Enhance OpenStack Access Control via Policy Enforcement Based on XACML

Hao Wei, Joaquin Salvachua Rodriguez and Antonio Tapiador
Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, Madrid, Spain

Keywords: Cloud Computing, Security, Authorization, XACML, Access Control, OpenStack.

Abstract: The cloud computing is driving the future of internet computation, and evolutes the concepts from software to infrastructure. OpenStack is one of promising open-sourced cloud computing platforms. The active developer community and worldwide partners make OpenStack as a booming cloud ecosystem. In OpenStack, it supports JSON file based access control for user authorization. In this paper, we introduce a more powerful and complex access control method, XACML access control mechanism in OpenStack. XACML is an approved OASIS standard for access control language, with the capability of handling all major access control models. It has numerous advantages for nowadays cloud computing environment, include fine-grained authorization policies and implementation independence. This paper puts forward a XACML access control solution in OpenStack, which has Policy Enforcement Point (PEP) embedded in OpenStack cloud service and a XACML engine server with policy storage database. Our implementation allows OpenStack users to choose XACML as an access control method of OpenStack and facilitate the management work on policies.

1 INTRODUCTION

Cloud computing is an emerging computing paradigm today. Many people see it as the future generation of utility computing. The concept of "Cloud" extends from Internet of Service, includes Web Service, Service-Oriented Architecture (SOA), and Utility Computing like Grid, Virtual Organizations, also from visualization and outsourcing (Schubert and Jeffery, 2012). It's the solution to the challenges of modern IT requirements: high performance, accessibility, scalability and availability, etc.

The term Cloud Computing denotes the services of infrastructure, platform and software. Rapidly, it changed the way people used to think and act. Now businesses and users could just pay for the services they need, instead of buying software or purchasing servers, data centers.

In the era of cloud computing, the system protection of cloud is a critical issue. Data security and resource security, those two issues raised serious concerns both from users and cloud service providers. Users concern that the privacy and confidentiality of the data stored in the cloud, while cloud service providers concern that the safety of the resources which are opened to the public.

Access control is a classic method utilized to constrain the activity of users. This authorization determines that if a legitimate user could execute an action on certain resources. In area like cloud computing, XACML is an operational access control method and has proven ability in distributed environment(Lorch et al., 2003). It supports fine- and coarse-grained authorization policies, allows an interchangeable policy format.

What follows is that discussion of OpenStack architecture and some of OpenStack components. In the section following, the contemporary mechanism of access control in OpenStack is discussed. The introduction of XACML is in the section IV. Then in the next section we presented our design to implement XACML access control in OpenStack. In the end, the implementation of a prototype and two experiments which validate the feasibility of our proposal and show the ability of XACML.

2 OpenStack AND ITS ARCHITECTURE

OpenStack is an open-sourced cloud operating system, it enables the users to provide the computing, storage and networking resource pool through a sim-

Wei H., Salvachua Rodriguez J. and Tapiador A..
Enhance OpenStack Access Control via Policy Enforcement Based on XACML.
DOI: 10.5220/0004893802830289
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 283-289
ISBN: 978-989-758-028-4

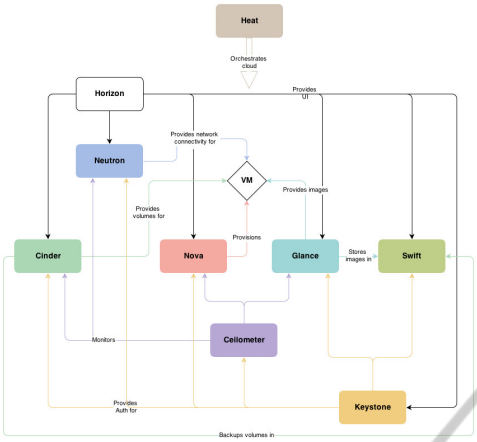

Figure 1: OpenStack conceptual architecture (OpenStack, 2013b).

ple web service(OpenStack, 2013b). With increasingly popularity and active developer community, it becomes the one of the projects of Infrastructure As A Service (IAAS) with great potential.

Among many open-sourced clouds, like Eucalyptus, OpenNebula, OpenStack has good scalability to handle massive resource. While, Eucalyptus scalability is limited and the source code of some modules is closed (Sefraoui et al., 2012) and OpenNebula is aimed to build up private cloud with a few cloud machines(Mahjoub et al., 2011). Further more, the better compatibility of OpenStack with supporting of most virtualization solutions, like KVM, HyperV, LXC, etc. makes it popular and promising in the cloud solution market(Wen et al., 2012).

2.1 OpenStack Components

OpenStack is collective of several projects as showed in Figure 1 they are different aspects of cloud computing. Nova is the main part of OpenStack system. It hosts and manages VM instances. With provided API, Nova has good compatbility with Amazon Web Services (AWS).(Beloglazov et al., 2012).

Glance is the name of OpenStack Image Service, which provides retrieval, storage and meta-data assignment for images running on OpenStack cloud. This service supports multiple VM image formats, such as Raw, AMI, VHD, etc.

Neutron Networking Service (former name Quantum) offers Network As A Service (NAAS). It has APIs to manage networks, build rich networking topologies and configure advanced network policies.

Swift and Cinder both provide storage services. Swift is OpenStack Object Store project and offers highly scalable and durable multi-tenant object storage service for large amounts of unstructured data. While Cinder is the block storage service, with which user could manage volumes.

Keystone is the identity service in OpenStack, it includes user management and service catalog of available services with their API endpoints.

Horizon is dashboard for whole OpenStack platform; Ceilometer is the infrastructure of measuring and collecting data; Heat aims to create a human and machine accessible service for managing entire life cycle of infrastructure and applications.

3 ACCESS CONTROL IN OpenStack

Each of OpenStack cloud service has its access control module. The modules use different ways to validate access privilege, except Swift has no policy check. The similarities among these modules are : 1) JSON is the format of access control policy files; 2) the phase of policy check is before the execution of API from each cloud service; 3) The policy engine only returns True or False, indicate “allow” or “disallow”.

Generally Nova, Neutron, Glance and Keystone share the same policy engine. It is the most popular and typical engine in OpenStack. Though Keystone’s policy engine is slightly different from others. Its policy engine has an Enforcer class. And also not like other services API to check the whole access privilege, Keystone policy check only validates whether the user has an administrator role.

While Cinder has another type of policy check engine. The algorithm simply validates rule, role and remote target information with policy file in Cinder. The imaginable reason is that operations on volumes could also be controlled by Novas policy check.

3.1 Nova Access Control: Algorithm and Engine

We take Nova’s policy engine as an example. This engine is mainly combined by different checks.They validate requests against policies and give the result. The checks are logic checks, like False check, True check, Not check, And check and Or check; and some other checks, like Rule check, Role check, Http check and Generic check. Rule check checks if request matches policy rule. Role check is to see if user role information matches policy rule. Http check validates the availability of a remote server.

The process is that when the cloud service starts, policy engine loads the JSON policy file, and parses

the rules in policy file into different check objects as introduced above. These objects are stored in cache. The operation on cloud service executes through APIs. Before an API takes any action, it sends a request to the policy engine to check the access privilege and receives the result. As part of the request, a context object which contains user information, project information and the action is also sent.

3.2 Nova Policy Logic

In OpenStack, policy is composed by lines of rules. Each rule contains a string of resource and action, followed by a rule check. Policy engine looks for the resource and action string, then perform the request validation against rule check. The rule check uses name-value pairs, "and" and "or" logics. The rule check has three kinds of checks. First two are: 1) "@" or an empty rule means accept all the access; 2) "!" means reject all the access. The third one is the most common check. It can be expressed in two forms: a list of lists, or a policy language string. For the list of lists expression, the innermost lists combine with "and" logic, and then with "or" logic. The policy language is similar with list of lists, but with one more logic of "not" operator(OpenStack, 2013a).

3.3 Problem Area

In the time of cloud, open and sharing, users' data, even their infrastructure are shifting from private, self-owned to public, third party stored. Information security and protection are never more vital than now. OpenStack provides a basic functioned access control method, it can be easy to handle some requirements of authorization. However, there are still some issues we found that needs improvements on.

1. Flexibility of access control. OpenStacks JSON based access control module still cannot compare with XACML access control, which has a sophisticated logic of policy validation. OpenStack's access control policy language faces a lot of limitations when compared with XACML language. Specially, XACML version 3 brings more useful features and combining algorithms.

2. Modification of policy file. If an administrator wants to modify the access privileges for users, he has to, at first, change the policy file and then restart the cloud services to reload the policy files in order to make the effect. Because when cloud service starts, they load the policy files in the cache, even the policy file is changed, the policy in cache is still the same. However, the policy engine provides a method to reload the policy file, but could not find that there is a mechanism to call this method. This brings the trouble that every time policy file changes, the service will have to restart. That could further affect stability of cloud services.

3. Policy files management. Each service has its specific policy file in the installation folder. It is a challenge for admin to universally manage the access privileges. Like explained, if an administrator makes adjustments to the access privileges for a user role, then he needs to visit each services installation folder, modify the policy files and restart all the services. That brings extra workload, and difficulty with the management of OpenStack access control policies.

4. Security issue of policy files in OpenStack. Keystone, Nova and all other cloud services simply put access control files into one JSON file, which is stored in the installation folder. They are susceptible to all the users who can visit this folder. This constitutes a security issue and brings risks to the access control module.

4 ACCESS CONTROL WITH XACML

XACML, eXtensible Access Control Markup Language, an XML implemented access control language specified by Organization for the Advancement of Structured Information Standards (OASIS)(Erik, 2012). It is the one of most popular and widely used access control standard, and OASIS XACML Technical Committee includes member from Oracle, IBM, Cisco, etc.

The advantages of XACML are, firstly, its simplicity and strength which make it suitable for numerous environments. Specially, its independence from implementation could be utilized in cloud computing and cloud federation; Secondly, XACML's RBAC profile well supports RBAC function and due to the large and well-adopted XML ecosystem, XACML is the one of the best RBAC and policy based authorization method; Moreover, compared with IBM's Enterprise Privacy Authorization Language (EPAL), XACML has more features on complex enterprise policies and is better standardized and implemented (Anderson, 2005).

This standard defines both policy language and request/response language. They express the access control policies, the queries and the answers of the queries. In the policies, XACML specifies three main parts, such as PolicySet, Policy, Rule. PolicySet is the root of other PolicySets or Policies, and Policy could contain multiple Rules. One Rule in policy is the minimum unit of policy check, could return results: per-

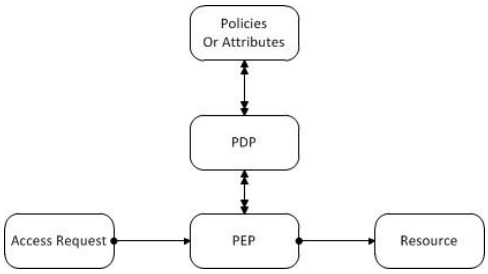

Figure 2: XACML data flow diagram.

Table 1: Abstraction of XACML 3.0 Components.

XACML Policy Components		
PolicySet	::= $\langle$Target, $\langle$PolicySet$_1, \ldots,$ PolicySet$_m\rangle$, CombID$\rangle$	
	| $\langle$Target, $\langle$Policy$_1, \ldots,$ Policy$_m\rangle$, CombID$\rangle$	where $m \geq 0$
Policy	::= $\langle$Target, $\langle$Rule$_1, \ldots,$ Rule$_m\rangle$, CombID$\rangle$	where $m \geq 1$
Rule	::= $\langle$Effect, Target, Condition$\rangle$	
Condition	::= *propositional formulae*	
Target	::= `Null`	
	| AnyOf$_1 \wedge \ldots \wedge$ AnyOf$_m$	where $m \geq 1$
AnyOf	::= AllOf$_1 \vee \ldots \vee$ AllOf$_m$	where $m \geq 1$
AllOf	::= Match$_1 \wedge \ldots \wedge$ Match$_m$	where $m \geq 1$
Match	::= *att*(*val*)	
CombID	::= `po` | `do` | `fa` | `ooa`	
Effect	::= **d** | **p**	
att	::= `subject` | `action` | `resource` | `enviroment`	
val	::= *attribute value*	
XACML Request Component		
Request	::= $\{ A_1, \ldots, A_m \}$	where $m \geq 1$
A	::= *att*(*val*) | `error` | *external state*	

mit, deny, not applicable or interminate. In both Policy and Rule, a target is a match condition which determine whether the request is applicable to the policy or rule. While combination algorithms in PolicySet and Policy weigh the results from Policies and Rules, and then give the final result. Along with Obligation expressions and Advice expressions, therefore XACML could be able to implement complex logic to access control policies.

A general XACML data flow model is: an XACML access request is sent to Policy Enforcement Point (PEP). Then the PEP sends request to Policy Decision Point (PDP). PDP evaluates the request against policy, and returns the response to PEP. We can see the flow from Figure 2.

To get an overview of XACML logic, the semantic of XACML v3 specification could almost be formalized to multi-valued logic approach, like Belnap logic(Belnap Jr, 1977) and D-algebra(Ni et al., 2009). Generally, the Table 1 (Ramli et al., 2013) describes the syntax of XACML v3.

5 XACML IN OpenStack

Implementing XACML is to apply XACML access control to OpenStack and as an alternative to the native JSON based access control. The design of XACML access control solution in OpenStack doesn't only apply the XACML standard to OpenStack and extend the access control ability, but also facilitates the management work and improves the security on policies. This design bases on existing architecture and native access control design concepts of OpenStack. Meanwhile it tries to leave some space for future requirement.

The following section discusses the design consideration, the solution and use of XACML in OpenStack. This design is for OpenStack, Grizzly release, the seventh stable release at the time of this research.

5.1 Design Consideration

The cloud services already have a simple access control. Based on several experiments on Nova, we have verified that there is a data package contains user credential information like user name, token, user role, action name etc. Therefore all the information a PEP needs is already existing in OpenStack. For the security reason of not exposing the credential information and the architecture of OpenStack cloud service, the PEP should build inside the cloud service in order to safely obtain the data package.

Furthermore, we have investigated several XACML implementations. Balana is an XACML v3 well-supported open source software with Java(WSO2, 2012). It developed from widely adopted Sun XACML implementation, which only has supports for XACML v2. Additionally OpenStack is implemented with Python. So to address the problem of languages and also to decouple the XACML access control module to other OpenStack modules. The access control module should be designed as a service. Inside XACML access control module, we also selected an open sourced native XML database, eXist-db(Meier, 2003), as policy storage.

5.2 Use XACML in OpenStack

As discussed above, XACML has more potent ability to handle complex access control requirement. Integrated with XACML in OpenStack could provide more comprehensive authorization functions. Basically, access control in Grizzly release only does the user role match to each of the resource. For example, we query all the network instance, the action is "get_all", the resource is "network". So the access control will look for a rule of "network:get_all" in the policy, and check whether the user role of user who wants to execute this action can match with the user role specified in that rule. However, besides this plain access check, XACML could implement a much more complex policy. This will be discussed in experiment section.

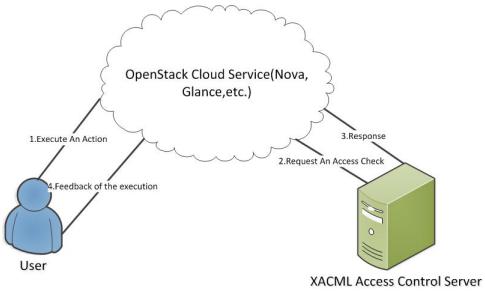

Figure 3: XACML access control in OpenStack.

This design of XACML access control module also centralized all the policies in separate cloud services into one place. Through eXist-db database, administrator could manage policies for the whole OpenStack platform. More importantly, this modification of policies could be made on the fly, it will take effect without restarting the cloud service.

In addition, Store the policy files in database also improves the security level of access control of OpenStack. It could not be easily accessed, considering that before the policy file is just in the installation folder.

5.3 XACML Access Control as a Service

For design considerations mentioned above, in our solution, XACML access control server is detached from OpenStack. Therefore, the server is as a universal authorization interface which could accept requests from all the cloud services.

The interaction is showed in Figure 3. When a user calls a cloud service API, the API needs to check the authorization of the user at first. The API connects to the XACML access control server and sends the authorization request. The server receives and validates the request, then constructs a response and sends it back. When the API gets the response, it will validate the response. If the response is positive, the API will execute the action user requests and returns the result to the user. Otherwise it will reject user's request. The detailed prototype architecture is discussed in following section.

6 PROTOTYPE IMPLEMENTATION

We implemented a prototype on Nova. However, the same architecture could also apply to other cloud services.

XACML access control in OpenStack involves two parts: PEP part is in Nova, it abstracts necessary data and constructs XACML request then sends to PDP in XACML server; all other parts of XACML access control are organized in the server.

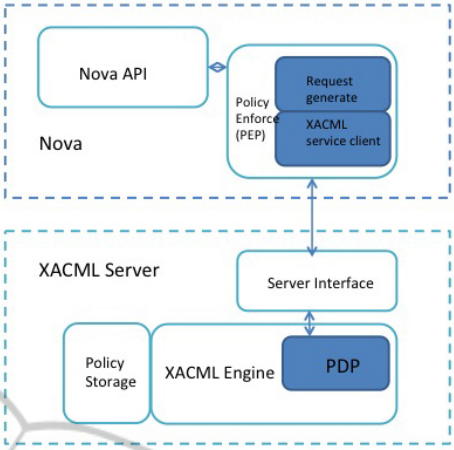

Figure 4: Prototype architecture for XACML access control in OpenStack.

Therefore each of OpenStack cloud service has one PEP connect to XACML access control server. All the policies are universally managed and stored in XML database.

As Figure 4 shows, the policy enforce is as an interface, gets data package from cloud service API and returns result of access control check; request generation component constructs XACML request from given data which is from cloud service API; the XACML client component is responsible to configure the client, connect to XACML server and listen to the response from server. The policy enforce checks the result, and returns it to cloud service API.

In XACML standard, it defines several results for the response of evaluation, however, here we process these result only into True or False, and only the case of "Allow" will be considered as True, other cases are all False.

On server, XACML server starts, PDP and other parts of XACML initiate and be ready to receive request. When server receives the XACML request from the client and passes it to PDP; PDP evaluates the request and returns the response; the response will be processed in order to provide a suitable result to server; in the last, server sends back the result to client.

6.1 Two Experiments

We also designed two sets of experiments to demonstrate the functionality and advantages of this XACML access control module. Experiment One is designed to test that the entire prototype we developed

Table 2: Policy of Experiment One.

Policy	Target Rule	"network" Effect Target Condition	"Permit" "create" or "get_all" Function-id AttributeValue	 "string-equal" "admin"

Table 3: Result of Experiment One.

Resource	Action	Role	Result
Network	Create	Admin	True
Network	Get_all	Admin	True
Compute	Get_all	Admin	False
Network	Create	Member	False
Network	Delete	Admin	False

is functional, could be utilized to do the access control check and is capable of handling various results. Experiment Two illustrates the advantages of expressiveness of XACML. In modern enterprise access control scenarios, XACML could be more suitable and have more features than OpenStack's native access control module. Experiment Two uses the benefits of powerful XACML policy language to handle the complex access control cases, in which the original module is limited.

6.1.1 Experiment One

We took a network control in Nova as an example. We utilize Experiment One to verify the proposed design and prototype.

We defined a policy that defines if user role is "admin", he could do "create" or "get_all" actions on resource "network". There are totally 12 possible combinations to test. However, we only prepared 5 different requests, which are sufficient for covering the major route of policy check and getting all kinds of results. The policy is showed as Table 2. The requests are showed as Table 3.

6.1.2 Experiment Two

Experiment Two has two scenarios as examples to see how XACML access control could satisfy the complex requirement in contemporary days, while OpenStack's access control is not able to handle these situations. The first scenario introduces the data type of time into the policy, in order to add a time constraint to access control. The second one utilizes string function in XACML policy language to achieve the conditional authorization. Because of the solo user role check mechanism, the original access control could not deal with any of these scenarios. These two scenarios are:

Table 4: Policy of Experiment Two: Scenario 1.

Policy	Target Rule	"network" Effect Target Condition	"Deny" "create" or "delete" Function-id AttributeValue	 "Datetime-equal" 2013-09-01

Table 5: Policy of Experiment Two: Scenario 2.

Policy	Target Rule	"network" Effect Target Condition	"Permit" "manager" Function-id AttributeValue	 "String-contains" "@company.com"

1) Due to maintenance, on date "1st, September, 2013" user with user role "Member" could not do "create" or "delete" action on resource "network";

2) Users with user role "Manager" and its name contains "@company.com" could do "create" action on the resource "network". The scenario 1 is inspired by (Evered and Bögeholz, 2004).

Both of the policy file structures are showed on Table 4 and Table 5.

7 CONCLUSIONS AND FUTURE WORK

To improve the security, OpenStack uses JSON file based Role Based Access Control (RBAC). For each combination of resource and action, it defines a condition in order to establish whether a user could perform this action or not. However, due to the logic of access control check in OpenStack, the current mechanism has its limits to meet the requirement of modern cloud computing system. Also it exposes some security issues, like policy file security, the difficulty of policy file management and the variety of policy engines in different cloud services.

In this paper, we introduce XACML access control in OpenStack. In each cloud service of OpenStack, an XACML PEP is embedded in authorization part and responsible for generating, sending requests and receiving the responses from XACML server. An independent XACML server will process requests and manage policies, it contains a PDP and a policy storage database. As a result, user is able to switch from OpenStack native access control to XACML access control. The prototype of our design is successfully implemented and is able to work on OpenStack Grizzly release.

This research presented here enables the possibility of future work. Based on this work, the very first

objective would be to provide the solution of building the identity and access control federation. In the future, we aim to design a hybrid cloud solution with good scalability and security with OpenStack platform. That means for a private cloud it is easy to acquire public cloud resource to form hybrid cloud, and also easy to withdraw from hybrid cloud back to private cloud without scalability and security problems.

ACKNOWLEDGEMENTS

This work developed in collaboration with the Project Fi-ware, funded by the 7th EU R & D Framework Programme (subsidy agreement number 285248).

REFERENCES

Anderson, A. (2005). A comparison of two privacy policy languages: Epal and xacml.

Belnap Jr, N. D. (1977). A useful four-valued logic. In *Modern uses of multiple-valued logic*, pages 5–37. Springer.

Beloglazov, A., Piraghaj, S. F., Alrokayan, M., and Buyya, R. (2012). Deploying openstack on centos using the kvm hypervisor and glusterfs distributed file system. Technical report, Technical Report CLOUDS-TR-2012-3, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne.

Erik, R. (2012). Extensible access control markup language (xacml) version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs02-en.html.

Evered, M. and Bögeholz, S. (2004). A case study in access control requirements for a health information system. In *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation-Volume 32*, pages 53–61. Australian Computer Society, Inc.

Lorch, M., Proctor, S., Lepro, R., Kafura, D., and Shah, S. (2003). First experiences using xacml for access control in distributed systems. In *Proceedings of the 2003 ACM workshop on XML security*, pages 25–37. ACM.

Mahjoub, M., Mdhaffar, A., Halima, R. B., and Jmaiel, M. (2011). A comparative study of the current cloud computing technologies and offers. In *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pages 131–134. IEEE.

Meier, W. (2003). exist: An open source native xml database. In *Web, Web-Services, and Database Systems*, pages 169–183. Springer.

Ni, Q., Bertino, E., and Lobo, J. (2009). D-algebra for composing access control policy decisions. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 298–309. ACM.

OpenStack (2013a). The nova.openstack.common.policy module. http://docs.openstack.org/developer/nova/api/nova.openstack.common.policy.html.

OpenStack (2013b). Openstack: The open source cloud operating system. http://www.openstack.org/software/.

Ramli, C. D. P. K., Nielson, H. R., and Nielson, F. (2013). The logic of xacml. *Science of Computer Programming*.

Schubert, L. and Jeffery, K. (2012). Advances in clouds. Technical report, European Union, Tech. Rep.

Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42.

Wen, X., Gu, G., Li, Q., Gao, Y., and Zhang, X. (2012). Comparison of open-source cloud management platforms: Openstack and opennebula. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 2457–2461. IEEE.

WSO2 (2012). Balana xacml for authorization. http://xacmlinfo.org/category/balana/.