Towards a Software-Defined Security Framework for Supporting Distributed Cloud

Maxime Compastié, Rémi Badonnel, Olivier Festor, Ruan He, Mohamed Kassi-Lahlou

Towards a Software-Defined Security Framework for Supporting Distributed Cloud

Maxime Compastié[1,2], Rémi Badonnel[1], Olivier Festor[1], Ruan He[2], Mohamed Kassi-Lahlou[2]

[1] LORIA - INRIA, Campus Scientifique, 54600 Villers, France
maxime.compastie@loria.fr, remi.badonnel@loria.fr, olivier.festor@loria.fr
[2] Orange Labs, 44 Avenue de la République, 92320 Chatillon, France
ruan.he@orange.com, mohamed.kassilahlou@orange.com

Abstract. Cloud computing provides new facilities for building elaborated services hosted through various infrastructures over the Internet. In the meantime, these ones pose new important challenges in terms of security due to their intrinsic nature. We propose in this paper to detail a software-defined security framework supporting the protection of these services, in the context of distributed cloud. These ones require security mechanisms able to cope with their multi-tenancy and multi-cloud properties. The foundations of this framework rely on the software-defined logic to express and propagate security policies to the considered cloud resources, and on the autonomic paradigm to dynamically configure and adjust these mechanisms to distributed cloud constraints. In particular, we describe the main components and protocols of this software-defined security framework, evaluate this one and discuss implementation considerations, through the analysis of different realistic scenarios.

1 Introduction

The cloud computing architectural model permits to build elaborated services and applications based on multiple computing resources, such as virtual machines, network devices, software components, themselves provided as a service that can be easily deployed through the Internet. Based on the NIST Institute [1] definition, this model is mainly characterized by the following features: *on-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity*, and *measured service*. It supports an as a service scheme that permits a transparent access to resources and the outsourcing of part of the management to the cloud provider. This separation enables optimizing the resource allocation and usage, but may also introduce management complexity due to its distributed nature. In particular, the cloud infrastructure and its applications may typically be divided into isolated sets of resources called *tenants*, corresponding to different ownerships and requirements, defining the *multi-tenancy* property. Another property comes to the facts that the resources may be distributed among several infrastructures, as each of them may be specialized in a dedicated processing.

Distributed cloud can be defined by the conjunction of the *multi-tenancy* and *multi-cloud* properties. In this context, security management has become a major challenge. The dynamics of cloud infrastructures induced by their *on-demand self-service*, *rapid elasticity* and distribution has outrun traditional security management, while the ubiquity and high availability of cloud resources make them attractive targets for attackers [2].

Exploiting autonomic and programmability mechanisms opens new perspectives for enabling such a security management. Autonomic computing permits to address the scalability issues induced by large and distributed cloud infrastructure resources, by delegating part of the management tasks to the environment itself. In our context, this concerns more particularly the management tasks related to self-protection and self-configuration, and aims at maintaining the security level of a distributed cloud and its services in an adequate manner with the security threats, based on the activation or deactivation of available countermeasures in a proactive and/or reactive manner. In addition, network programmability has already shown its advantage for software-defined networking by separating the network infrastructure into two separate planes, i.e. the data plane and the control plane, and contributing to its dynamic configuration and adaptation. Similarly, there is an important need for supporting *software-defined security* in the context of distributed cloud.

We have already highlighted the benefits of software-defined security for distributed cloud environments in [3]. We detail in this paper the different components and protocols of our security framework relying on software-defined and autonomic paradigms, and provides a critical analysis of the proposed solution considering a set of validation scenarios based on a realistic use case. The framework permits to specify security policies, and enables their autonomic enforcement in a multi-tenant and multi-cloud environment. Security mechanisms should be dynamically aligned and adjusted based on changes that may occur in the distributed cloud. The rest of this paper is organized as follow: Section II gives an overview of existing work related to our software-defined security solution. The proposed framework, its components and their interactions are detailed in Section III. We evaluate it and give a critical analysis as well as implementation considerations in Section IV. Finally, we conclude the paper and point out future research efforts in Section V.

2 Related Work

The security of cloud infrastructures has already been largely explored in the literature. In particular [4] highlights several challenges related to policy-based security management, such as the specification of a cloud security policy, the assurance of the security decisions, as well as the the certification of security components in that context. In the same manner, the *TCloud* framework [5] proposes to enforce a security policy with a hardened cloud stack. This one provides infrastructure-level and platform-level security components, that might be compatible with multi-cloud environments, with a hardened build of Open-

Stack environments. However, these solutions do not specifically address self-configuration mechanisms, nor the management issues generated by multi-cloud and multi-tenancy properties. The *Iceman* architecture [6] enables secure federated inter-cloud identity management. The author of [7] proposes a cloud management framework able to deal with multi-tenancy, but this one is limited to access control policies and cannot support other security mechanisms. The proposed architecture is independent from the available security mechanisms and addresses their self-configuration in a distributed cloud.

In the area of programmability, *software-defined networking (SDN)* permits to separate the *control plane* making decisions about where the traffic should be sent from the *data plane* forwarding of packets. This paradigm enables a dynamic and adaptive policy enforcement. It may also serve as a support for chaining security functions. For instance, the *Flowtags* framework described in [8] enables the integration of middleboxes whose composition is supported by SDN controller. [9] proposes a framework for enforcing a network security policy through a set of middleboxes. But, this solution only considers middleboxes for instantiating security mechanisms. We have also shown in [10] how to exploit the SDN paradigm to build a chain of security functions, including intrusion detection systems and firewalls, to protect smart devices. IETF is also working on SDN-based security services using interface to network security functions [11]. Such approaches take advantage of SDN with respect to security policy enforcement.

Important efforts have also focused on the verification of security chains. For instance, *VeriCon* [12] combines a language for specifying SDN policies with an approach to check whether a policy verifies invariants expressed in predicate logic. In the same manner, *FlowChecker* [13] represents the network as a binary decision diagram (BDD), whereas properties are expressed in computation tree logic (CTL). However, the model based on BDDs requires a certain expertise of formal methods, which cannot be generally expected from network operators. In our context, we are focusing on a software-defined security framework to protect distributed cloud, in line with software-defined networking, but not limited to network enforcement considerations.

The autonomic computing paradigm gives a framework for self-management activities, and relies on several main areas: self-configuration, self-optimization, self-protection and self-healing [14]. Although it does not bring a formal distributed cloud support, it may introduce the negotiation among independent components. This approach may deal with exhaustive enforcement issues, as autonomic components can continuously enforce the security policy and adapt to the changes in their action perimeters. Even if the two previous paradigms do not directly deal with distributed cloud issues, they provide important building blocks for supporting security policy enforcement and defining a security management architecture in that context and in our framework.

With respect to security policies, the OASIS consortium introduces two standardized languages: *XACML (eXtensible Access Control Markup Language)* for representing and exchanging security policies [15] and SAML (*Security Assertion Markup Language*) for specifying security statements [16]. However, they

do not handle any modifications of cloud policies nor its evolution propagation to enforcers. This approach remains relevant as the XACML defines modular components for security enforcement. Besides, an architecture and use-cases featuring XACML and SAML in distributed environment have been detailed in [17]. The latter validates the usability of XACML in distributed systems, underlining some limitations such as the need for a high granularity of sub-policies and the difficulty of maintaining an encoded security policy. The languages and formats introduced by SCAP protocol constitutes also an interesting support, as they cover many complementary specifications, such as vulnerability descriptions and scorings, that are exploitable for automating security in distributed cloud [18]. These standards are usable in our solution.

In accordance with [3] where we give the basement of our software-defined security approach, the autonomic paradigm is tied to endorse the continuous security policy enforcement able to cope with the changes occurring on the security policy, the tenant configuration and the protected resource state. We extend our previous work by detailing each components and protocols supporting our framework, and giving a critical analysis and implementation considerations based on realistic scenarios.

3 Software-Defined Security Framework

We propose a software-defined security (SDSec) framework for protecting distributed cloud. These one is composed of two main layers, called respectively security control plane and security data plane (as depicted on Figure 1). It relies on a *software-defined* scheme to provide a global security policy specification interface and exploit autonomic mechanisms within distributed cloud infrastructures to enable cloud resources to be dynamically and exhaustively protected according to this policy. More precisely, it first consists in a *global security policy (GSP)* which formally defines at a business level the security objectives of cloud resources and is then translated into several *tenant-level security policies (TLSP)*, providing security statements that must be verified by specified resources at the tenant level within the distributed cloud.

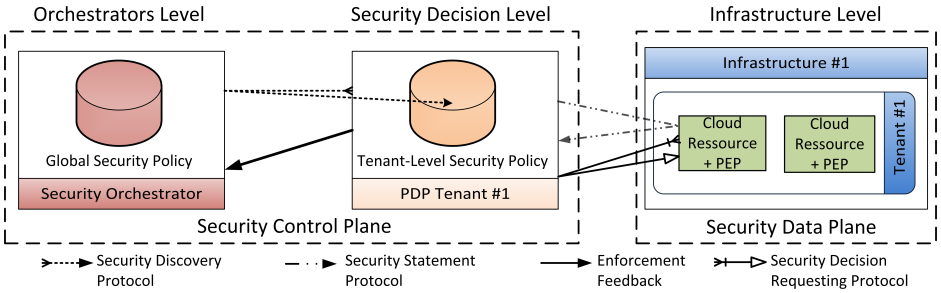

Fig. 1. SDSec framework in a single-infrastructure single-tenant scenario

These *security statements* are then enforced on cloud resources, i.e. virtualized infrastructures and software products. They aim at altering the behavior of these components and protecting them based on countermeasures available with distributed cloud. This application can be active if its application requires negotiation with a decisional instance. The enforcement should be performed dynamically, more precisely in an adaptive (it adapts to any change in the enforced resource state or in the infrastructure), automatic (no operator interventions are needed for it), and self-configured manner (policy decisions for it are automatically made according to several criteria including the security requirements).

The components of the framework part of the security control plane include the *security orchestrator* hosting a GSP specified by the system administrator, exposing through a dedicated interface the TLSPs, and receiving enforcement feedbacks from the *policy decision point* (PDP) to adapt them. These interactions are supported by the *security discovery protocol* enabling the PDP to identify the security orchestrator and fetch its security policy. The components part of the security data plane correspond to the *policy enforcement points* (PEP) executing the security statements (using the *security statement protocol*) and dedicated to the policy enforcement on one type of cloud resources. It may also solicit the PDP for taking a needed security decision for an active enforcement (using the *security decision requesting protocol*).

This framework follows a software-defined paradigm to specify security constraints, and relies on self-configuration mechanisms to enable a dynamic and local management. Self-configuration enables a lower coupling with respect to orchestration. Instead of the regular orchestration model addressing requests and expecting feedbacks, the security orchestrator adopts a passive approach by exposing security requirements, and letting the PDP to interpret them, according to their enforcement contexts. In addition, the framework has been designed to fit with distributed cloud constraints, in particular the following ones:

- *multi-tenancy*, corresponding to the characteristic for a cloud infrastructure to be subdivided into different sets of isolated cloud resources called *tenants*. With that isolation comes the need of regulated access control between each tenant of the infrastructure,
- *multi-cloud*, corresponding to the capability for cloud infrastructures to collaborate to enable communications and common treatments on their resources. With a security-oriented point of view, these treatments come with a security coordination over potentially heterogeneous infrastructures.

In doing so, we detail the role and functioning of its different components, considering a multi-cloud and multi-tenant context, as depicted on Figure 2. This figure makes the assumption that each PDP is dedicated to a tenant, which is a simple interpretation of software-defined security in this multi-tenant context. We consider the existence of a cloud orchestrator in charge of managing cloud resources. Even though this component is not meant to be a part of the proposed security framework, its supposed existence allows taking into account the changes on cloud resources, which can be done manually by a system administrator or automatically by one or several potential orchestrators.

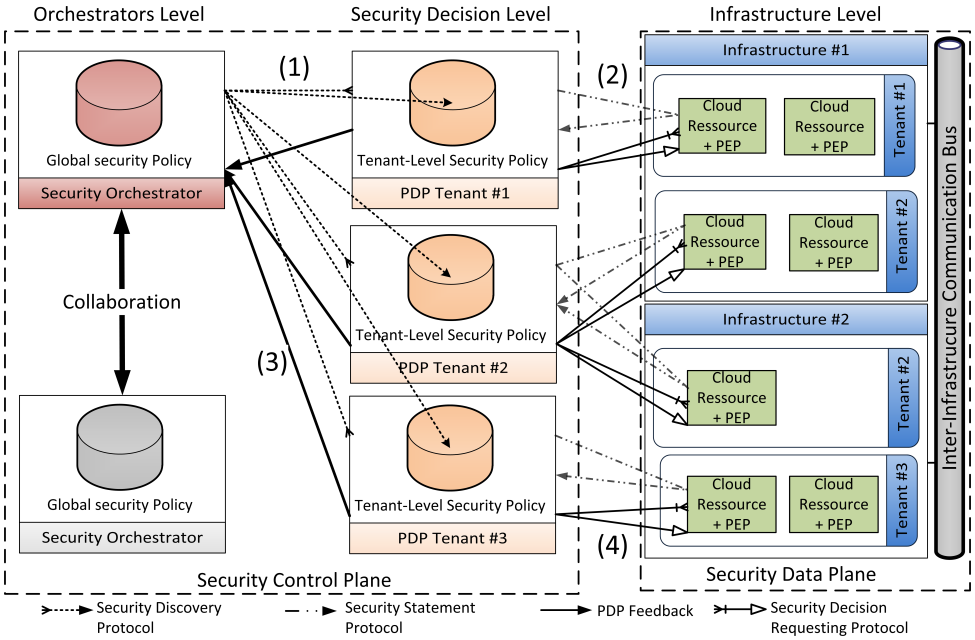

Fig. 2. SDSec framework interacting with a cloud orchestrator, in a multi-cloud multi-tenant scenario. (1) accounts for the TLSP fetching, (2) for the security statement, (3) for the enforcement feedback and (4) for the policy decision request.

3.1 Security Orchestrator

Amongst the framework components, the security orchestrator is responsible for the management of the GSP, its interpretation (TLSPs) and distribution. This policy is meant to be enforced on the distributed cloud, and so, on multiple collaborating cloud infrastructures with different tenants. The interpretation is influenced by feedbacks provided by the enforcement. In line with the XACML terminology [15], the security orchestrator can be seen as a Policy Administration Point (PAP) allowing the storage of the global policy and generating TLSPs. The changes operated on the global security policy must be propagated to the whole enforcement perimeter. Contrary to the cloud orchestrator, the security orchestrator is not meant to manage cloud resources. Consequently, the instantiation, the removal or the reconfiguration of cloud resources is not endorsed by the security orchestrator.

However, this highlights the need for the security orchestrator and the cloud orchestrator to collaborate. For instance, the security orchestrator requires to be noticed in case of deployments of new cloud resources, in order to enforce the security policy on them. In the same manner, the cloud orchestrator must remove a cloud resource and reconfigure its workflow, when the security orchestrator requests its removal for security purpose. This collaboration is modeled

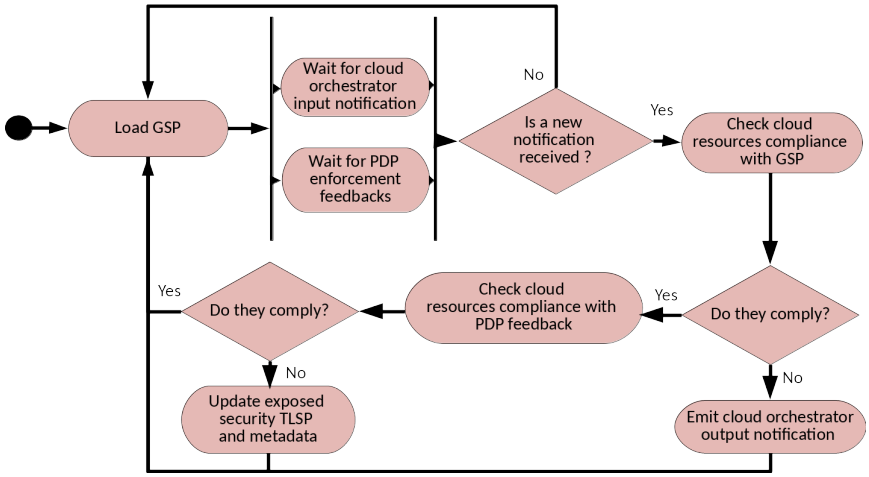

Fig. 3. Overview of the Security Orchestrator Activity Diagram

on Figure 2 by the double arrow between the two orchestrators on the leftmost plane. An overview of the activity diagram of the security orchestrator is given on Figure 3. The orchestrator does not push the TLSPs to the PDPs for privacy purposes, the multi-tenancy property implying the isolation of tenants amongst each others and with the cloud administrator. These TLSPs must be attached to meta-datas to enable PDPs to fetch only the policies they are concerned to, by discriminating each TLSP according to enforcement context criteria. The policy must be exposed through a dedicated interface accepting incoming connections from PDPs (with the use of the security discovery protocol). Another interface assumes the reception of all PDP enforcement feedbacks. The determination of the exposed TLSPs (as well as the notification sent to the cloud orchestrator) is correlated to the GSP, the PDP feedbacks and the notifications potentially sent by cloud orchestrator.

3.2 Policy Decision Points

The Policy Decision Point (PDPs) play a central role in this software-defined security framework, serving as intermediates between the security orchestrator and the PEPs enforcing policies on resources. More precisely, the PDPs are in charge of fetching and hosting the TLSPs using the policy security discovery protocol, and locating their PEPs by invoking the enforcement discovery protocol. Moreover, they support the interactions with PEPs by collecting their feedbacks and responding to security requests in according to the hosted TLSPs. According to the XACML terminology [15], the PDPs assume different roles: the role of PDPs providing authorization decisions, but also the role of PAPs with respect to TLSPs, and the role of PRPs (Policy Retrieval Points). PDPs must take into account external informations modulating the interpretation of their TLSPs. For

instance, time-regulated access control policy requires an access point to a system clock, as this parameter cannot be generalized to all PDPs of the enforced perimeter, it is necessary that the PDP proposes an extensible interface able to communicate with third-party security information providers. In the XACML terminology [15], these third-party resources are assimilated to Policy Information Points (PIPs). Besides, the PDPs maintain several meta-datas describing their decisional capabilities, which are directly related to their enforcement context. These meta-datas are important for the tenant-level security policy discovery. Consequently, the security statements intended to the PEPs is directly related to the stored TLSPs, modulated by the preceding feedbacks generated by the PEPs, and eventually, by the PIP contents.

3.3 Policy Enforcement Points

The Policy Enforcement Points (PEPs) are in charge of the enforcement of TLSPs for a dedicated cloud resource. More precisely, a cloud resource refers to an instantiated resource on a cloud infrastructure (i.e. a virtual machine, a service, a set of files, a network function). The considered enforcement consists in (1) the control and modification of security parameters on the resource according to security statements and (2) the insertion of security event hooks to handle with state changes and prepare associated security decisional requests. Besides, these objectives correspond the ones defined by the XACML for PEPs. Consequently, the PEPs must expose an interface to the PDP for receiving security statements, and be able to contact the PDPs to return feedbacks (after the execution of a security statement or after an event hook) and to transmit a security decisional request. The configuration of security parameters is directly dependent on received security statements. The feedbacks are defined based on received security statements, states of considered security parameters and event hook states. Security decisional requests are emitted by PEPs based on event hook states.

3.4 Interactions Amongst Components

The interactions amongst the software-defined security framework components is supported by different protocols. The *Security Policy Discovery Protocol* is a discovery protocol invoked by a PDP to discover the security orchestrator and fetch a TLSP. The discovery process takes as inputs the PDP meta-data, and gives back the required TLSPs. Because of the criticality of this protocol, its specification must integrate technical measures to protect the integrity of information and remain tamper-proof. In addition, the *Enforcement Discovery Protocol* enables a PDP to discover available PEPs in its enforcement perimeter, and so, to quantify its enforcement capabilities. More precisely, these capabilities are expressed by available PEPs through their enforcement meta-datas, and brought back to the PDP which determines their potential contributions to the security enforcement. To prevent security policy information leaks to an intruder or to prevent an intruder to weaken the security enforcement by providing false

security assessment feedbacks, the protocol must enable the PDP to verify the authenticity of the discovered PEPs. The *Security Statement Protocol* enables the PDPs to generate security statements, and send them to PEPs in their enforcement perimeters. The feedback must be emitted asynchronously, in case of enforcement statement execution time-out. Hence, to provide a reactive enforcement, it must be able to emit new feedbacks, when a correctness of a previously executed security statement changes. Finally, the *Security Decision Requesting Protocol* offers to the framework its dynamic enforcement properties. Indeed, this protocol enables the PEPs to solicit the PDPs for handling a security decision. This security decision request occurs when a security hook of a PEP is triggered and verification of the issued security statement cannot be handled by the PEP itself. The security of these different protocols is out of the scope of this paper, but is of course a mandatory to guarantee the security of the whole framework.

4 Framework Evaluation

In order to analyze and validate our proposed framework, we have confronted it to a set of scenarios based on a realistic use case, corresponding to a Cloud Service Provider (CSP) proposing a *Platform-as-a-Service (PaaS)* solution to customers, based on world-wide infrastructures. The multi-tenancy corresponds to the use of the same infrastructure by several independent customers, while the multi-cloud property comes from the world-wide location of cloud infrastructures. To protect its solution, the CSP enforces a security policy on its own infrastructure, and on its client instantiated cloud resources. In that context, we will consider the case of a customer, deploying two virtual machines (VM) for hosting two web applications: one for the European version of his application and one for the American one.

4.1 Validation Scenarios

The scenarios make the following assumptions: the CSP has implemented every business process in the cloud orchestrator, each customer request is endorsed by the cloud orchestrator, the customers are unable to remove the PEPs of its cloud resources, no connection error occurs between PEPs and PDPs, the deployment of software stacks in the PaaS resources is governed by the cloud orchestrator and embeds the related PEPs, the cloud resource manager comes with its own PEP which is managed by the tenant PDP. We have analyzed a set of five scenarios: the deployment of a new system instance for a customer, the security policy update by a CSP, a DDoS attack to an instantiated cloud VM, an inter-resource access request, and the removal of a VM instance.

Resource Instantiation Scenario. The customer sets up a dedicated server associated to his tenant to synchronize and back up the informations of the instances of his web application. The virtual machines hosting its web applications are Linux-powered, embeds a SSH server for administrative tasks and a

web server. The chosen technical solution consists in using a SQL server and a FTP server in a dedicated VM stored in the European infrastructure, which will accept connections from the two web application servers. The cloud orchestrator processes the deployment of these two services with their respective PEPs and notifies the security orchestrator. As FTP and SQL are newly deployed services in the tenant, the security orchestrator assumes that the TLSP of the tenant PDP is not adapted anymore, and modifies the exposed TLSP to this PDP. The PDP discovers the two new PEPs, fetches the newly available TLSPs from the security orchestrator, and sends the security statements to the PEPs. Finally, the PDP transmits a positive enforcement feedback to the security orchestrator. This prevents the security orchestrator to request the cloud orchestrator to take counter-measures against the tenant.

Security Policy Update Scenario. The CSP security administrator enforces the security of its infrastructure, by restricting the access of critical services only to the local network and the CSP VPN. The criticality of a service is not defined in the GSP, but is delegated to the PDP. After the update of the GSP, the PDP of each tenant detects and collects updated TLSPs. All the PDPs interpret their TLSPs into security statements restricting the critical service access.The PDP associated to the consider customer has deduced that all SSH and SQL servers were critical. It requests their PEPs to restrict their access and notifies the security orchestrator of the effective enforcement. If one of the PDPs receives a PEP negative feedback and has no other counter-measure to apply, it notifies the security orchestrator which will in turn notify the cloud orchestrator to disable vulnerable services.

Resource Evolution Scenario. The virtual machine in charge of the European version of the web application hosting is targeted by a Distributed Denial of Service (DDoS) attack. An alert is generated by the PEP to the PDP, indicating the resource consumption is higher than a threshold (initially specified by the PDP). Consequently, the PDP activates a counter-measure by temporarily increasing the resources allocated to the customer. As this counter-measure is not efficient, the PDP informs the security orchestrator of its inability to enforce the GSP. The security orchestrator then relies on the security enforcement stack dedicated to the network infrastructure to perform investigation and block attacker IP addresses. It requests the tenant PDP to switch the affected VM into a fail-safe mode. Once the DDoS attack has been countered, the security orchestrator reverts back the TLSP exposed to the customer in order to restore the attacked VM state.

Access Request Scenario. The cloud service provider has defined in its GSP that the used credentials for the connections amongst cloud resources have a limited lifetime, and have to be regularly changed. The verification of the validity is committed by the PDP using a third-party module. Meanwhile, the client has

set-up an automatic back-up process between the backup server hosted in the European infrastructure and the production server located in the USA, by using SQL and FTP transactions: the production server authenticates to the backup server using a dedicated password. When the production server connects to the back-up server, the connection attempts trigger the connection hooks of PEPs related to SQL and FTP servers. Both of them block temporarily the connection attempts, and make decision requests to the PDP, providing hashes of used credentials. As the TLSP imposes the verification of the credential lifetime, it uses its third party module to check it. As this module has no precedent records of hashes, it concludes that the transmitted credentials are newly created and are allowed to be used. The PDP responses to both security decision requests are positive, and incoming connections are authorized by respective PEPs.

Resource Removal Scenario. The client wants to update the virtual machine supporting the American web application by proceeding to a fresh installation. To meet this objective, the client wants to completely remove it and reconfigure a new virtual machine. He uses the cloud orchestrator to remove this virtual machine, which is notified to the security orchestrator. The security orchestrator updates its GSP, to take into account the removal of the cloud resource and checks its consequences on the enforcement: the TLSP is updated. The PDP of the customer fetches the new TLSP, and stores it. Through the Business Orchestrator, the security orchestrator starts deallocating resources to the American VM and the PEP addresses a security decisional request to its PDP for allowing the removal. According to its TLSP, the PDP grants the request. The PEP lets the cloud orchestrator to complete the resource removal.

This analysis shows that all the presented scenarios can be addressed by our proposed software-defined security framework. However, some limitations with respect to the considered use case should be highlighted. First, the use case has dealt with a GSP set by one security orchestrator. The case of multiple security administrators, with different enforcement parameters is an addressable issue as well although we still can abstract it through the single security orchestrator case. Second, the use case assumes that one PDP is allocated to one tenant, corresponding to one customer. This is however only one possible interpretation of the multi-tenancy notion, but other ones would have made the use case unnecessarily more complex.

4.2 Implementation Considerations

After reviewing validation scenarios to evaluate the consistency of our framework, we are discussing in this subsection implementation considerations.

Cloud Environment. Before considering a software-defined security stack for our framework, we focus on the environment and the resources we want to en-

force. We address distributed cloud infrastructure security. The retained technical solution should be a proven solution in the multi-tenancy area as well as the multi-cloud one. Moreover, as arisen in the third validation scenario, some of the countermeasures are likely to rely on infrastructure configuration. This highlights the need for an extensible cloud stack embedding add-on mechanisms. In both cases, the OpenStack cloud suite is an attractive solution, as it supports multi-tenancy through the users and region management, and the main components of this suite provide plug-in managers.

Considering the orchestration, we have to distinguish the need of a security orchestrator based on a security policy ruling, and a regular cloud one whose actions are driven by customer solicitation or CSP management tasks. The first one will be further analyzed in the next subsection. The second has no specific security expectation except its capability to handle cloud orchestration notifications, and reciprocally emits notification to it. These two requirements are related to common orchestrator features as both are linkable to basic messaging between cloud appliances, each one issuing a request to the other and waiting for a feedback. Therefore, no more prerequisite other than distributed cloud support is expected from them.

In the cloud resource area, our framework is designed to be resource agnostic in the sense that the PEPs are the only agents of the architecture depending on cloud resources. Their interactions are based on resources programmability, inspection and event handling. Those common features could arise particular interests the more they are related to dynamic and complex resources. In this context, virtual machines operating systems and applications are well-suited for exploring this kind of enforcement, but cannot be generalized as the only type of resources to be protected. Besides, their nature directly influences the way PEPs are implemented: an executable cloud resource opens the debate about whether the PEP should be totally, partially or not at all included in it while a non-executable one excludes it.

Framework Components. Considerations are also raised by the implementation of the framework itself. The security orchestration is the component responsible for the coordination of the PDPs with each others and the cloud infrastructure (through the cloud orchestrator). As such, it is a highly critical single point of failure in charge of supervising several tenants and infrastructures. Such a criticality raises technical issues about redundancy or distribution among the infrastructure, but also policy concerns such as handling enforcement state transition due to GSP modification: if the modification process is not properly handled, as cloud tenant-level security policy and cloud-resource statement are not instantly propagated (due to network or processing overhead), we can conceive that a subset of resources of the cloud infrastructure managed by the security orchestrator to be trapped into a inconsistent security state. This eventuality must urge the orchestrator to check the consistency of intermediate enforcement stated, at the infrastructure level (resource enforcement state can

conflicts) and at the policy-decision level (concurrent low-level security policy can as well conflicts).

Moreover, the privacy concerns is risen with the PDP. Indeed, it can access all the PEPs it is in charge of, and any data leak may allow an attacker to collect resource data or metadata. Incidentally, the confidentiality of the communication between PEPs and PDPs is as critical as the isolation between PDPs is. This statement decides the question of the relation between PDPs and tenants. To enforce a correct isolation between PDPs, it is necessary that none of them address several tenants. Otherwise, one tenant could compromise a multi-tenant PDP, and use-it to fetch data from the other tenant resources.

Finally, the variability of the resources this security framework addresses the enforcement leads to the question of PEP design. Building one PEP for each type of resource to enforce a TLSP in a cloud is not a sustainable approach as the workload for a sufficient enforcement coverage would go too far. Thus, we should consider a more generic approach allowing an automatic adaptation to cloud resource. A *model-driven* design and instantiation of PEP is a interesting response element as the core logic of the PEP could be specified in the model, before being compiled and adapted on-the-fly to the specificities of the resource to protect. Moreover, such an approach could eventually take advantage of the cloud resource build environment: if this PEP design and integration process is able to extract the required information from cloud resources being constructed, it would lead to an automatic and adaptive design of PEPs tied to cloud resource dynamics.

5 Conclusions

We have proposed in this paper a software-defined security framework for protecting distributed cloud. It relies on the programmability of software-defined security, and exploits the autonomic paradigm for addressing the constraints induced by multi-tenancy and multi-cloud properties. We have detailed the different components of this framework, including a security orchestrator, policy decision points (PDPs) and policy enforcement points (PEPs) interacting according to a dedicated set of protocols. Based on the specification of a security policy, our framework supports the dynamic configuration of security mechanisms to adjust to contextual changes, based on available resources and counter-measures. Autonomic methods also enable a lower coupling with respect to orchestration. We have evaluated the proposed solution and discussed implementation considerations, through a set of validation scenarios corresponding to a realistic use case. The proposed solution has raised several challenges with respect to the design of the considered components, and the specification of security policies in a multi-cloud and multi-tenant context. The PEPs will apply model-driven scheme to facilitate the interoperability of heterogeneous enforcements. In the longer term, the security policy specification of distributed cloud, and the dedicated access mode will be investigated to complement the security orchestration.

References

1. Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. 2011.
2. Cloud Security Alliance. Top Threats to Cloud Computing v1. *White Paper*, 2010.
3. M. Compastié, R. Badonnel, O. Festor, R. He, and M. Kassi-Lahlou. A Software-Defined Security Strategy for Supporting Autonomic Security Enforcement in Distributed Cloud. In *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), PhD Track, Short Paper*, pages 464–467, Dec 2016.
4. Adrian Waller, Ian Sandy, Eamonn Power, Efthimia Aivaloglou, Charalampos Skianis, Antonio Muñoz, and Antonio Maña. Policy based management for Security in Cloud Computing. In *FTRA International Conference on Secure and Trust Computing, Data Management, and Application*, pages 130–137. Springer, 2011.
5. Alysson Bessani, Leucio A Cutillo, Gianluca Ramunno, Norbert Schirmer, and Paolo Smiraglia. The TClouds Platform: From the Concept to the Implementation of Bench. Scenarios. *ACM SIGOPS Operating Systems Review*, 48(2):13–22, 2014.
6. G. Dreo, M. Golling, W. Hommel, and F. Tietze. ICEMAN: An Architecture for Secure Federated Inter-cloud Identity Management. In *Proc. of the IFIP/IEEE Int. Symposium on Integrated Network Management (IM 2013)*, May 2013.
7. Olubisi Atinuke Runsewe. A Policy-Based Management Framework for Cloud Computing Security. Master's thesis, University of Ottawa, 2014.
8. Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in Software Defined Networking*, pages 19–24. ACM, 2013.
9. Tommy Koorevaar. Dynamic Enforcement of Security Policies in Multi-Tenant Cloud Networks. Master's thesis, École Polytechnique de Montréal, 2012.
10. Gaëtan Hurel, Rémi Badonnel, Abdelkader Lahmadi, and Olivier Festor. Behavioral and Dynamic Security Functions Chaining for Android Devices. In *Proceedings of the 11th IFIP/IEEE/In Assoc. with ACM SIGCOMM International Conference on Network and Service Management (CNSM'15)*, 2015.
11. J. Park J. Jeong, H. Kim. Software-Defined Networking Based Security Services using Interface to Network Security Functions, October 2015.
12. Thomas Ball and All. Vericon: Towards Verifying Controller Programs in Software-Defined Networks. In *Proc. 35th ACM SIGPLAN Intl. Conf. Programming Language Design (PLDI'14)*, pages 282–293, Edinburgh, UK, 2014.
13. Ehab Al-Shaer and Saeed Al-Haj. FlowChecker, Configuration Analysis and Verification of Federated OpenFlow Infrastructures. In *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration (CCS'10)*, 2010.
14. Jeffrey O Kephart and David M Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.
15. Simon Godik, Tim Moses, A Anderson, B Parducci, C Adams, D Flinn, G Brose, H Lockhart, K Beznosov, M Kudo, et al. EXtensible Access Control Markup Language (XACML) version 1.0, 2003.
16. Eve Maler et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML). *OASIS, September*, 2003.
17. Jennifer Golbeck. Trust on the World Wide Web: a Survey. *Foundations and Trends in Web Science*, 1(2):131–197, 2006.
18. David Waltermire, Stephen Quinn, Karen Scarfone, and Adam Halbardier. The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP version 1.2. *NIST Special Publication*, 800:126, 2011.