mOSAIC-Based Intrusion Detection Framework for Cloud Computing

Massimo Ficco, Salvatore Venticinque, and Beniamino Di Martino

Department of Information Engineering
Second University of Naples (SUN)
Via Roma 29, 81031 Aversa, Italy
{massimo.ficco,salvatore.venticinque,beniamino.dimartino@unina}@unina2.it

Abstract. In recent years, with the growing popularity of Cloud Computing, security in Cloud has become an important issue. Cloud Computing paradigm represents an opportunity for users to reduce costs and increase efficiency providing an alternative way of using services. It represents both a technology for using computing infrastructures in a more efficient way and a business model for selling computing resources. The possibility of dynamically acquire and use resources and services on the base of a pay-per-use model, implies incredible flexibility in terms of management, which is otherwise often hard to address. On the other hand, because of this flexibility, Denial of Service attacks represent a serious danger, which can compromise performance and availability of services provided to final users. In this paper, a mOSAIC-based framework for providing distributed intrusion detection in Cloud Computing is proposed. It is an architectural framework that collects information at different Cloud architectural levels, using multiple security components, which are dynamically deployed as a distributed architecture. The proposed solution allows to monitor different attack symptoms on different Cloud architectural levels, which can be used to perform complex event correlation and diagnosis analysis of intrusion in the Cloud system.

Keywords: Cloud security, distributed intrusion detection, monitoring, mobile agents.

1 Introduction

Cloud Computing is an emerging paradigm that allows customers to obtain easily services according to a pay-per-use business model [1]. In general, Cloud service providers offer guarantees in terms of service availability and performance during a time period of hours and days. The provisioning contracts regulate the cost that customers have to pay for provided services and resources. On the other hand, the Cloud provider must pay a penalty if the customer's requirements are not satisfied. For example, the Cloud provider is liable to pay a penalty for service requests that are rejected due to resources unavailability; as the same manner, if the average service response time exceeds a fixed threshold. In order to support this model, the Cloud infrastructure has to continually adapt to changes

R. Meersman et al. (Eds.): OTM 2012, Part II, LNCS 7566, pp. 628–644, 2012.

of customer demands and operation conditions. For example, in order to prevent service availability violations may be required additional standby resources to handle a given number of failures, whereas to prevent performance violations may be required to scale up or move a virtual machine (VM) to another physical machine, if the current machine is overloaded.

Cloud applications, due to their openness to the Internet, are prone to cyber attacks, such as Denial of Service (DoS) and Distributed DoS (DDoS), which aim at reducing the service's availability and performance by exhausting the resources of the service's host system (including memory, processing resources, and network bandwidth) [2]. Such attacks have special effects in Cloud due to the pay-per-use business model described above, *i.e.*, they have direct effects on the application costs and not only on the performance perceived by users. Specifically, ensuring determined security levels is not easy in the Cloud. In order to face peak loads due to DDoS attacks, additional resources should be either already acquired, or dynamically acquired on demand. On the other hand, such resources are not free, and a cyber attack could make it economically prohibitive. Such a scenario can be called '*economic denial of sustainability*'. The inability of the Cloud service infrastructure to diagnose the causes of service performance degradation (*i.e.*, if it is due to either an attack or an overloading), can be considered as a security vulnerability, which can be exploited by attackers in order to exhaust all the Cloud resources allocated to satisfy the negotiated QoS. It is clear that the Service Level Agreement (SLA) availability and performance parameters alone is not enough to ensure the satisfaction of service delivery. Several works explore SLAs for security, and analyze security metrics in new paradigms like Cloud Computing [5, 6]. By incorporating security parameters in the resource requests can improve the QoS of the service being delivered [7]. These requests have profound implications in the security solution to be implemented and delivered to protect the provided Cloud services.

Although several distributed Intrusion Detection System (IDS) solutions has been proposed to face security aspects in large scale networks [8], their utilization in Cloud Computing is still a challenging task [9]. The security of applications and services provided in Cloud, against cyber attacks, is hard to achieve for complexity and heterogeneity of such systems. Moreover, the presence of different kinds of users lead to different security requirements. For example, an important issue for the Cloud user is the missing full control over the used infrastructure. They need to know if the service performance degradation is due to attacks against their application, as well as if the acquired resources and services have been compromised to penetrate other hosts. Furthermore, each user can require different types of information, sensors, configurations and thresholds to monitor their virtualized components. Therefore, Cloud provider have to contemplate the possibility that end users can fully control the resources. On the other hand, Cloud providers also want to detect attacks to both the virtualized components and the underlying infrastructure. These attacks can be performed by an external attacker, or by an malicious user or a virtualized component who may have been

compromised (*i.e.*, they need to know if their systems and infrastructure is being used by attackers to penetrate other victims).

In this paper, we propose an extensible IDS management framework, which can be offered to Cloud providers in order to implement an "hierarchical" correlation process for detection of cyber attacks to their Cloud, as well as to the customers in order to monitor their applications. The security framework consists of distributed security components, which can be configured and deployed by users. They allow to collect streams of information at different Cloud architectural levels: the infrastructure level, the platform level, and the application level. In general, the infrastructure level includes the VM and the related virtual networks. An example of infrastructure provider is Amazon EC2 [11]. The platform level includes the basic software hosted on the VM, *e.g.*, the operating system, the Application Programming Interface (API), the Java Virtual Machine. An example are the Azure Platform [12] and Google App Engine Engine [13]. Finally, the application layer includes other software running in the Cloud, such as a Web application. Google is the main service provider example of this level.

The presented framework is based on the mOSAIC solution [10], which offers open-source APIs and a Software Platform that enable the development, the deployment and the execution of Cloud applications on a federation of Cloud Providers. In this paper, mOSAIC represents a programming paradigm and a technology, which can be adopted to develop a distributed IDS in Cloud. It allows to develop autonomous and proactive security agents able to migrate dynamically on the VMs together with their code and the execution status. The agents implement context-aware software components, that appropriately configured, can be enabled to infer security information at infrastructure level. The Software Platform provides functionalities to control and monitor software components at platform level. The mOSAIC APIs enable developer to implement Cloud applications that offer specific security features at application level.

A specific interface and APIs have been defined to implement security management components, which provide flexible integration of various security sensors. In particular, they allow to acquire and correlate security information inferred by the different security components deployed on different Cloud architectural levels. Moreover, they can be installed and used by Cloud providers and customers on their local machine, in order to remotely manage, configure and monitor the Cloud systems. The Intrusion Detection Message Exchange Format (IDMEF) standard is used to represent and exchange the security information [14] among the different entities defined in the architecture. Finally, a prototype, which implements the proposed architecture, is presented in the paper as proof-of-concept.

The rest of the paper is organized as follows: Section 2 presents the related work in the field of IDS management in Cloud. Section 3 describes a view on the security requirements in Cloud. The proposed IDS architecture is presented in Section 4. Section 5 describes the implementation of the architecture by means of the mOSAIC framework is described in Section 4. Section 6 presents a short summary and some future works.

2 Related Work

Several research works that propose intrusion detection solutions have been presented [15–19]. They face security problems involved with the usage of public and private Cloud infrastructures. In the following, some countermeasures proposed in literature are described.

Cheng et *al*. [20] present a distributed IDS architecture for Cloud. They propose to separate the IDS for each user, from the actual component being monitored. In particular, each virtual component should be secured by a separated IDS sensor, which is responsible for one virtual machine. The Cloud user is responsible to configure and manage the IDS running on its virtual machine. Additionally, to correlate and analyze alerts generated on multiple distributed sensors deployed in the Cloud, a central management unit is provided. It can be also used by user to configure the IDSs. The Cloud provider is responsible to enable different IDS and provide the possibility to connect them to a virtual component.

A similar solution has been proposed by Chi-Chun et *al*. [21]. They propose a federation defense approach, in which the IDSs are deployed in each Cloud computing region. IDSs cooperate with each other by exchanging alerts, and implement a judgment criterion to evaluate the trustworthiness of these alerts. The IDS conceptual architecture is composed by several conceptual entities, which have to be designed to perform the following activities: intrusion detection, alert clustering, threshold computation and comparison, intrusion response and blocking, and cooperative operations. A cooperative component is used to exchange alert messages from other IDSs. Majority vote method is used for judging the trustworthiness of an alert.

Park et *al*. [22] propose a multi-level IDS and log management method based on consumer behavior for applying IDS effectively to Cloud Computing system. The multi-level IDS applies differentiated levels of security strength to users according to the degree of their trustworthiness. On the base of risk level policies, risk points are assigned to user in proportion to risk of anomaly behaviors. User are judged by their previous personal usage history, and assigned virtual components with the security level derived by the judgment. The virtual component to assign to the user may change according to anomaly behavior level of the user, and migration can occur. Moreover, IDS sensors that have to be installed on the assigned user guest OS image, are choose on the base of the security level correspondent to the user's anomaly level.

Rak et *al*. [23] propose *Infrastructure as Service* (IaaS) enriched with ad-hoc solutions for protecting the delivered resources against a set of security attacks. The goal is to enable the customer to negotiate with the provider the level of security offered, so that the service acquired (for example a Web server) will be protected against a given set of attacks. Customer pays for the additional security service, but he/she is granted that the services is enabled to tolerate a specific set of DoSs attacks (*i.e*., continues to work even under attack), and the additional load generated is not charged. The proposed architecture is composed of two subsystems, with distinct properties. The first subsystem is an Application

System VM that hosts the application to protect, whereas the second subsystem, named ITmOS, is the VM that hosts the guest OS image to which are added Intrusion Tolerant mechanisms. The two subsystems are connected through a secure channel isolated from other connections. The interaction of the application with the outside world is done only through the network, using a proxy hosted on the ITmOS VM. A VM monitor monitors the Application System VM. It is used to collect information on system resources consumption (including CPU, memory, disk). An intrusion detector module collects data from the proxy and alerts a decision engine component whether an anomalous behavior is observed. The decision engine is a centralized engine, which receives and correlates security data. It determines whether the monitored data are malicious behaviors, and what are the effects on the monitored subsystem. It is responsible to identify the best reaction to take, in order to mitigate the attack effects on the target application. In particular, it analyzes the received data and performs the reactions by the proxy in response to the attacks, filtering messages to the guest system as needed.

3 Cloud User Requirements

The different kinds of users and the Cloud heterogeneity in terms of architectural levels lend to different security requirements.

In general, Cloud providers offer specific services in the Cloud, whereas Cloud users consume these services from the providers and have to pay based on consumed resources.

Therefore, the Cloud customers need mechanisms to monitor attacks against their applications and services that could be offered to final users (*e.g.*, SQL injection, XSS attacks, DoS attacks to the Web applications). Moreover, they needs to know if their services and virtual hosts are used to attack other victims. Additionally, each customer should be enabled to manage and specify personalized rules and thresholds of its security sensors for each components being monitored. Furthermore, in order to guarantee that the monitoring works properly when the virtual host or the application is compromised by an attack, customers could require that the IDS components used to collect and analyze security data acquired by security sensors in the Cloud, are separated from the component being monitored.

The Cloud provider is responsible for ensuring the quality of the offered services in terms of the security. Therefore, the Cloud providers need to detect attacks on their Cloud infrastructure, as well as attacks to the services offered to their customers, including DDoS, Buffer Overflow, IP spuffing. Moreover, the providers need to know if their infrastructure is being used by attackers to compromise other victims. For example, an attacker could compromise several hosts of Cloud in order to perform a coordinate attack against a single target, thereby causing denial of service for users of the targeted system, which may affect the Cloud provider reputation.

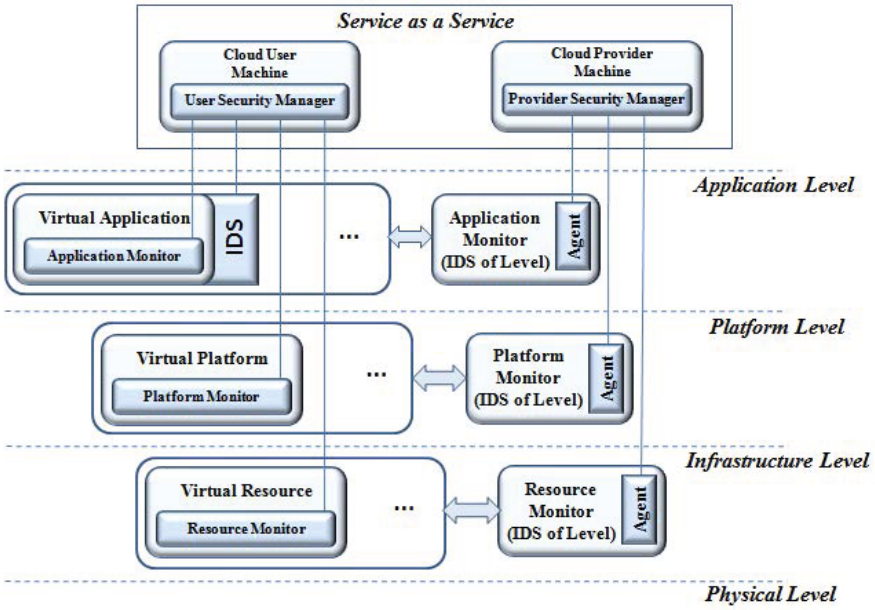

Fig. 1. IDS Architecture in Cloud

4 The Intrusion Detection Architectural Framework

In Figure 1 the architecture of the proposed intrusion detection framework is shown. It is a distributed architecture that consists of different security components deployed on the Provider's machines, as well as on the machines of customers, who are interested to monitor their applications.

First of all we have to clarify that running security services inside the user's resource allows to the provider for collecting fine grained information, but needs the acceptance by the customer to host these services and to share this kind of security information. The Provider, on its own, can get, and eventually share, a complementary set of information, by accessing the hypervisor and the physical infrastructure.

Distributed agents monitor security parameters at different level of Cloud. Agents are autonomous and proactive software components able to migrate dynamically from a virtual node to another together with their code and the status of their execution. They implement security-aware entities configurable by Cloud customers and providers. In particular, each agent is ables to collect raw information about the virtual component (including CPU and memory consumption, network throughput, and so on). The agent reports the collected information both to the user of the monitored virtual node, and to a centralized monitor running on the Cloud provider's machine, which is responsible to gather and correlate the alerts of all agents in the Cloud.

At platform level, security components are enabled to collect on the single VM, security information, such as number of failed authentication requests, policy violations, access rights violations.

Finally, Cloud customers could be interested to infer specific information at application level, such as the application throughput, the number of failed query to the database, the contents of the HTTP requests to their Web application. The customers can use either specific IDSs (including Snort, FSecure Linux Security, and Samhain), which have to be installed from the users on their VM machine, or security features offered by the proposed framework. In particular, as described in Section 5.1, the proposed solution is based on the mOSAIC framework, which offers a Software Platform and APIs to develop Cloud applications that include security mechanisms, like threshold-based filter, proxy components, SLA based components.

The Cloud provider is responsible to enable additional IDSs and attach them to the independent VMs or to the physical machine. Specifically, there should be at least one IDS per level, called "*IDS of Level*": a Network-based IDS and a Host-based IDS for monitoring the infrastructure and the platform level, as well as Network-based IDS for the application level. Thus, the provider can recognize compromised virtual components of their users, by using the security agent deployed on the customer' VM, as well as large scale DDoS attacks against several VMs, by correlating the information collected by the IDSs of Level. The provider can also use automatic countermeasures to react to attacks, *e.g.*, it could decrease the provided resources related to a malicious customer who is running a DDoS attack, or to shut down the compromised VM before more harms can be done.

The customers can configure and monitor their agents by a specific interface that exploits an "User Security Manager" deployed on its local machine. Moreover, they can configure the kinds of agents that are attached to their virtual components, for example, specifying the rule-set and thresholds for each deployed agent to improve the efficiency of the detection activity.

5 Development of Distributed Detection Architecture by mOSAIC Framework

mOSAIC aims at offering a simple framework and APIs to develop and deploy Cloud applications. In mOSAIC a Cloud application is modeled as a collection of components that are able to communicate each other, and consumes Cloud resources (*i.e.*, resources offered as a service from a Cloud provider).

The mOSAIC architecture is composed of two main independent components: *Software Platform* and *Cloud Agency*. The Software Platform enables the execution of applications developed using mOSAIC APIs; and the Cloud Agency acts as a provisioning system, brokering resources from a federation of Cloud providers.

In general, the mOSAIC solution can be adopted in three different scenarios: (*i*) a developer wants to develop an application that runs independently from the Cloud Providers, (*ii*) a Cloud Provider aims at offering enhanced Cloud services, and (*iii*) a single user (like a scientist) is interested to acquire Cloud resources on which start his own application for computational purposes.

In such scenarios, the mOSAIC user develops the application on its local machines, trough the mOSAIC development tools, then it deploys the application on the Cloud. In particular, the developer uses a local instance of the Cloud Agency in order to startup the process of remote resource acquisition, then he/she deploys the Software Platform and the application, and if it is necessary, his/her Application IDSs on the chosen remote VMs.

The Software Platform is the distributed environment that offers a single homogeneous interface by which the components of the mOSAIC Application run. The Software Platform runs on the top of VMs (Cloud resources provided by Cloud Agency), using a dedicated environment: the mOSAIC Operating System (mOS), which is a lightweight Linux distribution, customized in order to run the mOSAIC components. The Software Platform coordinates a set of VMs hosting the mOS as a virtual cluster, on which the software components are independently managed, controlled, and monitored in a completely transparent way.

The Cloud Agency acquires Cloud resources from different Cloud Providers. The acquired VMs host the mOS operating systems, on which the Software Platform has been installed. During the deployment phase, the Cloud Agency executes on the developer's machine. When the Cloud resources for running the mOSAIC applications are available, the Cloud Agency is deployed in the Cloud. Moreover, in order to monitor the acquired resources (*e.g.*, VMs, storage systems), the Cloud Agency is able to migrate on the new VMs independently of the developer's machine.

If it is necessary, the Software Platform uses the Cloud Agency in order to buy new Cloud resources (*e.g.*, in case of overload), which may be either new VMs to be added to the virtual cluster of mOSs, or other kind of resources directly exposed to mOSAIC applications (such as storage systems and queue servers).

5.1 mOSAIC Applications

A mOSAIC Application is a collection of *Components*, which are building blocks able to communicate each other through Cloud communication resources, like queues (*e.g.*, AMQP [25], Amazon SQS [26]). Components instances run in a Cloud environment and exhibit a well defined behavior, implementing functionalities and exposing them to other applications. The components can be categorized in:

- *User Components*, that are developed by developer by means of the facilities offered by mOSAIC, and providing defined functionalities of the mOSAIC Application;
- *Service Components*, that are provided by the Software Platform, and offer predefined functionalities to the applications;

- *Legacy Components*, that are the way in which mOSAIC is ables to interoperate with user technologies, which are cloudified just through installation on VMs (*e.g.*, IDS deployed on a separated VM); the Software Platform is not ables to manage them explicitly, but it is just ables to start and stop them;
- *COTS Components*, that are Commercial Off-The-Shelf components based on non-mOSAIC technologies, and adapted in order to interoperate with mOSAIC Applications; they can be managed directly by the Software Platform in order to assure features like scalability and fault tolerance.

The Legacy Components are pre-configured virtual machines, which host users proprietary software/application. They can interoperate with other mOSAIC Components through the Cloud resources exactly as all the others components (*e.g.*, through a queue). An example of Legacy Component is a VM hosting a IDS like Snort. Such a VM can be started and made available to Cloud providers and Cloud customers.

mOSAIC API offers a simple and flexible way to develop User Components, also called *Cloudlets* (a name inspired by the Java 'Servlet'). An example of User Component is a Cloudlet that receives from a queue (obtained as a Cloud resources by a Cloud provider) a message containing a document, on which it applies a classifying algorithm, extrapolates some user needed information and sends out a message containing the extracted information. Cloudlets are designed to be executed and monitored through the Software Platform facilities, and to be subject of scaling inside the Cloud infrastructure. Each Cloudlet is managed by a *Cloudlet Container*.

The Service Components are software components offered by the Software Platform, which provide specific functionalities developed internally to the framework. From mOSAIC User point of view, the way in which such components are developed is completely transparent: they have to be just instantiated and connected to the resources needed for communication, computation and storage. Example of Service Component can be: the *HTTPgw*, that offers a Web (HTTP) interface and forwards out the content of the received messages on a queue; and a *XML Filter*, that filters all the malicious/malformed XML messages.

Using the programming model offered by mOSAIC, it is possible to add security countermeasures at application levels in a transparent way respect to final user. An example of mOSAIC application that provides intrusion detection and tolerance to web attack is presented in our previous work [24]. The proposed solution consists of a HTTP gateway component (*HTTPgw*) that accepts HTTP requests from Internet and forwards them on internal queues. An intrusion tolerant component is interposed between the gateway and the application components. This additional component acts as filter, cutting away Web attack messages on the base of specific roles, which are defined according to the type of attack that have to be tolerated.

5.2 Platform

The Software Platform exposes several services to the applications:

- *Deploy applications*: Deploy the mOSAIC Cloud application on the resources acquired by the Cloud Agency.
- *Components monitoring*: The deployed application components are monitored in terms of the Cloud resources usage (*e.g.*, the number of activated Cloudlets, the CPU and memory consumption of each software component). The collected information is made available to other internal Software Platform components and to external monitoring tools.
- *Start/Stop application*: The Software Platform manages the start/stop of the application when such a request is received.
- *Autonomic scale of the application components*: On the base of the monitored parameters related to Components' execution (CPU, memory consumption, etc.), the Software Platform may decide to scale the components up or down. The mOSAIC Application itself is not aware of the scaling process.
- *Schedule components execution*: The Software Platform schedules application components to run on specific VMs. The scheduling is defined either at the deployment time, when all the application components have to be started on the VMs, or at execution time, when the scaling manager decides on scaling up or down.

The Software Platform architecture is composed of several components that are responsible of application deployment, platform control, scheduling, scaling, and monitoring. All the components run on the top on mOSAIC Operating System (mOS), customized to manage mOSAIC components.

The mOS boot is a 'clean' system without any operative feature, except the the deployment functionality available at mOS startup. Such functionality is performed by the *Deployer* Component, which enables the dynamic deployment of other Components on VM hosting the mOS.

The main Component needed for both the execution of the other Components and the management of Cloud resources is the *Controller*, which is automatically loaded by the Deployer. This Component has the role of managing all the Components on the top of the VMs and controlling their execution (including start, stop, choose the target VM). A single Controller Component is able to manage more than one single VM. When new resources are offered by the provisioning system (*i.e.*, when the Cloud Agency buy a new VM hosting the mOS), it is notified to the Controller, which maintains the cluster of VMs assigned to it. When a new VM starts, hosting the mOS, it can be assigned to an already existing Controller, which has the role of deciding which Components start on the newly acquired resources. mOSAIC developer has no visibility of both the number of resources effectively available and the allocation of Components on VMs. Acquisition and removal of VMs are completely transparent respect to the application execution.

The *Drivers* are a different class of Component; they aim at offering an homogeneous way to access to different Cloud technologies. A dedicated Driver

exists for each API to which a mOSAIC Component needs to directly access. mOSAIC Cloudlets access to them trough the *Connectors* abstraction. For instance, mOSAIC offers a queue Connector that enables a Cloudlet to publish and consumer messages from a generic queue. The Connector offers the standard interface, and when it is used by the Cloudlet, it sends a standard message to a dedicated Driver. If the application uses a COTS Component, like RabbitMQ, the Connector sends the message to the RabbitMQ Driver, which uses directly the native RabbitMQ API. If the application chooses to acquire an Amazon SQS queue server, the Connector sends its messages to the Amazon SQS Driver that uses the Amazon API. The Controller starts the right Driver when the Cloud resource (*e.g.*, RabbitMQ COTS Component or Amazon SQS link) is obtained trough the Cloud Agency.

Finally, at this level it is possible to collect information about the mOSAIC APIs invocation, about the number of instances and the utilization of all mOSAIC components and of their failures. Furthermore, it is available the status of all COTS component used by the applications and by the platform itself. Some concrete examples for a queue service are the queue length, the dimension and the frequency of exchanged messages. In the case of a Key-Value store, useful information could be the profile of write and read operations, including the distribution of their performers. It allows also for identifying at a finer level what is the attacked service or the compromised component.

5.3 Cloud Agency

The Cloud Agency is a *multi-agent based system* that is responsible for Cloud resource negotiation, re-negotiation, and monitoring. The Agents are software components that provide service to the Cloud user and the Software Platform. In particular, the agent layer provides facilities to design, develop and deploy agents-based services. These services are accessible through a repository. While making services accessible by standard technologies, the agent technology is made "invisible" to the final user.

A core set of agents implements the brokering functionalities. They are in charge of interacting with the Cloud providers and the customers in order to negotiate and broker the needed resources. Agents communicate among them via standard ACL (Agent Communication Language) over HTTP, or over other transport protocols, if it is necessary. Services provided by agents are exposed through a Web service interface that works as a Message Gateway, which translates the OCCI REST messages to ACL, and ACL to OCCI REST messages. Other agents provide the necessary facilities to expose management, monitoring and reconfiguration services.

The Cloud Agency architectural model includes the following concepts, implemented by different kind of agents:

– *Vendor*: It implements a wrapper for a specific Cloud provider. It is used to: (*i*) get an offer for resource provisioning; (*ii*) accept or refuse such offer; (*iii*) get information about a Cloud resource; and (*iv*) perform an action on it (start, stop, resume).

- *Broker*: On the base of required SLA, the Broker asks to Vendor the retrieved offers and brokers the best one. It implements different policies for effectively composing the SLA according to the application requirements and the Vendors' proposals.
- *Meter*: It is deployed on Cloud resources to measure performance indexes.
- *Archiver*: It collects measures from Meters and stores them in a knowledge base. It provides metrics and performance figures computed on the knowledge base.
- *Tier*: It uses reconfiguration policies to: (*i*) detect critical events from performance monitoring of Cloud resource; (*ii*) evaluate critical conditions, for example for each VM to monitor, it identifies SLA violations, CPU and memory saturation or underutilization; and (*iii*) perform required reactions, such as new provisioning, and reconfiguration.
- *Mediator*: It receives information and configuration requests directly from the Cloud provider, the customer, and the Software Platform. It is in charge of starting new transactions for provisioning by creating broker agents. It starts also new Tiers and Meters.
- *Notifier*: It is enabled to notify specific events to the Software Platform, including resource provisioning, resource reconfiguration, resource failure, etc.

However, at this level, performance figures of each virtual host are collected and made available. They can allow for detecting anomaly conditions, but they represent an aggregated information, at VM level, and still need to be correlated with the information collected at the other levels, in order to discriminate between an exceptional 'normal' overload and a security attack.

5.4 mOSAIC Monitoring Features

In the context of mOSAIC Framework, the monitoring system offers a set of tools, whose goal is to evaluate the global state of both the distributed Cloud application and the Cloud resources, and to generate warning when the target application and/or the associated resources are in conditions which may lead to a violation of the SLA due to shortfalls in performance, peak loads, or security concerns. In the following, we briefly summarize the main concepts related to mOSAIC monitoring, a deeper analysis and examples of the components usage can be found in [27].

The main duties of the monitoring system are:

- monitoring of resources and software components at different Cloud levels (Infrastructure, Platform, Application);
- collection and notification of security information; and
- verification of security rules, in order to detect warnings and malicious activities.

As it is shown in Figure 2, resources monitoring (at infrastructure level) is mainly managed by Cloud Agency. A number of meters are migrated and started on each

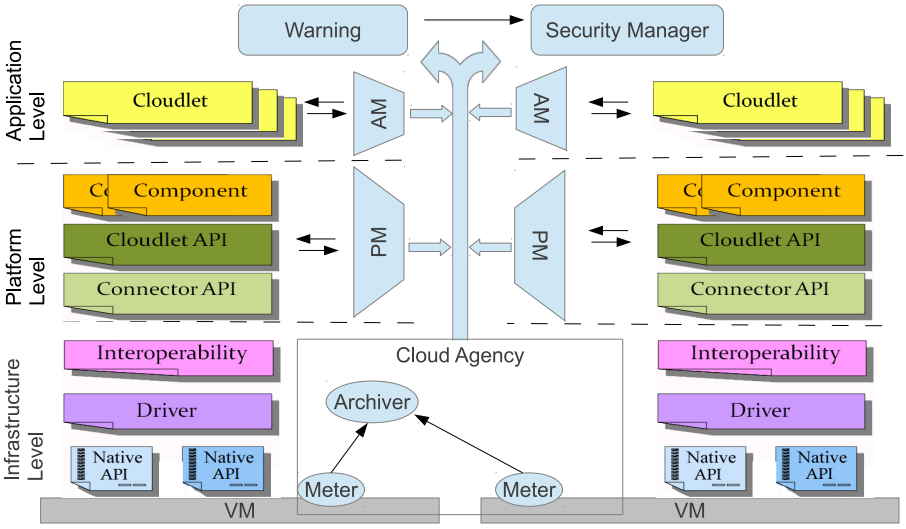

Fig. 2. mOSAIC monitoring components architecture

Cloud resource. Type of measures, time period for measure and for notification can be configured. All information are collected by an Archiver that can be queried about raw information or particular indexes.

Monitoring at Platform level (PM) is performed by an introspection of Software Platform' components, services and connectors, which are provided by mOSAIC and are used by Cloud applications. PM components can be configured for tracing utilizations, failures, and overload conditions of the Software Platform. Cloudlets that compose the customer's application, can be monitored at this level. Cloudlet invocation is the finest grain of resolution that we can observe at platform level.

About the monitoring of what is strictly related to the application development, it is supported by both the API and the mOSAIC Service Components (AM), and it must be implemented by the developer trough ad-hoc solutions. We mean the is up to the mOSAIC user to instrument its application, if he/she wants to observe what is happening inside a Cloudlet. Of course this kind of monitoring can be supported by the Software Platform in the sense that the developer is provided with some facilities to publish the collected information and make them available to the warning engine.

Monitoring information are available by a common *Event Bus*, that exports a publish/subscribe interface. Using the publish method any component, which produces new observations, can share its results by sending asynchronous messages. Publish frequency depends on the component configuration. Subscribers can ask for being notified about new messages. Special filters can be set up to receive messages only from defined senders or with specified preferences.

In our architecture, messages are notified to a *Security Manager* that processes a set of rules that are triggered by the received messages. In case of violation, an alert is notified to the Cloud provider and the Cloud customer.

In order to unify the heterogeneous data collected by different security components and IDSs, at different architectural levels, we use the IDMEF standard for all the communication within the IDS management system.

5.5 Security Manager

The Security Managers are software components that offers to the Cloud customers and to the Cloud providers, the interfaces and an API to build customized applications used to monitor the Cloud applications and the Cloud resources respectively. It is responsible for representing the collected warning events as well as analyzing the events by using specific security rules, *e.g.*, correlation rules for detect complex attack scenarios. Warning events are security information collected by mOSAIC monitoring facilities, and remote Agents that are able to infer security data from the deployed IDSs. All the received events can be hold in a local or remote data base. The IDSs are deployed by Cloud Providers on physical machines or dedicated VMs, as well as by customers on either their own VMs, sharing the resources with the applications, or separate VMs. Of course, while Cloud providers can get information about all their customers and correlate a wider set of aggregated information, the customers have a restricted vision about the system (above all they know only about their own virtual infrastructure), but are able to perform analysis of the behavior of their applications with different degree of awareness.

6 Conclusions

Facing the complexity of a Cloud architecture, this paper proposes a framework to build distributed Intrusion Detection architecture in the Cloud Computing.

The proposed solution allows to develop and deploy security components on each level of the Cloud to collect and correlate remotely the alerts acquired by different sensors. The proposed architecture meets the requirements of the Cloud providers and their customers. It is based on the mOSAIC framework and Software Platform, which allow to configure and deploy security component directly by the local machine of the Cloud provider and the Cloud customers. In particular, by using the functionalities offered by mOSAIC, security agents and sensors can be deployed at different levels of the Cloud architecture: infrastructure, platform and application, in order to monitor both the Cloud resource and the software components hosted on the VMs.

The implemented architecture is a first prototype of complex IDS management system deployed in the Cloud. Specifically, at the state of art, a first release of the mOSAIC framework (including the Cloud Agency, the Software Platform, the API for Cloudlet and Agent development, and supporting tools) and the developer user guide are available on the main project site [10] and on the public code repository [28].

A first prototype implementation of the Security Manager is developed by an open source framework, named Prelude [29], which collects the events generated by the Agents and the mOSAIC monitoring facilities. The prelude manager server processes the received events and delivers them to a specified media (mysql database, postgresql database, XML file). Moreover, Libprelude provides the necessary functionality for processing and emitting IDMEF events. Finally, a '*Prelude console*' has been extended to implement Web Monitors used to view the detected attacks and the system status.

In future work, the Security Manager will be extended to perform complex processing and event correlation. In particular, we will define complex models for filtering and correlating the large volume of heterogeneous security data collected by different sensors in the Cloud. Additionally, new types of security sensors, which are useful for the Cloud needs, have to be developed and integrated into the architecture. Moreover, specific Agents have to be designed to support the more common IDSs.

Finally, since cyber attacks that aim to exhaust target resources, have special effects on service performance in Cloud, due to the pay-per-use business model, specific countermeasures have to be defined and implemented, in order to avoid paying credits in case of deliberate intrusion that cause violations of QoS guarantees negotiated with the customers.

Acknowledgment. This research is partially supported by the FP7-ICT-2009-5-256910 (mOSAIC) project and the MIUR-PRIN 2008 project "Cloud@Home".

References

1. Westphall, C.B., Lamin, F.R.: SLA Perspective in Security Management for Cloud Computing. In: Proc. of the Int. Conf. on Networking and Services (ICNS), pp. 212–217 (2010)
2. Ficco, M., Rak, M.: Intrusion Tolerance of Stealth DoS Attacks to Web Services. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 579–584. Springer, Heidelberg (2012)
3. Kossmann, D., Loesing, S.: An evaluation of alternative architectures for transaction processing in the cloud. In: Proc. of the Int. Conf. on Manag. of Data (2010)
4. Emeakaroha, V.C., Maurer, M., Dustdar, S., Acs, S., Kertesz, A., Kecskemeti, G.: LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageble Cloud Infrastructures. In: Proc. of the IEEE 34th Conf. on Computer Software and Applications, pp. 365–370 (November 2010)
5. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: An Early Performance Analysis of Cloud Computing Services for Scientific Computing. TU Delft/PDS Technical Report PDS-2008-12 (December 2008)
6. Barbosa, P.R., Righi, R.R., Kreutz, D.L.: Defining Metrics to Sec-SLA Agreements in Conformance to International Security Standards. In: Proc. of the 23rd Latin American Informatics Conference, pp. 36–47 (2007)

7. Righi, R.R., Pelissari, F.R., Westphall, C.B.: Sec-SLA: Specification and Validation of Metrics to Security Service Level Agreements. In: Proc. of the Computer System Security Workshop, pp. 199–210 (2004)
8. Ficco, M., Romano, L.: A Generic Intrusion Detection and Diagnoser System Based on Complex Event Processing. In: Proc. of the 1st International Conference on Data Compression, Communications and Processing, pp. 275–284. IEEE CS Press (June 2011)
9. Gul, I., Hussain, M.: Distributed Cloud Intrusion Detection Model. Int. Journal of Advanced Science and Technology 34, 71–82 (2011)
10. mOSAIC Project, mOSAIC: Open source API and platform for multiple Clouds (May 2012), http://www.mosaic-cloud.eu
11. Amazon Elastic Compute Cloud (Amazon EC2), Amazon (April 2012), http://aws.amazon.com/ec2/
12. Windows Azure Platform, Microsoft Corporation (April 2012), http://www.microsoft.com/azure/
13. Google App. Engine, Google (April 2012), http://code.google.com/appengine/
14. Curry, D., Debar, H.: Intrusion Detection Message Exchange Format: Extensible Markup Language (XML) Document Type Definition, draft-ietf-idwg-idmef-xml-10.txt (January 2003)
15. Ramgovind, S., Eloff, M., Smith, E.: The Management of Security in Cloud Computing. In: Proc. of the Int. Conf. on Information Security for South Africa (2010)
16. Schulter, K.: Intrusion Detection for Grid and Cloud Computing. IEEE IT Professional Journal (July 2010)
17. Bhadauria, R., Chaki, R., Chaki, N., Sanyal, S.: A Survey on Security Issues in Cloud Computing (September 2011), http://arxiv.org/abs/1109.5388
18. Palmieri, F., Pardi, S.: Towards a federated Metropolitan Area Grid environment: The SCoPE network-aware infrastructure. Future Generation Computer Systems 26(8), 1241–1256 (2010)
19. Zhang, R., Xie, W., Qian, W., Zhou, A.: Security and Privacy in Cloud Computing: A Survey. In: Proc. of the the 6th Int. Conf. on Semantics Knowledge and Grid, pp. 105–112 (November 2010)
20. Cheng, F., Meinel, C.: Intrusion Detection in the Cloud. In: Proc. of the IEEE Int. Conf. on Dependable, Autonomic and Secure Computing, pp. 729–734 (December 2009)
21. Lo, C.-C., Huang, C.-C., Ku, J.: A Cooperative Intrusion Detection System Framework for Cloud Computing Networks. In: Proc. of the 39th Int. Conf. on Parallel Processing, pp. 280–284. IEEE CS Press (September 2010)
22. Park, M.-W., Eom, J.-H.: Multi-level Intrusion Detection System and Log Management in Cloud Computin. In: Proc. of the 13th Int. Conf. on Advanced Communication Technology, pp. 552–555. IEEE CS Press (February 2011)
23. Ficco, M., Rak, M.: Intrusion Tolerance as a Service: A SLA-Based Solution. In: Proc. of the 2nd Int. Conf. on Cloud Computing and Services Science. IEEE CS Press (April 2012)
24. Ficco, M., Rak, M.: Intrusion Tolerance in Cloud Applications: the mOSAIC Approach. In: Proc. of the 6th Int. Conf. on Complex, Intelligent, and Software Intensive Systems (2012)
25. Amqp - Advanced message queuing protocol (April 2012), http://www.amqp.org/

26. Amazon Web Services LLC - Amazon simple queue service (amazon sqs) (April 14, 2012), http://aws.amazon.com/sqs/
27. Rak, M., Venticinque, S., Mhr, T., Echevarria, G., Esnal, G.: Cloud application monitoring: The mosaic approach. In: Proc. of the IEEE Int. Conf. on Cloud Computing Technology and Science, pp. 758–763 (2011)
28. mOSAIC Consortium. mOSAIC source repository (April 14, 2012), https://bitbucket.org/mosaic
29. Prelude, an Hybrid Intrusion Detection System (February-April 2012), http://www.prelude-technologies.com/en/welcome/index.html