INTRUSION TOLERANCE AS A SERVICE
A SLA-based Solution

Massimo Ficco and Massimiliano Rak
Dipartimento di Ingegneria dell'Informazione, Seconda Università di Napoli, Aversa, Italy

Keywords: Intrusion Tolerance, Service Level Agreements, SLA, Cloud, Denial of Services.

Abstract: Among the incredible number of challenges in Cloud Computing two of them are considered of great relevance: Service Level Agreement management and Security management. In this paper we will try to show how it is possible, using a cloud-oriented API derived from the mOSAIC project, to build up an SLA-oriented cloud application which enables the delivery of security solutions as a service. We will focus on intrusion tolerance solutions, i.e., systems which grant that a system maintain a (limited) availability even when a security attack take place.

1 INTRODUCTION

Cloud Computing is an emerging reality. Following the NIST definition, Cloud computing is *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or Service Provider interactions [...]"*. The key idea is the delegation *to the network* of every kind of resources, that can be obtained and managed on a self-service based approach and with the perception of infinite amount of available resources.

At the state of art, the most diffused actual solutions focus on the so-called Infrastructure as a Service (IaaS) Providers, which mainly offer virtual machines (and storage systems) following the *as a service* paradigm. This kind of services are offered now by both big players, like Amazon, IBM and Microsoft, and by little providers, like Rackspace and GoGrid. Such resources are offered to Cloud users on the basis of a pay-per-use model: the resources are payed only for their effective usage (CPU active time, amount of data stored, etc).

In such environment, a well-known diffused problem is related to the security features that can be offered (and granted) by such Cloud Providers: are the offered virtual machines protected against security attacks (like Denial of Services)? Due to the pay-per-use business model these problems are really relevant: who pays for resources consumption due to a security attack?

In order to define a clear agreement between responsibility assignment, a lot of interests are assuming the *Service Level Agreements (SLAs)*. An SLA is an agreement between a Service Provider and a customer, that describes the Service, the service level targets, and specifies the responsibilities of the Provider and the customer. SLAs aim at offering a simple and clear way to build up an agreement between the final users and the Service Providers in order to establish *what is effectively granted* in terms of quality. From user point of view, a SLA is a contract that grants him/she about what he/she will effectively obtain from the service. From Cloud Providers point of view, SLAs are a way to have a clear and formal definition of the requirements that the application must respect. SLAs are a way to formalize the agreements, but no stable solutions exists today in order to monitor and enforce the respect of such agreements, even if a lot of research effort is spent today in such problems.

In this paper we will try to show how it is possible, using a Cloud-oriented API derived from the mOSAIC project (mOSAIC Project, 2010; van Sinderen F. Leymann et al., 2011; Massimiliano Rak, 2011), to build up a SLA-oriented Cloud application, which enables an IaaS Cloud Provider to offer security service customized on user needing on the top of the resources delivered.

The problem on which we focus is to offer, in a transparent way, a service that is ables to offer the typical IaaS services (mainly Virtual Machines delivery, starting and stopping) enriching them with ad-hoc solutions for protecting the delivered resources against a set of security attacks. The goal is to enable the Cloud user to negotiate with the provider the level of security offered, so that the service offered (for example a Web server) will be protected against a given set of attacks. The user pays for the additional security service, but he/she is granted that the services is tolerant to a given set of Denial of Services attacks (*i.e.*, continues to work even under attack) and the additional load generated is not charged.

The remainder of this paper is organized as follows: next section is dedicated to clarify the approach adopted to solve the problem. Section 3 is dedicated to intrusion tolerance technologies and offers a basis for the description of the Intrusion Tolerant solution adopted in our proposal, described in Section 4. Section 5 describes the technology we adopted in order to build up the solution, while Section 6 describes our Cloud application that enables the offering of SLA negotiation and enforcement. Section 7 describes the final integrated solution and the offered SLAs. Section 8 summarizes the related work and Section 9 concludes the paper.

2 THE PROPOSED APPROACH

The problem we focus on this paper is the design and implementation of a system able to offer Intrusion Tolerance (IT) solutions *as a Service*. The key idea is that the approach of offering security solutions *as a Service* can be better obtained following a SLA-based approach: final user invokes the services always in the same way, but, due to the integration of the system together with an SLA-based application, the services offered can be enriched with security mechanisms.

In order to better illustrate the problem, Figure 1 shows the approach we proposed: the Cloud Provider offers typical Infrastructure as a Service (IaaS) functionalities, offering virtual machines following a well known interface. In the example presented, the Cloud Provider offers, among the list of available images, a set of images offering pre-configured Web Servers: the user, invoking the provider's services, is ables to startup a Web Server, that he is able to customize and/or enrich with given functionalities.

Thanks to the adoption of the mOSAIC Framework (see section 5), the provider is ables to offer more complex services on the top of the virtual machines. In the proposed example, the provider offers SLA-based application, which enables the *final user* to negotiate, trough the WS-Agreement standard description, the quality of services delivered.

In this paper, we will show that it is possible even

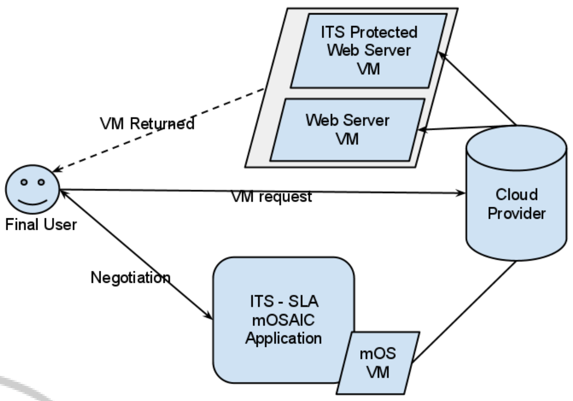

Figure 1: Intrusion Tolerance as a Service with SLA-based approach.

to negotiate security-oriented parameters. In our case study, we will show that the delivered machine will be enriched, after the negotiation, with IT techniques, that grant the user against some of possible Denial of Services (DoS) attacks.

As shown in Figure 1, the user invokes exactly the same service, but, due to the SLA negotiation process, instead of the binding to the unprotected Web Server, he will receive the binding to a VM, which is enriched with IT functionalities. Service invocation does not change and from final user, obtaining a standard machine or a protected one is completely transparent.

Moreover, trough the SLA-based application, it is possible to help Cloud Provider and user to agree on the details of the security granted, identifying the features that should be offered.

3 INTRUSION TOLERANCE: DEFINITIONS, METHODS AND TECHNIQUES

Intrusion Tolerance is the ability of a system to continue providing (possibly degraded) adequate service, despite the presence of malicious faults, *i.e.*, deliberate attacks on the security of the system by both insiders and outsiders. In order to enforce the IT several techniques can be used.

Replication is the technique most commonly used to perform IT. It consists to use more replicas of the same component and use specific voting algorithms, which are used to resolve any difference in redundant responses and to arrive at a consensus result based on the responses of perceived non-compromised components in the system. It has two complementary goals: masking of intrusions, thus tolerating them, and providing integrity of the data. Examples of algorithms

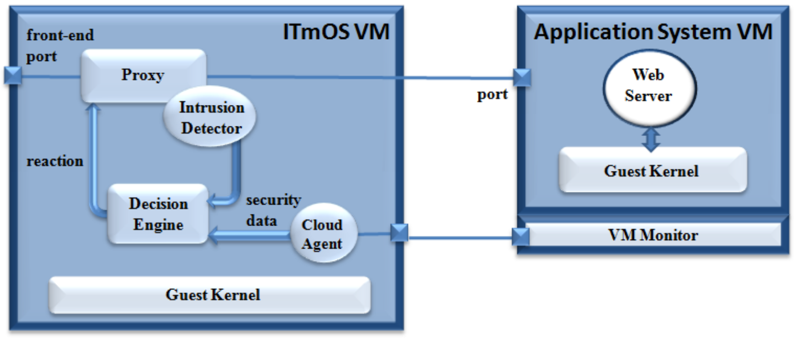

Figure 2: Intrusion tolerant architecture.

are Byzantine replication algorithms (P. Kouznetsov and Druschel, 2006).

Using a *rejuvenation* approach, critical components are periodically rejuvenated to remove the effects of malicious and intelligent attackers that find ways to compromise them. Example of rejuvenation procedures could aim at loading a clean version of the application or change the cryptographic keys (Marsh and Schneider, 2004; N. F. Neves and Verissimo, 2006).

Redundancy is an approach different from replication, which is just one type of redundancy. Replicated components are pure replicas of each other. If the attacker has found a technique to subvert one component and all are pure replicas, it is likely that all components are likewise vulnerable. To combat this, another common technique used is 'diversity'. Diversity is the property such that the redundant components should be substantially different in one or more aspects, from hardware diversity and operating system diversity, to software implementation diversity. Therefore, through the use of diversity, the probability of a replica being compromised is independent of the occurrence of intrusion in other replicas (A. K. Caglayan and Eckhardt, 1989; R. Mista and Medidi, 2002).

Indirection allows designers to insert protection barriers and fault logic between clients and servers/-components that provide services. Since the indirection is hidden outside of the black box system, clients see only what looks like a COTS server. There are at least four main types of indirection used by IT systems: proxies, wrappers, virtualizations, and sandboxes.

Several previous techniques, commonly named proactive, aim at preventing system components being compromised. *Reactive techniques* aim at mitigating and reacting to intrusion. For example, they aim at minimizing stolen resources and disable inappropriate information flows (*e.g.*, through roll back and roll forward) to react/mitigate intrusion impact on the system. Intrusion detection and correlation mechanisms can be used to detect intrusion and identify the specific recovery action (Ficco, 2010) . For example, in replication, it could force the recovery of a replica that is detected or suspected of being compromised.

Reconfiguration can be proactive or reactive and can help in prevention, elimination as well as tolerance. A wide variety of reconfiguration strategies are employed (D. Heimbigner. and Wolf, 2002). A challenge in devising the reconfiguration mechanisms is to protect them from being (mis)used by the attacker. It is important that the reconfiguration process be not very predictable by the attacker. Therefore, a major challenge is to make them unpredictable and resilient to oscillations due to transient and malicious effects that may lead to reconfigurations that drive the system to an inconsistent state.

4 AN INTRUSION TOLERANT SOLUTION

As case study, in this paper we adopt a Intrusion Tolerant reactive technique. The proposed IT architecture is composed of two subsystems, with distinct properties (Figure 2).

The first subsystem is the *Application VM* that hosts the application to protect, whereas the second subsystem, named *ITmOS*, is the VM that hosts the IT 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* (based on running Squid Web proxy (Squ,)) hosted on the ITmOS VM.

A *VM Monitor* monitors the Application VM. It is a Java-based component based on Ganglia-Gmond, which is a real-time monitoring system (Gan,). It is used to collect system resources consumption (including CPU, memory, disk).

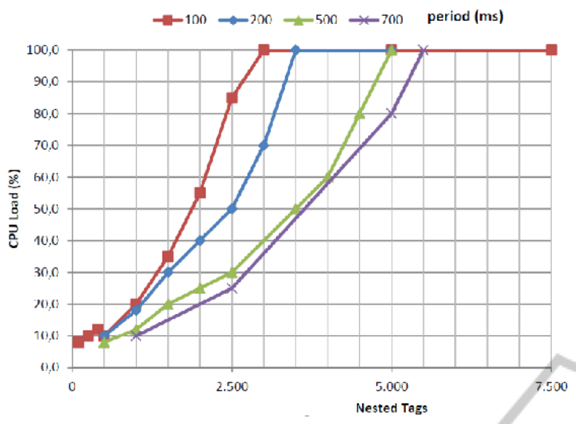

Figure 3: CPU consumption with different message frequencies.

An *Intrusion Detector* module collects data from the Proxy and alerts the *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 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. The Intrusion Detector is connected to a *Cloud Agent*, which provide the necessary facilities to interact with the VM Monitor through a secure communication channel.

4.1 An Example of Intrusion Tolerance Approach for Denial of Service Attacks to Web Server

In our previous work (Ficco and Rak, 2011), we present an IT approach for Denial of Service attacks to Web Server, which exploits XML vulnerabilities. We focus on the Deeply-Nested XML DoS attacks (XDoS), which exhaust the computational resources of the target system by forcing the XML parser within the server application to process numerous deeply-nested tags. In particular, the attack consists of inserting of a large number of nested XML tags in the Web messages.

Figure 3 shows the CPU consumption depending on the number of nested XML tags and the frequency with which the malicious Web messages are injected. We perform different attack scenarios. Each scenario consists of a sequence of messages injected with a fixed frequency and a fixed number of nested tags. An attack scenario takes about 30 seconds. In particular, Figure 3 represents the average value of CPU for the different scenarios, which are performed using different message frequencies and with a number of tags nested to different depths. The experiment shows that it is sufficient to inject messages with about 3000 nested tags every 100 ms, to make unavailable the Application VM (*i.e.*, to exhaust the CPU resource).

4.1.1 XDoS Attack Reaction

The implemented solution allows to trigger a specific reaction to mitigate the effects of the XDOS attacks.

In order to detect the attack, an anomaly-based monitoring approach is adopted that assigns a weight to the anomalous events detected. The weight reflects the anomaly levels with regard to an established profile. If this weight does not exceed a threshold estimated during a training phase, the event is discarded (*i.e.*, it is considered as a normal behavior), otherwise an alarm is triggered and a reaction is activated.

In the proposed architecture (Fig. 2) the Decision Engine correlates the events generated by both the VM Monitor hosted on the Application VM and the Intrusion Detector.

As presented in (Ficco and Rak, 2011), on the occurrence of an excessive CPU consumption, if an anomalous number of nested XML tags with respect to a normal profile is monitored, a reaction in triggered. In particular, the Decision Engine alerts the Proxy, which filters each Web request that contains a number of nested tags greater than a fixed threshold. The purpose of this action is to reduce the CPU load on the Application VM, thus reducing the period in which the Web Server is unavailable.

In order to evaluate the proposed solution, a workload based on TPC Benchmark W (TPC-W) is adopted (TPC,). It is a transactional Web benchmark. The workload is performed in a controlled environment that simulates the activities of a business oriented transactional Web Server. It simulates the execution of multiple transaction types that span a breadth of complexity. Multiple Web interactions are used to simulate the activity of a retail store, and each interaction is subjects to a response time constraint. The performance metric reported by TPC-W is the number of Web interactions processed per second (WIPS). It is used to simulated stress load and to assess the effectiveness of the proposed solution by the WIPS measurements.

An example of recovery effect is shown in Figure 4. It represents the WIPS and CPU variations with respect to the time, during an interval time of tree minutes.

The experiment consists of tree temporal windows. During the first two windows, the Decision Engine is disabled. In particular, the first 60 seconds show the values of the WIPS and the CPU load in absence of the attack. The second 60 seconds show the attack effects. We injected malicious messages with 3000 nested tags every 200 ms. During this period, the CPU load is about 100% and the number of TPC-W interactions processed is very low. Finally, during the last 60 seconds, the Decision Engine is enabled. When the Decision Engine detects the condition of a reaction (*i.e.*, the attack is in progress and the CPU consumption exceed the 90%), it alerts the Proxy, which filters all the suspicious messages. The reaction is applied until the CPU load falls below the 90%.

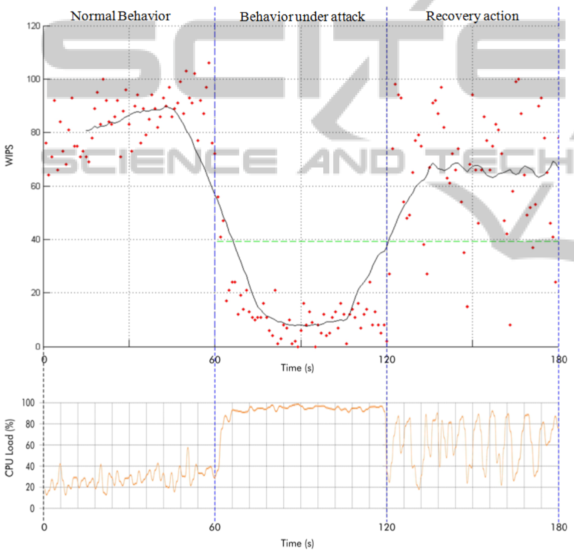

Figure 4: WIPS evaluation during a intrusion recovery process.

5 mOSAIC: DEVELOPMENT OF DISTRIBUTED CLOUD APPLICATIONS

mOSAIC aims at offering a simple way to develop Cloud application. The target user for the mOSAIC solution is a developer (mOSAIC User). In mOSAIC a Cloud application is modeled as a collection of components that are able to communicate each other and consume Cloud resources (*i.e.*, resources offered as a service from a Cloud Provider). Cloud applications often are offered in the form of *SoftwareasaService* and can be accessed from users different from the mOSAIC developer. Users different from the mOSAIC User, which uses a Cloud application are defined *FinalUsers*.

The mOSAIC solution is a framework composed of three independent components: *Platform*, *Cloud Agency* and *Semantic Engine*. The first one (mOSAIC Platform) enables the execution of application developed using mOSAIC API; the second one (Cloud Agency) acts as a provisioning system, brokering resources from a federation of Cloud Providers; the last one (Semantic Engine) offers solution for reasoning on the resources needing and the application needing. For the needing of this paper, we need concepts related to Platform and Cloud Agency, while Semantic Engine will not be focused in this context.

mOSAIC solution can be adopted in three different scenarios: (*i*) when a developer wants to develop an application, which runs independently from the Cloud Providers, (*ii*) when an IaaS Provider aims at offering enhanced services, and (*ii*) when a single user (like a scientist) is interested to start his own application in the Cloud for computational purposes. In this paper, we will focus on the second scenario (the one dedicated to a Provider).

The approach adopted is pretty simple in this case: the Provider starts up a set of its resources dedicated to management of the Cloud application and hosting the mOSAIC Platform. On the top of the Platform, the Provider runs its own mOSAIC Application that directly interacts with the IaaS service offered in order to grant added value services.

5.1 Programming with mOSAIC

A mOSAIC Application is defined as a collection of interconnected *mOSAIC Components*. Components may be offered by the mOSAIC Platform (*i*) as COTS (Commercial off-the Shelf) solutions, *i.e.*, common technologies embed in a mOSAIC component, (*ii*) as Core Components, *i.e.*, tools offered by mOSAIC Platform in order to perform predefined operations, or (*iii*) new components developed using mOSAIC API. In last case, a component is a *Cloudlet* running in a *Cloudlet Container* (van Sinderen F. Leymann et al., 2011). mOSAIC Components are interconnected trough communication resources, queues or other communication system (*i.e.*, socket, web services, etc.). The mOSAIC Platform offers some queuing system (rabbitmq and zeroMQ) as COTS components, in order to enable component communications. Moreover, mOSAIC Platform offers some Core Components in order to help Cloud application to offer their functionalities as a service, like an HTTP gateway, which accepts HTTP requests and forward them on application queues. mOSAIC Components run on a dedicated virtual machines, named mOS (mOSAIC

Operating System), which is based a very small Linux distribution. The mOS is enriched with a special mOSAIC Component, the *Platform Manager*, which enable to manage set of virtual machines hosting the mOS as a virtual cluster on which the mOSAIC Components are independently managed. It is possible to increase and decrease the number of virtual machines dedicated to the mOSAIC Application, which will scale in and out automatically.

The Cloud application is described as a whole in a file named *Application Descriptor*, which lists all the components and the Cloud resources needed to enable their communications. A mOSAIC developer has the role of both develop new components and write Application Descriptors, which collect them together. mOSAIC API, actually based on Java, enables the development of new components (in the form of Cloudlets), which self-scale on the above described Platform and consume every kind of Cloud resources like queues, NO-SQL storage system (like KV store and columnar databases), independently from the technologies and API they adopt, trough a wrapping system.

5.2 mOSAIC Offering for SLA-based Applications

mOSAIC offers a set of features dedicated to SLA management. In (Massimiliano Rak, 2011), we proposed a first outline of the full architecture, while in (Rak et al., 2011) we proposed a similar case study in which we presented an application that offers security access configurations to a GRID environment in terms of SLA. The components offered in the SLA architecture should help the application developer to implement an SLA-based architecture. The main modules offered are :

- *SLA Agreement Management*: This module contains all the Cloudlets and components that manage the SLA documents and their formal management, *i.e.*, negotiation protocols, auditing, and so on.
- *SLA Monitoring System*: This module contains all the Cloudlets and components needed to detect the warning conditions and generates alerts about the difficulty to fulfill the agreements. It should address both resource and application monitoring. It is connected with the Cloud Agency.
- *SLA Enforcement System*: This module includes all the Cloudlets and components needed to manage the elasticity of the application, and modules that are in charge of making decisions in order to grant the respect of the acquired needed to fulfill the agreements.

In this context, we are interested mainly in the SLA management module, which offers the functionality to automatize the SLA negotiation process. The main component dedicated to SLA management is the SLAgw, offered by the mOSAIC framework as a standalone component. SLAgw component offers an incredibly simple way to interact with final users in a SLA-based manner: they will negotiate the agreement. The SLAgw sends out a message to the mOSAIC Application each time a new agreement requests take place. At the state of art, we support the asynchronous WS-Agreement negotiation (*i.e.*, the users agreement requests always receive a wait reply, then it is up to the user to query for the agreement status and obtain an accept or a refuse). In future work, we will support even the synchronous negotiations and more evolved protocols.

The SLAgw component does not assume decisions about the submitted SLA, it just forward them internally to a mOSAIC Application and store the SLA in a shared KV store.

6 SLA-BASED CLOUD APPLICATION FOR IT MANAGEMENT

In our case study, the Provider aims at developing a Cloud application that offers SLA negotiation features, following the SLA model proposed in mOSAIC, in order to offer enriched services to Cloud users. The negotiation has the effect of enabling the IT techniques adopted.

The application we aims at offering offers mainly two different use cases:

- *SLA Negotiation*: that enables an Final User to negotiate a SLA identifying the security of the services offered. Trough the negotiation process, it is possible to enforce the security mechanisms described in Section 4.
- *VM Delivery*: that just delivers a virtual machine to the Final User. In this case, the application just forwards the request to the underlying IaaS infrastructure in order to offer the service to the Final User. Note that, depending on the SLA negotiated, the application may start different number of VMs for the same request done from the Final User.

The SLA negotiation follows the SLA model proposed in mOSAIC, briefly described in Section 5 and based on SLAgw, while the VM delivery is done simply forwarding the requests to the underlying IaaS

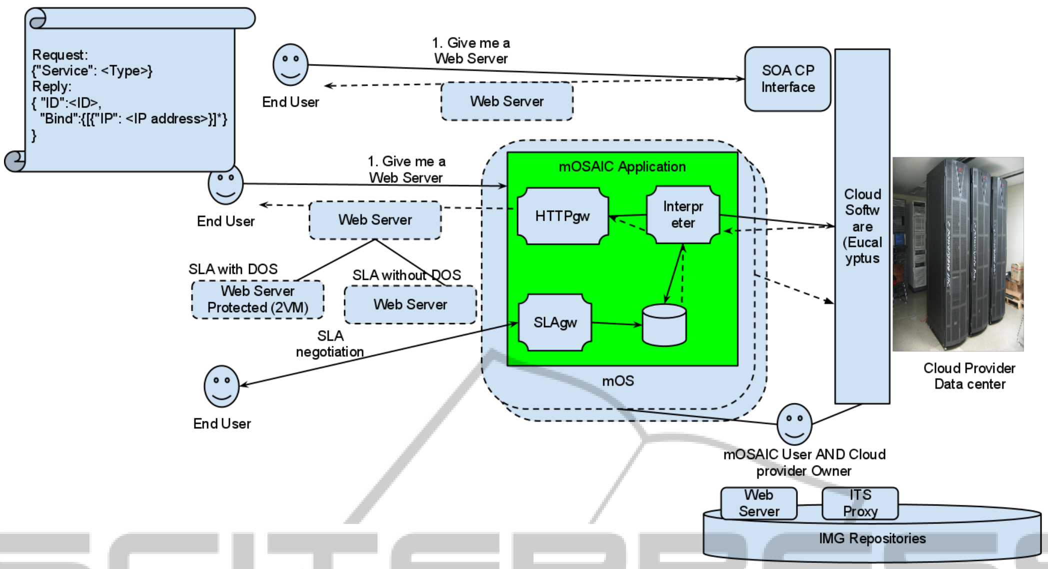

Figure 5: Overal architecture of the IT SLA-based services.

Provider in the case of non-protected requests, instead performing a set of operation when an IT system is required.

Figure 5 describes the global architecture of the proposed solution. As shown, the IaaS offers its services as usual to its own Final Users (the upper requests), but it is possible to offer the same services even trough the newly developed mOSAIC Application. In the latter case, the requests may have or not the same format of the underlying provider. In the example, the requests are done with a very simple Restful interfaces trough a JSON request attached to an HTTP POST invocation. When the request arrives, it is interpreted by the mOSAIC Application that performs the local requests in order to start all the VMs needed (*i.e.*, the standard Web Server or the solution presented in Section 4, which includes both the Web Server and the IT Proxy connected trough a dedicated virtual network).

The mOSAIC Application is fully described in Figure 6, where all the components involved in the SLA negotiation and SLA enforcement are proposed. Figure 6 shows the main components of the mOSAIC Application, following the architecture proposed in (Massimiliano Rak, 2011). Our mOSAIC Application consists of four components:

- *SLAgw* that receives the WS-Agreement, stores it in the local storage (signing it in state pending), and forwards it to the decision Cloudlet;
- *SecDecision* that evaluates if the SLA is acceptable (*i.e.*, if the user can assume that role and access that container). The Cloudlet has the role of updating the SLA status (as an example, signing it as accepted or refused) and in case it is accepted, it forwards a request to SecConfigurer;

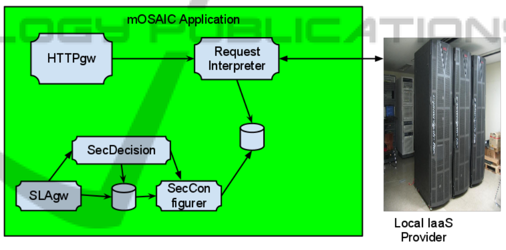

Figure 6: SLA-based mOSAIC Application.

- *SecConfigurer* receives the requests from SecDecision and update a KV store in which there is signed the security level agreed for each user. In this paper, we just assume hat we offer two security level for a Web Server: unprotected and protected. However, it is possible to enrich the offer with a lot of different solutions
- *Request Interpreter* receives the user requests, evaluate the requests, extracting the service requested, checks the KV store in order to identify the SLA agreed, and then performs the request to the local IaaS Provider. In case of unprotected request, just start the Application VM, while in case of protected Web Servers, it start both the Application VM and then the ITmOS VM, writing in the proxy configuration file the right configuration informations. In both cases returns an array of IP and port pairs, but in the first case the array contains just one value, in the second case two of them.

7 OFFERING INTRUSION TOLERANCE TROUGH SLA

The above proposed application enables to negotiate with application users a SLA, described in WS-Agreement, in order to adapt the requests for delivering VMs from the underlying IaaS Provider. In this section, we focus on how such requests can be described in Ws-Agreement and how to enforce the SLA in a VM request invocation.

It is important to point out that a SLA implies that the offering are *granted*, and if not respected some penalties are applied to the peer of the agreement. Following the above approach, we need to understand *what* the proposed IT system is able to grant. Moreover, the solution agreed should be verifiable from the Final User, in order to check the effective respect of the SLA. An additional consideration is important now, the adoption of SLA in offering services as the side effect of imposing to Cloud Provider to clear identify the advantages offered in a measured way.

Identifying the real grants offered in the context of security is a very hard task, being at the state of art very few available solutions able to quantitatively measure the security level of a system in an incontestable way.

The approach we propose to such a problem is pretty simple and can be easily moved to many different contexts: we identify the set of security threats we are able to face and try to model with quantitative parameters such threats. As an example, we can model a flood attack as a possible threat and model it in terms of the number of flooding messages received by the system. Our SLA will be built starting from the list of all the threats we are able to face and the level will be based on the quantitative parameters we have identified to model the security threat. Such an approach can be adopted in each case, in which security threats can be modeled in terms of an attack, and it is possible to build up a quantitative model of such an attack.

For simplicity's sake, in the following we will focus on a single attack against which our IT system work, in order to clarify the approach with a simple example. Being our proxy able to face a larger set of attacks, the real SLA is much more complex than the one proposed here.

The attack we focus, as described in Section 4 is an XDoS attack based on tag nesting. This attack founds on the simple idea that XML validators will be heavily CPU intensive when they have to check a (valid) XML file with a very high number of nested tags.

When an attack takes place the CPU consumption increases even if only few malicious messages are processed by the Web Server. Our solution detects the attack using the following set of information: $(< MeanNumberoftags >, < MeanCPUUsage >, < TimeRange >)$. The detection takes place on interval of time of duration $< TimeRange >$, in such time interval we evaluate if both mean CPU consumption and mean number of tags are over fixed thresholds.

Such a model to detect the attack, that we call *SimpleThreshold*, can be model by using the following simple parameters:

- *CPU Threshold*
- *TAG Threshold*
- *Time Range*

Our IT model is able to grant that the Web Server is protected against an Deeply-Nested XML DoS attacks, detectable with a `SimpleThreshold` technique with parameters `<CPU Threshold>, <TAG Threshold>, <Time Range>`. Our solution grants that if such an attack takes place, there will be no additional CPU usage on the Web Server. The user knows exactly the conditions under which its own Web Server is protected and he is able to adapt the IT Proxy parameters.

It is important to point out that, such an SLA is correct, but very hard to manage for final users, on the other side target users are Web Server administrator with great experience. In future work, we will offer tools that helps in managing such information in a more easy way, using semantic technologies.

In order to offer the SLA in a formal way, we translate such informations in a WS-Agreement template, that can be filled by users in order to negotiate the parameters. We defined a simple schema for management of our security tags, which enable to list the attacks against which the system is protected. The code listed in 1 shown the example of guarantee term for the Web Server's service (that is described in an OCCI compliant way).

Listing 1: XMLDos Attack WS-Agreement.

```
<ws:ServiceDescriptionTerm ws:Name
    ="WEB SERVER REQUEST"
    ws:ServiceName="SET VARIABLE">
        <Compute>
            <architecture>x86</
                architecture>
            <cpuCores>4</cpuCores>
                [...]
              <title>WebServer</
                  title>
          </Compute>
 </ws:ServiceDescriptionTerm>
[...]
```

```
<wsag:GuaranteeTerm wsag:Name="ITS
    " wsag:ServiceScope="WEB
    SERVER REQUEST" Obligated:"
    Provider">
 <wsag:ServiceLevelObjectives>
   <wsag:KPITarget>
     <wsag:KPIName>XML DoS</
         wsag:KPIName>
     <wsag:Target>
       <itsag:Attack name="Nested
           TAG" />
       <itasg:Detection name="
           SimpleThreshold">
         <itsag:Parameter name="
             CPUThreshold" value="
             90" unit="percentage"
             />
         <itsag:Parameter name="
             TAGThreshold" value="
             20" unit="number"/>
         <itsag:Parameter name="
             TimeRange" value="5"
             unit="minutes"/>
       </itasg:Detection>
       <itsag:Reaction time="120"
           unit="minutes"/>
       <itsag:Description link="
           http://www.mosaic-cloud
           .eu/ITS/Attacks/
           NestedTag" />
     </wsag:Target>
   </wsag:KPITarget>
   </wsag:ServiceLevelObjectives>
</wsag:GuaranteeTerm>
```

Our solution enables description of attacks, following the proposed approach just in terms of few parameters:

- *Attack* just contains the name of the attack.
- *Description* has several attributes, including a link to a full, the used language, the description of the attack and of the possible (supported) detection systems.
- *Detection* has an attribute that identifies the supported detection method and contains the list of parameters needed to evaluate the detection model.
- *Reaction* has an attribute, the time needed to react, which means that the system may have some side effects of the attacks for that interval of time.

Such a guarantee term grants the user that each attack listed as KPITarget and detectable with the listed detection methods will not affect the performances of the target system. It is up to the IaaS Provider identify the penalties to be paid in the case in which the condition is not respected.

8 RELATED WORK

To the best of our knowledge not much work has been done in the area of configuring security requirements specified through WS-Agreement documents. Karjoth et *al*. (Karjoth et al., 2006) introduce the concept of Service-Oriented Assurance (SOAS). SOAS is a new paradigm defining security as an integral part of service-oriented architectures. It provides a framework in which services articulate their offered security assurances as well as assess the security of their sub-services. Products and services with well-specified and verifiable assurances provide guarantees about their security properties. SOAS enables discovery of sub-services with the right level of security. SOAS adds security providing assurances (an assurance is a statement about the properties of a component or service) as part of the SLA negotiation process. Smith et *al*. (Smith et al., 2007) present a WS-Agreement approach for a fine grained security configuration mechanism to allow an optimization of application performance based on specific security requirements. They present an approach to optimize Grid application performance by tuning service and job security settings based on user supplied WS-Agreement specification. WS-Agreement describes security requirements and capabilities in addition to the traditional WS-Negotiation attributes such as computational needs, quality-of-service (QoS) and pricing. Brandic et *al*. (Brandic et al., 2008) present advanced QoS methods for meta-negotiations and SLA-mappings in Grid workflows. They approach the gap between existing QoS methods and Grid workflows by proposing an architecture for Grid workflow management with components for meta-negotiations and SLA-mappings. Meta-negotiations are defined by means of a document, where each participant may express, for example, the pre-requisites to be satisfied for a negotiation, the supported negotiation protocols and document languages for the specification of SLAs. In the pre-requisites there is the element ¡security¿ that specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation. With SLA-mappings, they eliminate semantic inconsistencies between consumer's and provider's SLA template. They present an architecture for the management of meta-negotiation documents and SLA-mappings and incorporate that architecture into a Grid workflow management tool.

9 CONCLUSIONS AND FUTURE WORKS

In this paper, we have shown how it is possible, using a Cloud-oriented API derived from the mOSAIC project, to build up a SLA-oriented Cloud application, which enables the management of security features related to Intrusion Tolerance against XMl Denial of Services attacks to an Infrastructure as a Service (IaaS) Cloud Provider. The application that enables SLA management is built in order to receive a WS-Agreement file containing a description of the security features. We proposed a simple schema for description of the guarantees offered by the system to the users against DoS attacks. Once the user has obtained an agreement with the SLA management system, his requests will be fulfilled following the required SLA and the services will be transparently enriched with security features. In our case study, we enrich a Web Server with an Intrusion Tolerance system that grants against a well defined set of attacks. This work is one of the steps we are doing in the direction of offering security features in terms of Service Level Agreement, trough the adoption of the mOSAIC SLA architecture. In future steps, we will enrich the set of attacks our solutions will be able to face and try to offer tools to help users to automatically setup a detailed SLA filled for his own needs.

ACKNOWLEDGEMENTS

This research is partially supported by FP7-ICT-2009-5-256910 (mOSAIC) and by MIUR funded project “Cloud@Home: A new enahnced paradigm”

REFERENCES

Ganglia, a scalable distributed monitoring system for high-performance computing systems.

Squid: an open source fully-featured http/1.0 proxy.

Tpc benchmark w (tpc-w), a transactional web benchmark.

A. K. Caglayan, P. R. L. and Eckhardt, D. E. (1989). A theoretical investigation of generalized voters for redundant system. In *The Nineteenth International Symposium on Fault-Tolerant Computing*, pages 444–451.

Brandic, I., Music, D., Dustdar, S., Venugopal, S., and Buyya, R. (2008). Advanced qos methods for grid workflows based on meta-negotiations and sla-mappings. *2008 Third Workshop on Workflows in Support of LargeScale Science*.

D. Heimbigner., J. K. and Wolf, A. (2002). The willow architecture: Comprehensive survivability for large-scale distributed applications. In *The Intrusion Tolerant System Workshop*, pages 71–78.

Ficco, M. (2010). Achieving security by intrusion-tolerance based on event correlation. *International Journal of Network Protocols and Algorithms*, 2, num. 3:70–84.

Ficco, M. and Rak, M. (2011). Intrusion tolerant approach for denial of service attacks to web services. In *The 1st International Conference on Data Compression, Communications and Processing (CCP 2011)*, pages 285–292.

Karjoth, G., Pfitzmann, B., Schunter, M., and Waidner, M. (2006). Service-oriented assurance, comprehensive security by explicit assurances. In Gollmann, D., Massacci, F., and Yautsiukhin, A., editors, *Quality of Protection*, volume 23 of *Advances in Information Security*, pages 13–24. Springer US.

Marsh, M. A. and Schneider, F. B. (2004). Codex: A robust and secure secret distribution system. In *IEEE Trans. on Dependable and Secure Computing*, volume 1, pages 34–47.

Massimiliano Rak, Salvatore Venticinque, R. A. B. D. M. (2011). User centric service level management in mosaic application. In Press, I., editor, *Europar 2011 Workshop*.

mOSAIC Project (2010). mosaic: Open source api and platform for multiple clouds. http://www.mosaic-cloud.eu.

N. F. Neves, P. S. and Verissimo, P. (2006). Proactive resilience through architectural hybridization. In *The ACM Symp. on AppliedComputing (SAC'06)*.

P. Kouznetsov, A. H. and Druschel, P. (2006). The case for byzantine fault detection. In *The 2nd Workshop on Hot Topics in System Dependability*.

R. Mista, D. Bakken C., D. A. and Medidi, M. (2002). Mr-fusion: A programmable data fusion middleware subsystem with a tunable statistical profiling service. In *The Int. Conference on Dependable Systems and Network (DSN-2002)*, pages 273–278.

Rak, M., Liccardo, L., and Aversa, R. (2011). A sla-based interface for security management in cloud and grid integrations. In Abraham, A. et al., editors, *Proceedings of the 2011 7th International Conference on Information Assurance and Security (IAS)*. IEEE Press.

Smith, M., Schmidt, M., Fallenbeck, N., Schridde, C., and Freisleben, B. (2007). Optimising Security Configurations with Service Level Agreements. In *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA 2007)*, pages 367–381. IEEE Press.

van Sinderen F. Leymann, I. I. M., – Science, B. S. S., and Publications, T., editors (2011). *Towards a cross platform Cloud API. Components for Cloud Federation*.