An Intrusion Detection Framework for Supporting SLA Assessment in Cloud Computing

Massimo Ficco, Massimiliano Rak, and Beniamino Di Martino
Dipartimento di Ingegneria Industriale e dell’Informazione
Second University of Naples (SUN)
Via Roma 29, 81031 Aversa, Italy
Email: massimo.ficco@unina2.it

Abstract—Cloud Computing is the emerging paradigm in distributed environment. It is an opportunity for users to reduce costs and increase efficiency. Cloud Computing represents both a technology for using computing infrastructures in a more efficient way, and a business model for selling computing services and resources. In this context, cyber attacks represent a serious danger, which can compromise the quality of service delivered to the customers. 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 distributed security components, which can be used to perform complex event correlation analysis to identify intrusions in the Cloud system that involve Service Level Agreement violations.

Keywords—Cloud Computing; SLA; security; complex event processing.

I. INTRODUCTION

Cloud Computing is an emerging paradigm that allows users to obtain computing services and resources (such as networks, servers, storage, and applications), according to a pay-per-use business model [1]. The provisioning contracts offered by the cloud service providers, regulate the cost that customers have to pay for delivered services and resources, as well as the penalty that the cloud provider has to pay, if the customer’s requirements are not satisfied. In order to support this model, the cloud infrastructure has to continually adapt to changes of customer demands and operation conditions. For example, in order to prevent service availability may be required additional standby resources to handle a given number of failures, whereas to prevent performance degradation may be required to scale up or move a virtual machine (VM) to another physical machine, if the current machine is overloaded. On the other hand, in the presence of cyber attacks, such an on-demand characteristic complicates the assessment of the quality of the provisioned service in Cloud Computing. For example, in order to face peak of loads due to Denial of Service attacks, 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), additional resources should be either already available, or dynamically acquired on demand [2], [3]. On the other hand, such resources are not free, therefore, the question is: who pays for resources consumption due to a cyber attack?

In order to define a clear agreement between responsibility assignment, a lot of interests are assuming the Service Level Agreements (SLA). From cloud consumer point of view, a SLA is a contract that grants the customer about what he/she will effectively obtain from the service. From cloud provider point of view, SLA is a way to have a clear and formal definition of the requirements that the delivered service has to respect.

Several works proposed in literature present models and mechanisms for monitoring and assuring service performance and availability guarantees in the Cloud Computing context [4]. On the other hand, the inability of the cloud service infrastructure to diagnose the causes of a 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 quality of service (QoS). It is clear that the 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]. On the other hand, these requests have profound implications in the security solution to be implemented for monitoring the delivered cloud services.

In this paper, we propose an extensible intrusion detection management framework, which can be offered to cloud providers in order to implement complex 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. The platform level includes the basic software hosted on the VM, such as, the operating system, and the Application Programming Interface (API). Finally, the application layer includes other software running in the Cloud, such as a Web application. The collected security information can be used to discriminate whether the SLA violation is due to either a malicious activities or a legitimate system overloading.

The presented solution is based on the mOSAIC framework [8], 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 Intrusion Detection System (IDS) in Cloud, needed to support the SLA assessment.

A specific interface and APIs have been designed to implement security engine components, which perform complex event correlation of security information, inferred by the different security components deployed on different cloud architectural levels, which can be used to identify malicious SLA violations. 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 rest of the paper is organized as follows: Section II presents the related work in the field of SLA and IDS management in Cloud. The proposed IDS architecture is presented in Section III. Section IV presents the implementation of the solution by means of the SLA and security features offered by the mOSAIC framework. Section V presents the conclusions and future works.

II. RELATED WORK

Several research work has been performed to face the problem of service level management in Cloud [12], [13]. In general, the proposed solutions enables a cloud provider to offer its services with a granted quality, as well as to provide automatic mechanisms for cloud resources adaptation, designed to ensure the contracted QoS. In this direction, the most interesting results come from SLA@SOI [11], which represents a clear starting point for the SLA management in complex architectures. It proposes solutions to design SLAs, and manage them trough different representations. Moreover, several works [14] propose solution to archive real-time assessment about the provided QoS by a service provider.

On the other hand, SLA and autonomic resources adaptation mechanisms are strictly related each other, and partially in conflict: how to grant an agreement for a specific service that dynamically changes its behavior due to malicious activities. Although several distributed IDS solutions have been proposed to face security aspects in public and private cloud infrastructures [16], [18], [9], [10], [17], [19], [20], their use in supporting the identification of SLA violations, due to malicious activities, is still a challenging task [5]. 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 [15]. 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 a malicious user, or by a virtualized component that may has been compromised. Therefore, cloud providers need to know if their systems and infrastructure is used by attackers to penetrate other victims.

III. THE DISTRIBUTED INTRUSION DETECTION ARCHITECTURE

The proposed intrusion detection framework is a distributed architecture represented in Figure 1, which consists of different security components deployed both on the provider’s machines, and on the machines of customers who are interested to monitor their applications.

Distributed agents monitor security parameters at different levels of the 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 the customers and the cloud providers. In particular, at infrastructure level, each agent is able to collect raw information about the virtual node on which it runs (e.g., CPU and memory consumption, network throughput). The agent reports the collected information both to the user of the monitored virtual node, and to a centralized security engine 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 security information on the single VM, such as number of failed authentication requests, policy violations, access rights violations. Finally, at application level, cloud customers could be interested to infer specific information, such as the application throughput, the number of failed query to the database, the contents of the HTTP requests to Web application.

The cloud consumers can use either specific IDSs (including Snort, FSecure Linux Security, and Samhain), which have to be installed from them on their VMs, or security
features offered by the proposed framework. In particular, as described in Section IV-A, 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. Moreover, the consumers can configure and monitor their agents by a specific interface that exploits an “User Security Engine” 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 the thresholds for each deployed agent to improve the efficiency of the detection activity.

The cloud provider is responsible to enable additional IDSs, and attach them to the independent VMs or to the physical machine. There should be at least one IDS per level, called “IDS of Level”. Specifically, Network-based IDS for monitoring the infrastructure and the platform level, and Network-based IDS and Host-based IDS (one for each virtual node) at 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 distributed 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, as well as to shut down the compromised VM before more harms can be done.

However, 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, which has 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.

IV. IDS DEVELOPMENT BY MOSAIC FRAMEWORK

mOSAIC aims at offering a framework and the APIs to develop and deploy cloud applications. In mOSAIC, a cloud application is modeled as a collection of components, which are able to communicate each other, and consumes cloud resources.

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

The mOSAIC user develops the application on its local machines, trough the mOSAIC development tools, and then, it deploys the application on the Cloud. In particular, the developer uses a local instance of the Cloud Agency in order to perform 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 a 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, 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. Then, in order to monitor the acquired resources (e.g., VMs, storage systems), when the cloud resources for running the mOSAIC applications are available, the Cloud Agency is deployed independently of the developer’s machine.

If it is necessary, the Software Platform uses the Cloud Agency in order to acquire 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).

### A. 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., Amazon SQS [21]). Component 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 provide defined functionalities of the mOSAIC Application.
- **Service Components**, that are provided by the Software Platform, and offer predefined functionalities to the applications. 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. Examples of Service Components 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.
- **Legacy Components**, that 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). However, the Software Platform is not able to manage them explicitly, but it is just ables to start and stop them. An example of Legacy Component is a VM hosting an IDS like Snort. Such a VM can be started and made available to cloud providers and cloud customers.

- **COTS Components**, that are Commercial Off-The-Shelf components based on non-mOSAIC technologies, and adapted in order that they can interoperate with mOSAIC Applications. They can be managed directly by the Software Platform in order to assure features like scalability and fault tolerance.

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.

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 [22]. The proposed solution consists of a HTTP gateway component, which 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.

### B. mOSAIC Offering for SLA-based Applications

mOSAIC offers a set of components that offer different functionalities for SLA management:

- **SLA Negotiation**: it manages the SLA documents and their formal management, i.e., negotiation protocols, auditing, and so on.
- **SLA Monitoring**: it detects the warning conditions and generates alerts about the difficulty to fulfill the agreements. It addresses both resources and applications monitoring. It is connected with the Cloud Agency.
- **SLA Enforcement**: it manages the elasticity of the application. Specifically, the SLA Enforcement is in charge of making decisions in order to grant the respect of the SLA agreements. It uses a SLA Policy sub-component, that acts mainly as a policy decision point, which evaluates policies and emits messages on the basis of the policy evaluation. It uses two different storages, the first one dedicated to the policies, the second one, shared with the SLA Monitoring, on which
the monitoring parameters are stored. The policy are described in a JSON format, and represent set of triples of kind:

\[
<\text{Parameters}>, <\text{Condition}>, <\text{Action}>
\]

The parameters are data that has to be collected by both the SLA Monitoring and the Security Engine (described in Section IV-D). Conditions is a boolean expression, which can be evaluated using Parameters as variables. If the Condition evaluation is true, a message containing the Action value is sent the mOSAIC Applications in order to assume decisions about the reaction to be performed for SLA satisfaction.

C. 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, as well as to generate warning when the target application and/or the associated resources are in conditions which may lead to a violation of the SLA (e.g., 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 [23].

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.

The resources monitoring at infrastructure level is mainly managed by Cloud Agency. A number of meter Agents are migrated and started on each cloud resource. All information are collected by an Archiver that can be queried about raw information.

Monitoring at Platform level is performed by an introspection of Software Platform, which can be configured for tracing utilization, failures, and overload conditions of the Cloudlets that compose the customer’s application. 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, and it has to be implemented by the developer trough ad-hoc solutions. We mean that 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 Security 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 the proposed architecture, messages are notified to the Security Engine, which processes a set of rules that are triggered by the received messages. In case of a violation, an alarm is notified to the SLA Enforcement, which on the base of the information received from both the SLA Monitoring and the Security Engine, decides which action performs. For example, if a SLA violation is related to a current attack, it may just notify an alarm to the cloud provider or the customer; otherwise, based on a policy evaluation, it may send a message containing an Action value to the application, which performs (by mOSAIC features) a specific self-adaptation to satisfy the SLA.

D. Security Engine

The Security Engines are software components that offer to the cloud consumers and providers, the interfaces and an API to build customized applications used to monitor the cloud applications and the cloud resources respectively. The Security Engine 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 distributed attacks. 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 their own VMs, sharing the resources with the applications, or on separate VMs. Of course, while cloud providers can get information about all their customer’ resources and correlate a wider set of aggregated information, the customers have a restricted vision about the system (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.

In order to recognize distributed attacks to the Cloud and enrich the semantic of diagnosis, a correlation approach based on a Complex Event Processing (CEP) is adopted. It captures the causal relationships between the resulting alarms (which may represent symptoms of the same attack collected at different cloud architectural levels), by correlating them on the base of temporal and logical constraints. It can be also used to detect complex attack scenarios that consist of specific sequence of malicious activities performed by the attacker in order to discover Cloud vulnerabilities.
As presented in a previous work [9], the CEP is in charges of performing real-time complex queries on different data streams in order to detect attack scenario. Knowledge-base is used to automate the process of deriving roles for recognition of complex and distributed attacks. The CEP engine is based on Borealis [24], an open source framework for query processing of event streams. It performs SQL-style processing on the incoming events streams, without necessarily storing them. It uses specialized primitives and constructs (e.g., time-windows, logical condition) to express stream oriented processing logic. Queries consist of merging streams produced by the Cloud Agency and the distributed Agents via Join constructs.

V. Conclusions

This paper proposes a framework to build distributed Intrusion Detection architecture in the Cloud Computing used to support SLA assessment. It is based on the mOSAIC framework and Software Platform, which allow to develop and deploy security component directly by the local machine of the cloud provider and the cloud customers, in order to monitor both the cloud resources and the software components hosted on the VMs.

The implemented architecture is a first prototype of an IDS management system deployed in the Cloud. Future works will consist in defining complex models for mapping the security monitored metrics and SLA parameters in cloud environments.

Acknowledgment

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

References


