** The security administrator defines the user attributes as a function that takes user as input and returns a value from the attribute's range. ($user1 : attr1 : val1$) means that for the user $user1$ the value of the attribute $attr1$ is $val1$. For example, a user attribute function such as $Role \in UATT$ maps $user1 \in U$ to a value $neurologist$. Furthermore, the cloud administrator defines the attribute function $UOwner$ to specify the user owner. For instance ($user1 : UOwner : ACC\_SH$) means that the user $user1$ is owned by the account ACC_SH. Finally, the administrator defines the attribute function $JoinCS$ to specify which users could join the collaborative sessions. The value of this attribute is either true or false. ($user1 : JoinCS : true$) means that the user $user1$ could participate in the collaboration.
- **Object attributes:** The tenant administrator assigns the object attributes as a function that takes object as input and returns a value from the attribute's range. ($obj1 : attr1 : val1$) means that for the object $obj1$ the value of the attribute $attr1$ is $val1$. Furthermore, the cloud administrator defines the attribute function $OOwner$ to specify the object owner. For instance ($MR1 : UOwner : ACC\_HH$) means that the object $MR1$ is owned by the account ACC_HH. Finally, the administrator defines the attribute function $SharedCS$ to specify which objects could be shared in the collaborative session. For example, ($Per\_info1 : SharedCS : false$) means that the object Per_info1 (personal information) could not be shared in the collaborative session.
- **Environment attributes:** The security admin defines the environment attributes that describe the environment parameters which represent the context in which the information access occurs. This repository is responsible for users management (e.g. to join/leave the collaborative session). The members of the collaborative sessions are specified with the attribute $Member$ as follows: ($CS1 : Member : \{ACC\_user1, ACC\_user2\}$) means that for the collaborative session $CS1$ the value of the attribute $Member$ is $\{ACC\_user1, ACC\_user2\}$.
  Regarding the shared resources management: ($CS1 : Shared : \{MR1, MR2\}$) means that for the collaborative session $CS1$ the value of the attribute $shared$ is $\{MR1, MR2\}$. Moreover, in this component the tenant admin defines other attributes related to the collaborative session such as: Template [5] (a pattern for a collaboration activity), State of the session. Finally, the administrator specifies the tenant trust relation established by the truster account as defined in [11] in order to support cross-domain collaboration. These relationships are specified with the attribute function $trustUser$ as follows: ($ACC\_SH : trustUser : ACC\_EMS$), which means that the tenant ACC_EMS is authorized to assign values from ACC_EMS's user attributes to Tenant ACC_SH's users that will join the collaborative sessions. In order to support the resources sharing in a collaborative session owned by another tenant, the tenant admin-

istrator defines a new trust relationship by using the attribute function $trustObject$ as follows: $(ACC\_SH : trustObject : ACC\_EMS)$, which means that the tenant ACC_EMS is authorized to assign values from ACC_EMS's object attributes to tenant ACC_SH's objects that will be shared in the collaborative sessions.

- **Authorizations:** The administrator specifies the authorizations policy. In our scenario, we consider that each tenant defines its policy rules. Note that at this level, we suppose that the security policy rules are valid and conflict-free. The policy rules are specified here as follows: $read - u : role : tenant\_admin \wedge cs : template : neuroEmergency \wedge cs : member : u \wedge cs : shared : o \wedge o : objecttype : MR \wedge u : UOwner : UH \wedge o : OOwner : UH$, which means that for the action 'read', this authorization is valid if only if : (1) The user u plays the role tenant_admin in the session cs; (2) There is a collaborative session cs that is an instance of the template $neuroEmergency$; (3) The user u is member of the collaborative session cs; (4) The object o is shared in the session cs; (5) The object type of o is the MR (medical_record); (6) The user u is owned by the tenant UH; (7) The object o is owned by the tenant UH.
- **Policy decision:** This component is responsible for evaluating the access request to the resources in the collaborative session based on the collected attributes values and authorizations. When a user sends a request to access a resource stored in the cloud swift, the policy decision component evaluates this request according to the policy rules in order to decide whether the user is authorized to access this resource or not.

4.2 Enforcement Model

The ACL model defines static access rules. During a collaborative session, a set of users from the same or different tenants join this session and share multiple resources. In our use case, $user1$, $user2$, $user3$, $user4$ and $user5$ are the members of the collaborative session $CS1$ and the objects $MR1$ and $Scan1$ are shared in this session.

A general authorization process for Swift component with ABAC module is illustrated in Fig. 2. When the user $user1$ attemps to access the resource $MR1$ stored in the swift. First, (1) The user requests keysone to get his/her token. (2) Keystone generates a token and sends it to the user. (3) The user sends a request to ABAC module by using his/her token to access the resource $MR1$. The Policy decision component receives this request to evaluate it. (4) During the evaluation process, the policy decision component requests the components: user attributes, object attributes and environment attributes. (5) to receive user1's attributes, $MR1$'s attributes and the attributes related to the collaborative session wherein this user is member. (6) The policy decision component requests the authorizations component and (7) receives all the policy rules stored in this component. These attributes and policy rules will be used by the policy decision to evaluate access request in order to decide whether the user is authorized to access this resource or not. (8) the policy decision will execute an ACL command to assign the authorization decision (permit or deny) to the user in the swift environment.

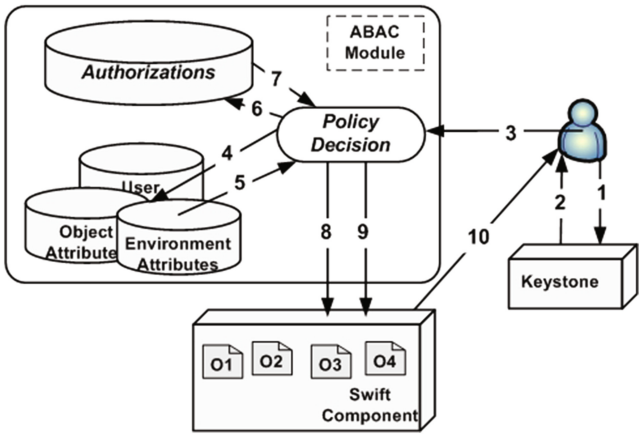

Fig. 2. The enforcement model

(9) The policy decision component executes a swift API command in the swift component using $user1$'s token in order to send the $user1$'s access request to swift. (10) $user1$ access to the resource $MR1$ if the authorization decision is permitted.

5 Implementation and Evaluation

In this paper, we implement the ABAC model on the swift storage component of openstack. Our experiments were run on a virtual machine with the following characteristics (Memory 1024 MB, 2 cores CPU, Hard Disk 30 GB). We consider the download time of a Swift object with and without ABAC module. We observe that the performance of enforcing our approach depends on many factors, such as numbers of rules, number of attributes and number of concurent collaborative sessions. In our analysis, we have used a synthetic dataset that contains up to 500 rules, 200 user attributes and 25 concurent collaborative sessions.

Figure 3(a) shows that the average time to authorize the access to a Swift object increases with 25% and 30.5% for policies of 100 and 500 rules respectively using the ABAC Module. The waiting time for getting a policy decision becomes larger when there are too many authorizations to be collected. We acknowledge that our implementation works well for a medium number of authorizations.

Furthermore, we compute the running time for access/deny decisions to a Swift object with and without ABAC module for 200 rules and user attributes with 40 to 200 UA assignments. Figure 3(b) shows that the average time for download of a Swift resource increases with 26% and 33% for 40 and 200 user attributes assignments respectively using ABAC Module. We acknowledge that our implementation works well for a medium number of authorizations.

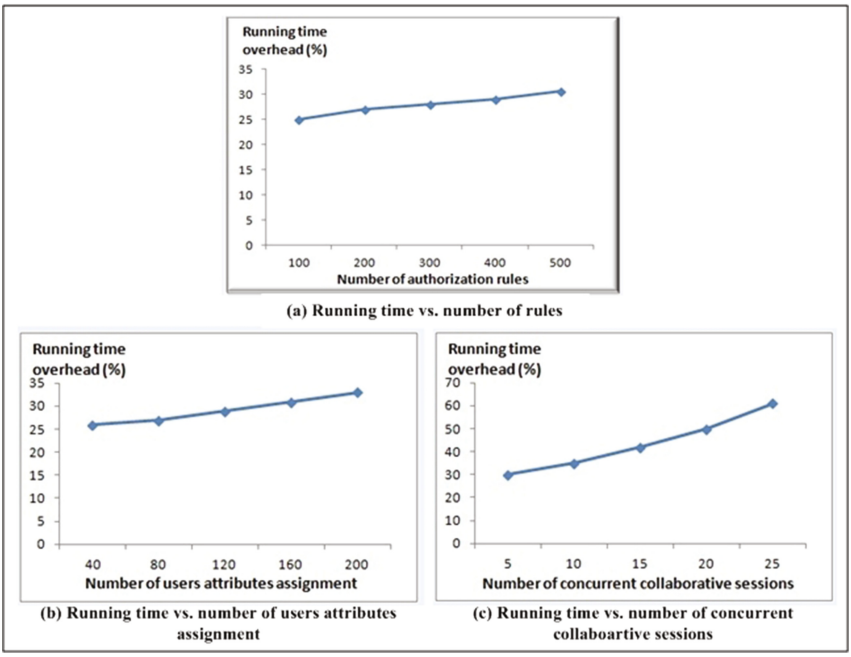

Fig. 3. Running time overhead for access/deny decisions

Finally, we compute the running time for access/deny decisions to a Swift object with and without ABAC module for 500 rules, 80 UA assignments and number of concurrent collaborative sessions with 5 to 25 active ones.

Figure 3(c) shows that the average time for access/deny decisions to Swift resources increases with 30.5% and 61% for 5 and 25 concurrent collaborative sessions respectively using the ABAC Module. We observe that our implementation works well for a medium number of active concurrent collaborative sessions. The overhead reaches 61% in an unusual situations where there are 25 concurrent parallel collaborative sessions.

6 Conclusion

In this paper, we provide an architecture that integrates ABAC in the storage level of the cloud platform. This arcitecture is implemented on the cloud platform Openstack to allow the use of the access control policies based on the ABAC model. It interacts with the Swift component of Openstack for policy enforcement. Therefore, this arcitecture provides a fine-grained access control to support collaborations between multiple organizations allowing a secure data sharing during a collaborative session. The evaluation results have shown that the suggested approach has a very limited overhead when the ABAC module is used.

References

1. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. NIST Special Publication 800–145 (Draft). http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145-cloud-definition.pdf (2011). Accessed 10 Sept 2011
2. Calero, J.M.A., 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)
3. Tang, B., Sandhu, R.: A Multi-Tenant RBAC model for collaborative cloud services. In: 2013 Eleventh Annual International Conference on Privacy, Security and Trust (PST), pp. 229–238 (2013)
4. Takabi, H., Joshi, J.B.D., Ahn, G.J.: SecureCloud: towards a comprehensive security framework for cloud computing environments. In: Proceeding of the 1st IEEE International Workshop Emerging Applications for Cloud Computing, pp. 393–398. Seoul, South Korea (2010)
5. Tanvir, A., Tripathi, A.R.: Specification and verification of security requirements in a programming model for decentralized CSCW systems. ACM Trans. Inf. Syst. Secur. **10**(2), 7 (2007)
6. OpenStack cloud platform. http://www.openstack.org/. Accessed 05 Oct 2016
7. OpenStack Swift Architecture. https://swiftstack.com/openstack-swift/architecture/. Accessed 05 Oct 2016
8. Zhang, Y., Krishnan, R., Sandhu, R.: Secure information and resource sharing in cloud. In: CODASPY 2015—Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pp. 131–133. Association for Computing Machinery, Inc. (2015)
9. 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). doi:10.1007/978-3-642-31540-4_4
10. Yuan, E., Tong, J.: Attributed Based Access Control (ABAC) for web services. In: ICWS, pp. 561–569. IEEE Computer Society (2005)
11. Aydoğan, R., Festen, D., Hindriks, K.V., Jonker, C.M.: Alternating offers protocols for multilateral negotiation. In: Fujita, K., Bai, Q., Ito, T., Zhang, M., Ren, F., Aydoğan, R., Hadfi, R. (eds.) Modern Approaches to Agent-based Complex Automated Negotiation. SCI, vol. 674, pp. 153–167. Springer, Cham (2017). doi:10.1007/978-3-319-51563-2_10
12. Thomas, R.: TMAC: a primitive for applying RBAC in collaborative environment. In: 2nd ACM, Workshop on RBAC, Fairfax, Virginia, USA, pp. 13–19 (1997)
13. Thomas, R., Sandhu, R.: Task-based Authorization Controls (TBAC): a family of models for active and enterprise-oriented authorization management. In: 11th IFIP Working Conference on Database Security, Lake Tahoe, California, USA (1997)
14. Sejong, O.H., Park, S.: Task-role-based access control model. Inf. Syst. **28**(6), 533–562 (2003)
15. Jin, X., Krishnan, R., Sandhu, R.: Role and attribute based collaborative administration of intra-tenant cloud iaas. In: 2014 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp. 261–274 (2014)
16. Biswas, P., Patwa, F., Sandhu, R.: Content level access control for OpenStack swift storage. In: CODASPY, pp. 123–126 (2015)
17. Biswas, P., Sandhu, R., Krishnan, R.: An attribute based protection model for JSON documents. In: NSS, pp. 303–317 (2016)