Expression and Enforcement of Security Policy for Virtual Resource Allocation in IaaS Cloud

Yanhuang Li[1,2(✉)], Nora Cuppens-Boulahia[2], Jean-Michel Crom[1], Frédéric Cuppens[2], and Vincent Frey[1]

[1] Orange Labs, 4 rue du Clos Courtel, 35510 Cesson-sévigné, France
{yanhuang.li,jeanmichel.crom,vincent.frey}@orange.com
[2] Télécom Bretagne, 2 rue de la Châtaigneraie, 35510 Cesson-sévigné, France
{nora.cuppens,frederic.cuppens}@telecom-bretagne.eu

Abstract. Many research works focus on the adoption of cloud infrastructure as a service (IaaS), where virtual machines (VM) are deployed on multiple cloud service providers (CSP). In terms of virtual resource allocation driven by security requirements, most of proposals take the aspect of cloud service customer (CSC) into account but do not address such requirements from CSP. Besides, it is a shared understanding that using a formal policy model to support the expression of security requirements can drastically ease the cloud resource management and conflict resolution. To address these theoretical limitations, our work is based on a formal model that applies organization-based access control (OrBAC) policy to IaaS resource allocation. In this paper, we first integrate the attribute-based security requirements in service level agreement (SLA) contract. After transformation, the security requirements are expressed by OrBAC rules and these rules are considered together with other non-security demands during the enforcement of resource allocation. We have implemented a prototype for VM scheduling in OpenStack-based multi-cloud environment and evaluated its performance.

Keywords: Cloud security · Resource management · Security policy

1 Introduction

Today cloud computing is essentially provider-centric. An increasing number of fiercely competing CSPs operate multiple heterogeneous clouds. In terms of IaaS, each provider offers its own, feature-rich solutions for customer VMs. More significantly, in cloud IaaS, physical hardware is usually shared by multiple virtual resources for maximizing utilization and reducing cost. Unfortunately, this vision suffers from a lack of homogeneity: many cloud virtual resources can not be deployed due to deficiencies in (1) unified expression; (2) interoperability. Lack of unified expression results in vendor lock-in: services are tightly coupled with the provider and depend on its willingness to deploy them. Lack of interoperability stems from heterogeneity of services, and more importantly of

J.-H. Hoepman and S. Katzenbeisser (Eds.): SEC 2016, IFIP AICT 471, pp. 105–118, 2016.
DOI: 10.1007/978-3-319-33630-5_8

service-resource mapping, not compatible across providers. For better interoperability and control, cloud brokering is nowadays the rising approach towards a user-centric vision. It may be seen as a paradigm in delivering cloud resources (e.g. compute, storage, network). With the help of brokering technology, user's security needs will be necessarily considered in cloud and these security requirements can be included in SLA contract which is a legal document where the service description is formally defined, delivered, and charged.

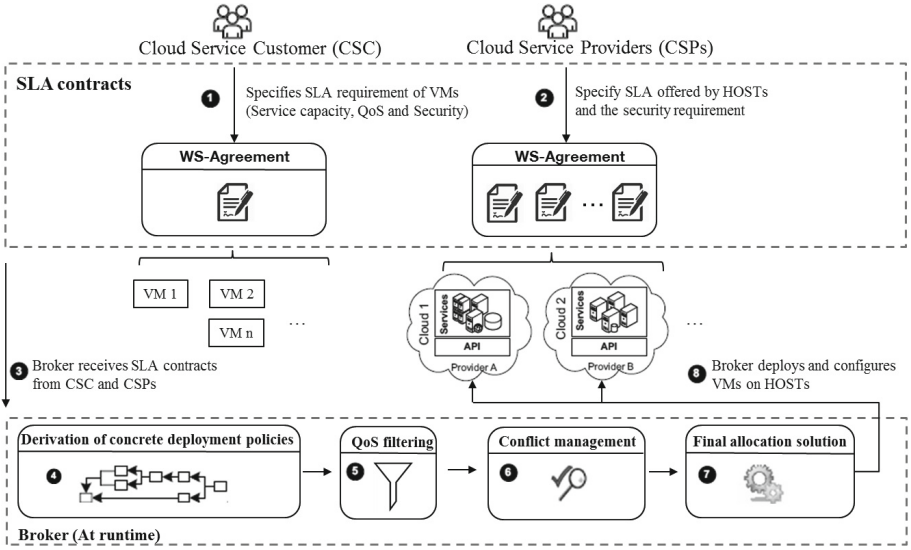

Fig. 1. The proposed policy based process to allocate virtual resources

Therefore, to overcome the aforementioned issues, we enhance the brokering technology by developing a configuration management process to allocate VMs in IaaS cloud. Shown in Fig. 1, with WS-Agreement [1] based contracts, both CSC[1] and CSP specify and manage their security requirements related to infrastructure in order to ensure end-to-end security across different components (Steps 1,2). After receiving the SLA contracts, the broker derives the concrete deployment policies according to security and non-security requirements (Steps 3,4,5). Particularly, the broker is able to arbitrate contradicting demands and make decisions (Step 6). In the end, the broker applies un algorithm to generate the final allocation solution (Step 7) then deploys and configures VMs on HOSTs (Step 8). Our method is evaluated by setting up a cloud computing environment to conduct virtual resource allocation process. Experimental results show that our approach demands minimal user (CSC and CSP)'s intervention and enables unskilled cloud users to have access to complex deployment scenarios. In particular, our solution tackles the lack of application of existing policy model that

[1] In this paper, CSC stands for the end customer of cloud.

can support security expression when dealing with multiple clouds. Our contribution meets key-functional requirement for user-centric as (i) it addresses the SLA configuration options at the IaaS layer from service capacity to security constraint. (ii) it considers multiple requirements of security and applies the OrBAC model to translate attribute-based security constraint to concrete policy. (iii) it provides conflict management to detect and handle the contradictory requirements from CSC and CSPs, with possibility to judge the policy priority by evaluating users' profiles. (iv) it proposes a resource allocation algorithm which takes resource capacity, QoS and security policy into account. To the best of our knowledge, there is no method in the literature that considers all these points.

The rest of the paper is organized as follows: Sect. 2 outlines the expression of security policy by CSC and CSPs with an exhaustive example. Section 3 illustrates the enforcement of security policy for VM allocation. Section 4 gives an implementation integrated with our solution and evaluates four experiments. Section 5 reviews existing proposals on cloud resource scheduling and security-aware allocation solution. Section 6 concludes the paper and outlines future work.

2 Expression of Security Policy

2.1 SLA Contract Expression

To generate security policies for CSC and CSP, we suggest, as a first step, to specify a generic document, which describes the requirements for service capacity, quality of service (QoS) and security constraint. SLA contract is such a document used in service negotiation and management. Based on a well-formatted template, CSP and CSC exchange their offers until reaching an agreement [2]. Among existing SLA specifications, we choose WS-Agreement because the format is open so that it can integrate various service parameters. Meanwhile, WS-Agreement is widely used by lots of research and industrial projects such as BREIN [3], IRMOS [4], and OPTIMIS [5]. Hence a WS-Agreement contract consists of name, context, service terms, guarantee terms and negotiation constraints, CSC and CSP can integrate service capacity, QoS and security requirement in its structure.

In cloud computing, the CSP's system can be viewed as a large pool of interconnected physical hosts. We use HOST to present the finite set of hosts from a CSP. Note that, VM and HOST may have multiple attributes each with their own values and these attributes can be assigned either manually by a user or automatically by the system. In terms of security requirement, as CSC and CSPs do not know the information of each other, they express their security constraints by attribute in Formulas 1, 2 and 3.

$$permission([H_{attr\_name} : H_{attr\_value}], [V_{attr\_name} : V_{attr\_value}]) \tag{1}$$

$$permission([H_{attr\_name} : H_{attr\_value}], [v_i]) \tag{2}$$

$$separate(v_i, v_j) \tag{3}$$

In the three formulas, H_{attr_name} and V_{attr_name} indicate the attribute name of HOST and VM respectively; H_{attr_value} and V_{attr_value} denote separately the attribute value of HOST and VM; each of v_i, v_j represents a unique virtual machine ID (VMID). Formulas 1 and 2 are used to specify the permission of VM allocation: HOST(s) with attributes assigned is (are) permitted to deploy VM(s). The difference is that in the first formula, the CSC specifies VM by attribute and in the second formula, VMID is given directly. These two options give the CSC more flexibility to express their security requirements. In addition, the CSC declares the coexistence constraint by Formula 3: v_i and v_j can not be allocated on the same HOST. Formula 4 is used by CSP to express the deployment prohibition. Similar with Formula 2, HOST with HOSTID h_i is not permitted to deploy VM(s) assigned with attribute.

$$prohibition([h_i], [V_{attr\_name} : V_{attr\_value}]) \tag{4}$$

In an example that we will use throughout the paper, we consider an DevOps [6] use case. DevOps is an emerged software development methodology that enhances collaboration between development, quality assurance (QA) and IT operations. Numerous companies are actively practicing DevOps since it aims to help them to maximize the predictability, efficiency, security, and maintainability of operational processes. Adoption of DevOps is being driven by many factors including using public IaaS. Suppose that a software company has to deploy 3 VMs (v1,v2,v3) in cloud for a development project. Each VM contains its metadata such as properties, required volume, QoS specification and security constraint. We suppose that each VM runs a project server and there exist three types of VM: production (prod), development (dev), and test. *prod* server runs live applications supporting the company's daily business and the data is public for e-business customers; *dev* server consists of development environment thus developers with private right can access it; *test* server is used to conduct software test between development and production phase and it is accessible by testers with private login account. At the same time, there exist 2 CSPs (h1,h2) and each has its own metadata such as price, location and state indicating if it is certificated by security audit organizations. A readable illustration of VM and HOST configuration is shown in Fig. 2.

2.2 Derivation of Security Policy

Security constraints need to be transformed to concrete security policies including VMID and HOSTID. Here we suggest using the OrBAC [7] model which supports the expression of permission and prohibition.

OrBAC in Brief. The OrBAC model is an extension of the role-based access control (RBAC) [8] model. It defines a conceptual and industrial framework to meet the needs of information security and sensitive communication and allows the policy designer to define a security policy independently. The concept of organization is fundamental in OrBAC. An organization is an active entity that

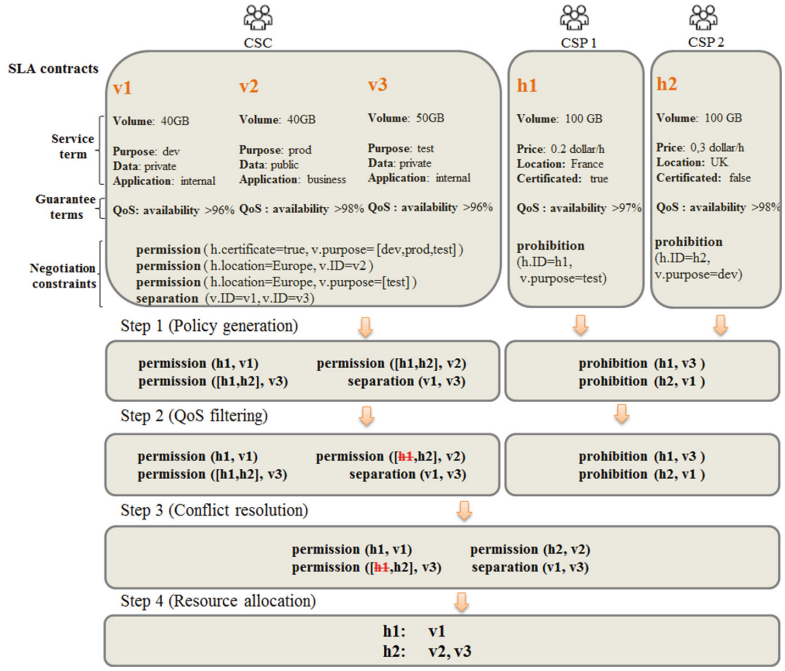

Fig. 2. An DevOps use case of virtual resource allocation

is responsible for managing a security policy. Each security policy is defined for an organization. The model is not limited to permissions, but also includes the possibility to specify prohibitions and obligations. Besides, the security rules do not apply statically but their activation may depend on contextual conditions [9]. Context [10] is defined through logical rules and it can be combined in order to express conjunctive context, disjunctive context and negative context. An OrBAC policy is defined as: **security_rule (organization, role, activity, view, context)** where **security_rule** belongs to {permission, prohibition, obligation}. Once a security policy has been specified at the organizational level, it is possible to instantiate it by assigning concrete entities to abstract entities by the predicates which assign a subject to a role, an action to an activity and an object to a view. Meanwhile, all the operations are related to a specified context:

- *empower(org, subject, role)*: in organization org, subject is empowered in role;
- *consider(org, action, activity)*: in organization org, action implements activity;
- *use(org, object, view)*: in organization org, object is used in view;
- *hold(org, subject, action, object, context)*: in organization org, subject does action on object in context.

Based on the above definitions, a concrete permission policy could be derived by the following rule[2]:

$$permission(org, role, activity, view, context)$$
$$\wedge\, empower(org, subject, role) \wedge consider(org, action, activity)$$
$$\wedge\, use(org, object, view) \wedge hold(org, subject, action, object, context)$$
$$\rightarrow Is\_Permitted(subject, action, object)$$

From Security Constraint to OrBAC Policy. Derivation of OrBAC policy from security constraint requires policy mining technology which parses the configured rules and automatically reaches an instance of high level model corresponding to the deployed policy. Most of the existing RBAC based mining methods [11,12] generate abstract policy by taking concrete rules as input. However, in our scenario, both abstract and concrete rules should be derived from attribute-based description. The following is the problem definition.

Definition 1. *Policy Mining Problem*

Given a set of attribute of Subject S (HOST), a set of attribute of Action A, a set of attributes of Objects O (VM), and SAO_attr an attribute-based subject-action-object assignment relation (Formulas 1, 2, 4), find a set of ROLES, a subject-to-role assignment SR, a set of activity ACTIVITIES, an action-to-activity assignment AA, a set of VIEWS, an object-to-view assignment OV and RAV$\subseteq$*ROLES*$\times$*ACTIVITIES*$\times$*VIEWS, a many-to-many mapping of role-to-activity-to-view assignment relation*[3]*.*

Algorithm 1 explains the generation of permission policy. First of all, after receiving contracts from CSC and CSPs, broker extracts the attribute information of each VM and HOST then generates three kinds of structures as input: (1) VM list: storing all the attributes of related VMs; (2) HOST list: storing all the attributes of related HOSTs; (3) VM security constraint list: storing all the security constraints of CSC. After initialization of policy p, concrete action *deploy* is assigned to a new *activity* (lines 2,3). Then the relevant HOSTID list ID_h_list and relevant VMID list VM_v_list are generated from each term in VM security constraint list c_v (line 4–6). For example, the relevant HOSTID and VMID for the security constraint $permission([\text{“}certificate\text{”} : \text{“}true\text{”}], [\text{“}purpose\text{”} : \text{“}dev\text{”}])$ are HOST1 and VM1. After finding the relevant VMID(s) and HOSTID(s), an abstract permission with a new role *currentRole* and new view *currentView* is created (line 7–9). Finally, all the HOSTIDs in ID_h_list are assigned to *currentRole* and all the VMIDs in VM_v_list are assigned to *currentView* (line 10–15). The prohibition policy for CSP is generated in the same way by taking

[2] A concrete prohibition policy $Is\_Prohibited(subject, action, object)$ could be derived by the same way from $prohibition(org, role, activity, view, context)$.

[3] In this paper, all the rules share the same action (“deploy”), organization (“superCloud”) and context (“default”). For reasons of simplicity, we do not illustrate organization and context in algorithm and policy.

input of VM list, HOST list and HOST security constraint list. Step 1 in Fig. 2 demonstrates an example of permission and prohibition generation.

Algorithm 1. *permissionGeneration*(l_v, l_h, c_v): permission policy generation
Input: VM list l_v, HOST list l_h, VM security constraint list c_v
Output: OrBAC policy p

```
Initiate p
p.activity ← create new activity
p.consider("deploy", p.activity)
for c_vi in c_v do
    ID_h_list ← get relevant HOSTID(s) from l_h
    ID_v_list ← get relevant VMID(s) from l_v
    p.currentRole ← create new role for HOSTs in ID_h_list
    p.currentView ← create new view for VMs in ID_v_list
    p_i ← create permission: permission(p.currentRole, p.activiy, p.currentView)
    for ID_hi in ID_h_list do
        p.empower(ID_hi, p.currentRole)
    end for
    for ID_vi in ID_v_list do
        p.use(ID_vi, p.currentView)
    end for
end for
return p
```

3 Enforcement of Security Policy

3.1 QoS Filtering

Shown in Step 2 of Fig. 2, this process aims to disable the permission which does not satisfy the QoS constraint. To this end, an evaluation between VM's performance requirements and HOST's capacity will be conducted. For example, in our scenario, QoS requirements contain the term of availability and the deployment permission between VM2 and HOST1 is disabled.

3.2 Conflict Management

After generating OrBAC policies from security constraint and executing QoS filtering, the broker aggregates permission rules of CSC and prohibition rules of CSP like:

$$permission(\{h_i\}, v_k) \tag{5}$$

$$prohibition(h_j, \{v_l\}) \tag{6}$$

In Formula 5, each VM v_k has a set of hosts $\{h_i\}$ which allow it to be deployed and in Formula 6, a set of VM $\{v_l\}$ are not permitted to deploy on the HOST

h_j. The rewriting of rules is used to detect conflicts between permissions and prohibitions. A conflict corresponds to a situation where a subject HOST is both permitted and prohibited to perform a given action *deploy* on a given object VM. We divide conflicts into the following two types and for each type an allocation solution is proposed.

Type I: Conflict With Concession Space. Defined in Formula 7, HOST h_j is permitted and prohibited simultaneously to deploy VM v_k. In fact, except for h_j, VM v_k has other allocation solutions. In this case, we disable h_j from the allocation permissions of v_k (Formula 8). For example, in step 3 of Fig. 2, $permission(\{h_1, h_2\}, v_3)$ and $prohibition(h_1, v_3)$ belong to this type and the solution is disabling $permission(h_1, v_3)$.

$$\begin{aligned} conflict\_TypeI(h_j, v_k) \leftarrow permission(\{h_i\}, v_k) \ \wedge \ prohibition(h_j, \{v_l\}) \\ \wedge \ h_j \in \{h_i\} \ \wedge \ v_k \in \{v_l\} \ \wedge \ (\{h_i\} \setminus h_j) \neq \phi \end{aligned} \tag{7}$$

$$disable(permission(h_j, v_k)) \leftarrow conflict\_TypeI(h_j, v_k) \tag{8}$$

Type II: Conflict Without Concession Space. Shown in Formula 9, compared with the conflict of type I, the difference is that in Type II, except for h_i, VM v_k has no other deployment solutions. In this case, we adopt a priority based approach proposed in [13] and introduce two labels $p(h)$ and $p(v)$ as priorities of VM and HOST. $p_1 \prec p_2$ means that p_2 has higher priority than p_1. As virtual resource allocation is related to different factors such as risk and trust, the priorities could be predefined by users or determined by the broker. For example, some of CSPs' prohibitions can be disabled by the broker in case that the CSC has a low risk score. Making decisions on priority is beyond the scope of this paper and here we suppose that CSPs obtain higher priority to fulfill all their security requirements. Thus, in Formula 10, the current conflict resolution is disabling the permission of h_i. For example, the solution for $permission(h_3, v_1)$ and $prohibition(h_3, v_1)$ is disabling the former rule.

$$\begin{aligned} conflict\_TypeII(h_i, v_k) \leftarrow permission(h_i, v_k) \ \wedge \ prohibition(h_j, \{v_l\}) \\ \wedge \ h_i = h_j \ \wedge \ v_k \in \{v_l\} \end{aligned} \tag{9}$$

$$disable(permission(h_i, v_k)) \leftarrow conflict\_TypeII(h_i, v_k) \wedge \ p(v_k) \prec p(h_i) \tag{10}$$

3.3 Virtual Resource Allocation

The aim of previous steps is to generate the final VM allocation solution. Without loss of generality, we demonstrate the generation of allocation solution from security policy by considering the CSC's preference on price. Algorithm 2 shows the resource allocation process. It takes permission policy p, VM list l_v, HOST list l_h and separation constraint c as input and generates the deployment solution which maps VMs to HOSTs. In each permission rule, VMID and a list of its

possible target HOSTs are extracted (line 1–4). To satisfy the price preference of CSC, the target HOSTs are ranked from low price to high price (line 5) thus the one with the lower price will be chosen preferentially. The final deployment solution depends on mainly two factors (line 9): (1) if the VM has coexistence conflict with the VMs which have been already deployed on the HOST. (2) if the HOST has enough volume to deploy the VM. Step 4 in Fig. 2 shows an example of resource allocation.

Algorithm 2. *resourceAllocation*(p, l_v, l_h, c): virtual machine allocation
Input: OrBAC permission p, VM list l_v, HOST list l_h, separation constraint c
Output: deployment solution

```
1: for each concrete rule r_i in p do
2:   if r_i is active then
3:     ID_vi ← get object in r_i
4:     ID_h_list ← get all the HOSTIDs permitted for ID_vi in r_i
5:     Rank ID_h_list from low price to high price
6:     for ID_hj in ID_h_list do
7:       v_i ← get VM from l_v by ID_vi
8:       h_j ← get HOST from l_h by ID_hj
9:       if ID_vi not in separation constraint c
         and h_j has enough volume for v_i
         and v_i has not been allocated then
10:        add (v_i attaches host h_j) to solution
11:      end if
12:    end for
13:  end if
14: end for
15: return solution
```

4 Implementation and Evaluation

SUPERCLOUD [14] is a European project which aims to support user-centric deployments across multi-cloud and enable the composition of innovative trustworthy services. Its main objective is to build a security management architecture and infrastructure to fulfill the vision of user-centric secure and dependable clouds of clouds. One use case is developing a middle-ware layer between CSC and CSPs and this middle-ware could allocate virtual resources on physical infrastructures. In this context, there is a need to consider a multi-cloud environment with security constraints. For example, virtual resources should not be mapped to physical resources that do not comply with its security requirements; physical resources should not deploy virtual resources that are potentially harmful to its operation; or virtual resources should not coexist on the same physical resource as another potentially malicious virtual resource [15].

In order to implement and evaluate our virtual resource allocation framework, we setup an IaaS cloud environment on a physical machine (Intel(R) Core(TM) i7-4600U 2.7 GHz with 16 GB of RAM running Windows 7). Then different VMs (2 cores and 2 GB of RAM) are created on VirtualBox platform with Ubuntu system. We now install DevStack [16] based cloud framework, a quick installation of OpenStack [17] ideal for experimentation. Each VM is regarded as a physical HOST for the purpose of experimentation. At the same time, a JAVA based program runs as cloud broker and it connects VirtualBox platform by SSH protocol. The OrBAC policy is generated and managed by the JAVA-based OrBAC API [18]. Figure 3 illustrates our experimental architecture.

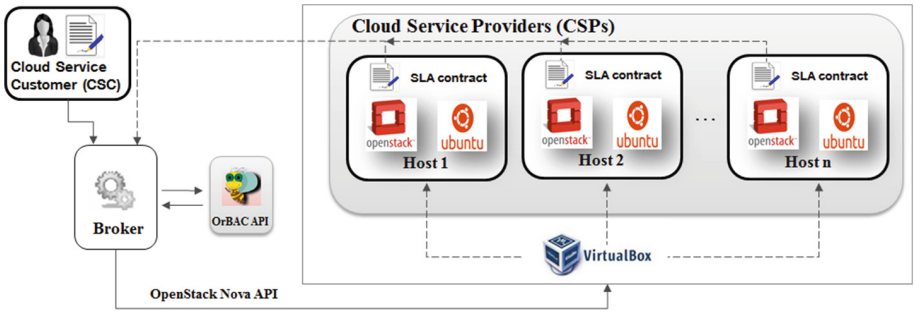

Fig. 3. Implementation for virtual resource allocation

4.1 Experiment 1: Contract Processing

This experiment measures the duration for contract processing which is the runtime required by the broker to process the JSON [19] based WS-Agreement file and generates VM and HOST list. Since there does not exist a great difference between SLA contracts of VM and HOST, here we measure contract processing time for VMs. We vary the VM number from 0 to 125 and for each number we randomly generate service attributes in different quantity from 5 to 20. Figure 4 shows the result. For a small scope of VM and attribute number, the runtime is very low (30 ms). The time increases with bigger scope of VM and attribute number. The maximum duration of the experiment is less than 100ms which indicates that the runtime is acceptable.

4.2 Experiment 2: Policy Generation

In the second experiment, we analyze the required time for OrBAC policy generation (Algorithm 1 for permission and similar algorithm for prohibition generation) once contracts are processed by the broker. In Fig. 5, we study the amount of time the broker takes to generate security policies with increasing number of

VM and HOST. For example, 60 as values in x-axis and y-axis indicates that there exist 60 VMs and HOSTs and the corresponding value in z-axis (400 ms) shows the short time needed to generate the OrBAC policies.

4.3 Experiment 3: Allocation Latency

Our third experiment investigates the impact of VM number and HOST number on the execution time of Algorithm 2. In Fig. 6, VM and HOST number vary from 10 to 60. Given 60 as VM and HOST number, the allocation latency takes about only 1 s. In real case, as HOST number is limited, the estimation of allocation latency is acceptable and it confirms the efficiency of our resource allocation algorithm.

4.4 Experiment 4: Price

The experiment measures the cost for CSC after VM allocation. We generate VMs randomly from 10 to 60 and configure 8 HOSTs. For simplicity, each HOST is supposed to provide only one type of IaaS solution with price fixed from 0.02 dollars/hour to 0.08 dollars/hour[4]. Then we compare the total price between two allocation solutions (Fig. 7). The first solution is Algorithm 2 which concerns CSC's price preference and the second solution does not consider it thus VMs are allocated arbitrary on HOSTs. As a result, Algorithm 2 shows a great advantage in reducing the deployment cost.

5 Related Work

Although virtual resource scheduling problems are NP-complete, it is well-studied by the research community by proposing various heuristic and approximate approaches for addressing different issues. Among three service models (SaaS, PaaS and IaaS) of cloud computing, virtual resource allocation in IaaS cloud has been considered by some works in the literature. Some of these works [20,21] focus on the capacity of CSP. In this case, some strategies like immediate, best effort and Nash equilibrium [22] have been applied to allocation algorithm in order to optimize the deployment algorithm with constraints such as QoS and energy [23]. Another effort is SLA-oriented resource management [24]. Among lots of requirements of CSC, security is a critical issue to be taken into account [25]. Bernsmed et al. [26] present a security SLA framework for cloud computing to help potential CSCs to identify necessary protection mechanisms and facilitate automatic service composition. Berger et al. [27]

[4] The prices are inspired from current cloud IaaS solution of Amazon EC2 and Microsoft Azure. For example, in Amazon EC2, price for the instance of m4.xlarge (4 cores, 16 GB RAM) is 0.239$/h and it costs 0.308$/h (4 cores, 7 GB RAM) for the instance of A3 in Microsoft Azure.

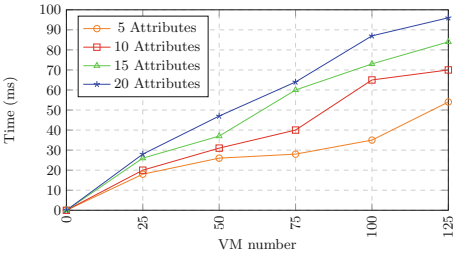

Fig. 4. Time for contract processing

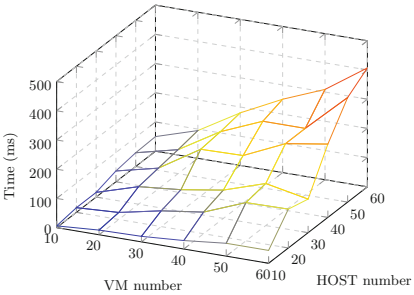

Fig. 5. Time for policy generation

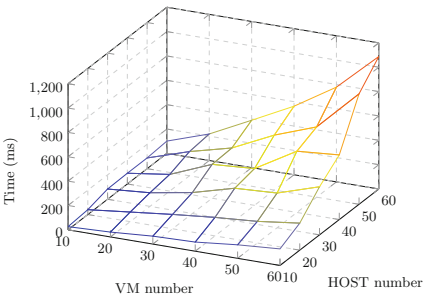

Fig. 6. Latency for VM allocation

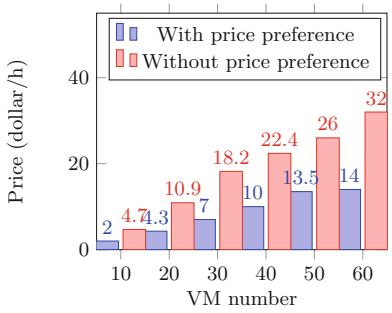

Fig. 7. Total price for VM allocation

take isolation constraint and integrity guarantee into consideration and implement controlled access to network storage based on security labels. In [28], different virtual resource orchestration constraints are resumed and expressed by attribute-based paradigm. Regarding these constraints, a conflict-free strategy is developed to mitigate risks in IaaS Cloud [29]. Most of above works have been motivated from security requirements expressed by CSC. In [30], CSP specifies its security requirements including forbid constraint which forbids a set of VM instances from being allocated on a specified HOST. However, in multi-cloud environment, as CSC and CSPs do not have vision of each other before establishing contract, specifying security requirement can be very tricky for both sides. The main focus of these efforts is scheduling VMs either for the purpose of high-performance computing or satisfying security constraints according to the requirements of CSC. Our approach is to capture security and non-security requirement from both CSC and CSP, and apply a formal policy model to drive virtual resource allocation.

6 Conclusion

In this paper, we have presented, formalized and enforced security requirement for virtual resource allocation. We first present the SLA contracts for CSC and

CSPs which contain service capacity, QoS and security constraint. We then transform the attribute-based SLA contract to concrete OrBAC policies. Finally, we allocate virtual resources after resolving conflicts in policies and demonstrate the efficiency and reliability of our solution by OpenStack-based implementation.

In future works, we plan to investigate the decision making during the conflict resolution. Another potential direction is to develop a suitable front-end application interface for SLA contract specification.

Acknowledgments. The work reported in this paper has been supported by ANRT (Association Nationale de la Recherche et de la Technologie) and Orange as a CIFRE (Conventions Industrielles de Formation par la REcherche) thesis and the work of Nora Cuppens-Boulahia and Frédéric Cuppens has been partially carried out in the SUPERCLOUD project, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 643964.

References

1. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement specification (ws-agreement). In: Open Grid Forum. vol. 128, 216 (2007)
2. Li, Y., Cuppens-Boulahia, N., Crom, J.M., Cuppens, F., Frey, V.: Reaching agreement in security policy negotiation. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 98–105. IEEE (2014)
3. Muñoz, H., Kotsiopoulos, I., Micsik, A., Koller, B., Mora, J.: Flexible SLA negotiation using semantic annotations. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 165–175. Springer, Heidelberg (2010)
4. http://www.irmosproject.eu/
5. Ziegler, W., Jiang, M., Konstanteli, K.: Optimis sla framework and term languages for slas in cloud environment. OPTIMIS Project Deliverable D 2 (2011)
6. http://en.wikipedia.org/wiki/DevOps
7. Kalam, A.A.E., Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks, POLICY 2003, pp. 120–131. IEEE (2003)
8. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **2**, 38–47 (1996)
9. Cuppens, F., Cuppens-Boulahia, N.: Modeling contextual security policies. Int. J. Inf. Secur. **7**(4), 285–305 (2008)
10. Coma, C., Cuppens-Boulahia, N., Cuppens, F., Cavalli, A.R.: Context ontology for secure interoperability. In: Third International Conference on Availability, Reliability and Security, ARES 2008, pp. 821–827. IEEE (2008)
11. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: finding a minimal descriptive set of roles. In: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, pp. 175–184. ACM (2007)
12. Hachana, S., Cuppens-Boulahia, N., Cuppens, F.: Mining a high level access control policy in a network with multiple firewalls. J. Inf. Secur. Appl. **20**, 61–73 (2015)

13. Cuppens, F., Cuppens-Boulahia, N., Ghorbel, M.B.: High level conflict management strategies in advanced access control models. Electron. Notes Theor. Comput. Sci. **186**, 3–26 (2007)
14. http://www.supercloud-project.eu/
15. Fernando, M.V., Ramos, N.N.: Preliminary architecture of the multi-cloud network virtualization infrastructure. Technical report (2015)
16. http://docs.openstack.org/developer/devstack/
17. https://www.openstack.org/
18. Autrel, F., Cuppens, F., Cuppens-Boulahia, N., Coma, C.: Motorbac 2: a security policy tool. In: 3rd Conference on Security in Network Architectures and Information Systems (SAR-SSI 2008), pp. 273–288. Loctudy, France (2008)
19. http://en.wikipedia.org/wiki/JSON
20. Ferreira Leite, A., Alves, V., Nunes Rodrigues, G., Tadonki, C., Eisenbeis, C., Alves, M., de Melo, A.C.: Automating resource selection and configuration in inter-clouds through a software product line method. In: 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), pp. 726–733. IEEE (2015)
21. Nathani, A., Chaudhary, S., Somani, G.: Policy based resource allocation in IaaS cloud. Future Gener. Comput. Syst. **28**(1), 94–103 (2012)
22. Wei, G., Vasilakos, A.V., Zheng, Y., Xiong, N.: A game-theoretic method of fair resource allocation for cloud computing services. J Supercomputing **54**(2), 252–269 (2010)
23. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. **28**(5), 755–768 (2012)
24. Buyya, R., Garg, S.K., Calheiros, R.N.: Sla-oriented resource provisioning for cloud computing: challenges, architecture, and solutions. In: International Conference on Cloud and Service Computing (CSC 2011), pp. 1–10. IEEE (2011)
25. Li, Y., Cuppens-Boulahia, N., Crom, J.-M., Cuppens, F., Frey, V., Ji, X.: Similarity measure for security policies in service provider selection. In: Jajodia, S., Mazumdar, C. (eds.) ICISS 2015. LNCS, vol. 9478, pp. 227–242. Springer, Heidelberg (2015). doi:10.1007/978-3-319-26961-0_14
26. Bernsmed, K., Jaatun, M.G., Undheim, A.: Security in service level agreements for cloud computing. In: CLOSER, pp. 636–642 (2011)
27. Berger, S., Cáceres, R., Goldman, K., Pendarakis, D., Perez, R., Rao, J.R., Rom, E., Sailer, R., Schildhauer, W., Srinivasan, D., et al.: Security for the cloud infrastructure: trusted virtual data center implementation. IBM J. Res. Dev. **53**(4), 1–12 (2009)
28. Bijon, K., Krishnan, R., Sandhu, R.: Virtual resource orchestration constraints in cloud infrastructure as a service. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pp. 183–194. ACM (2015)
29. Bijon, K., Krishnan, R., Sandhu, R.: Mitigating multi-tenancy risks in iaas cloud through constraints-driven virtual resource scheduling. In: Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, pp. 63–74. ACM (2015)
30. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: IEEE 15th International Conference on Computational Science and Engineering (CSE 2012), pp. 170–177. IEEE (2012)