A Cloud Application for Security Service Level Agreement Evaluation

Valentina Casola[1], Massimiliano Rak[2] and Giuseppe Alfieri[2]
[1]*DIETI, University of Naples Federico II, Naples, Italy*
[2]*DIII, Second University of Naples, Aversa, Italy*

Keywords: Cloud Computing, Security, Cloud Application, SPECS, Negotiation, SLA.

Abstract: Cloud security is today considered one of the main limits to the adoption of Cloud Computing. Academic works and the Cloud community (e.g., work-groups at the European Network and Information Security Agency, ENISA) have stated that specifying security parameters in Service Level Agreements actually enables the establishment of a common semantic in order to model security among users and Cloud Service providers (CSPs). However, despite the state of the art efforts aiming at building and representing Cloud SecLAs there is still a gap on the techniques to reason about them. Moreover a lot of activities are being carrying out to clearly state which are the parameters to be shared, their meanings and how they affect service provisioning. In this paper we propose to build up a cloud application that is able to offer Security level Evaluation based on SLA expressed in many different ways. Such application can be offered as a service by Third Parties in order to help customers to evaluate the offerings from providers. Furthermore it can be used to help customers to negotiate security parameters in a Multi-Cloud system and perform Cloud brokering on the basis of a quantitative evaluation of security parameters.

1 INTRODUCTION

Cloud security is today considered one of the main obstacle to the widespread adoption of Cloud Computing. In this paradigm, due to the self-service on-demand characteristic, all data and servers reside *on the cloud* and charged on a pay-per-use basis. Indeed, this model brings great advantages from a business point of view, as there are no maintenance and start-up costs for any infrastructures, but there is a strong negative perception of *loss of control* over the data and resources.

To face security requirements in the Cloud, early academic works like (Kandukuri B.R.,et. al., 2009) and the Cloud Community (e.g., work-groups at the European Network and Information Security Agency (ENISA) (Dekker M. and Hogben G.,2011)) stated that specifying security parameters in Service Level Agreements (referred as *Security Level Agreements or SecLAs* over this paper) actually enables the establishment of a common semantic in order to model security among users and CSPs.

However, despite the state of the art efforts aiming at building and representing Cloud SecLAs (e.g., the CSAs SLA and PLA working groups (Cloud Security Alliance, 2012)), there is still a gap on the techniques to reason about them.

Indeed, a lot of activities are carried out to clearly state which are the parameters to be included in the SecLAs, their meanings and how they affect service provisioning. One of the biggest problems that arises, even after Security parameters are clearly specified, is the needing to help comparing and choosing different providers on the basis of such parameters: security features cover an incredible amount of different aspects of the systems and the customers, often, are not security experts, even if they have specific security constraints. As an example, let us consider a health center that wants to use the Cloud to maintain a backup of its data. The information to be stored have privacy and availability constraints, moreover in many countries the law imposes constraints on data localization. The advantage of using the cloud, which can grant availability and in given conditions privacy constraints, is annihilated by the absence of clear grants on such security grants. Even if CSPs offer SLAs specifying the security grants, it will be hard for the customer to clearly compare the offerings from different providers that offer the same service in different ways, with different services and expressing the grants through different policies and SLAs.

The work proposed in this paper born in the con-

Casola V., Rak M. and Alfieri G..
A Cloud Application for Security Service Level Agreement Evaluation.
DOI: 10.5220/0004858702990307
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 299-307
ISBN: 978-989-758-019-2
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

text of the SPECS (SPE,) FP7 project, which aims at building a platform able to offer security services to Customers for Multi-Cloud environments. We propose a cloud application that is able to offer Security Level Evaluation over SLAs expressed in many different ways. The evaluation founds on the REM methodology, which aims at enabling a quantitative evaluation of security policy. Such application is offered as-a-service by a Third Party in order to help customers to evaluate the offerings from providers. Moreover it can be used to help customers to negotiate security parameters in a Multi-Cloud system as proposed in (Liccardo et al., 2012; Amato et al., 2012).

In order to evaluate Cloud providers, we will focus on the security mechanisms provided to end users, demonstrating that it is possible to use a standard language to represent security parameters and enabling the evaluation of different providers.

The reminder of this paper is organized as follows, next sections describe the key concepts to evaluate and compare different cloud providers and the requirements for the development of a security evaluation application; we will outline the background related to this paper, i.e. the mOSAIC Platofrm with the API used to develop the Cloud Application and the REM evaluation methodology. Section 4 describes how to build policies from the security mechanisms exposed to end users, in order to make a security evaluation. The following section, 5 describes the application architecture and its reusable components. Paper ends with section 6 dedicated to conclusions and future works.

2 PROBLEM STATEMENT AND APPROACH

As outlined in the previous section, our goal is the development of an application able to help Customers to evaluate and compare different Cloud Service Providers (CSP) on the basis of the security they are able to grant. In a companion paper we proposed to evaluate providers on the basis of the STAR repository, maintained by CSA, which contains the answers that CSPs has given to a complex questionnaire dedicated to Cloud Security. In this paper, instead, we will focus on the real security mechanisms the CSPs expose to Customers and the implication they may have on the security effectively granted.

We analysed different cloud providers (Amazon EC2, Google Compute Engine, GoGrid), they offer similar features (from customers point of view they are mostly equivalent) but they use completely different technologies and techniques in order to enforce security mechanisms into the service invocation mechanism. As an example, Amazon Web Services (aws) uses SOAP messages to control Amazon Ec2 services, they are secured through a set of HMAC applied and evaluated to each message. GoGrid, instead, performs an authentication through a dedicated authentication server, which issues a security token to be attached to each Web Service invocation. Such different choices have impact both on performances and on the level of security offered.

Even if such features are clearly visible to end users, so they can be explicitly evaluated by a third party, the high number of different technologies offered by different providers and the lack of a clear description of the adopted approaches, make such comparison a very difficult task, usually performed by experts or, intuitively, by Customers.

We propose to follow a more systematic approach, adopting existing standards to represent the way in which security mechanisms are described and services are invoked: WS-Policy(Bajaj et al., 2006) and WS-Security-Policy(Della-Libera et al., 2002).

WS-Policy is a container language that allows to create a XML Policy based on Assertions. It mainly allows the use of a group of statements that are alternatives to each other (also the empty alternative) (*ExactlyOne* statement) or a group of statements that must be met (even 0 assertions) (*All* statement). Ws-Security-Policy is a language that defines a set of Security Assertions to be used within Ws-Policy. These Assertions are useful to specify both what is required and what should be guaranteed on the messages and communication protocols in terms of security when using a particular service.

We want to point out that the semantic of Security Assertions made available by Ws-Security-Policy is often directly linked to the way in which Security is managed in the SOAP WebServices (namely in the structure of the SOAP messages described in the standard Ws-Security). Since, de facto, a REST message does not exist, there is not yet a standard language to describe what is required or offered in terms of Security on REST communications.

Such standard representation offers a common basis to represent the security mechanisms adopted by the CSPs and this will facilitate the adoption of the REM methodology to evaluate and compare the offered solutions. In particular, in order to use the REM, we will build a common template, whose instances will correspond to the mechanisms adopted by the different providers.

In synthesis the technique we propose is based on the following steps:

1. **Description.** Collect WS-SecurityPolicy repre-

sentations of the security mechanisms offered by different CSPs based on a standard template we prepared. Such description can be offered directly by a provider (rarely) or defined by third parties.

2. **Comparison.** Using the REM methodology we quantitatively measure each policy, offering a clear basis to compare the different offerings of providers. Thanks to the flexibility of the REM methodology different customers can specify different weights and evaluation criteria, obtaining customized evaluations.

The main goal is the implementation of a cloud application to provide the security evaluation as-a-service. Before illustrating the details of such technique, in the next section, we will provide a technical background to describe the the mOSAIC framework to easily develop the cloud application and the REM methodology.

3 TECHNICAL BACKGROUND

The solution we propose to build up a cloud application to evaluate providers, founds on two existing solutions, these will be summarized in this section. In particular, in the following we will detail only the key ideas needed to fully understand the proposed approach: the mOSAIC Framework to easily implement cloud applications and the REM evaluation methodology to evaluate security. For further details the interested reader may refer to the related papers.

3.1 The mOSAIC Framework

The mOSAIC framework aims at providing a simple way to develop cloud applications (Petcu et al., 2011c; Petcu et al., 2011b; Petcu et al., 2011a); the target user for the mOSAIC solution is the application developer (*mOSAIC user*). In mOSAIC, a cloud application is structured as a set of components running on cloud resources (i.e., on resources leased by a cloud provider) and able to communicate with each other. Cloud applications are often provided in the form of Software-as-a-Service, and can also be accessed/used by users other than the mOSAIC developer (i.e., by *final users*). In this case, the mOSAIC user acts as service provider for final users.

The mOSAIC framework is composed of a few stand-alone components. Among them, the most important roles are played by the *Platform* and the *Cloud Agency*. The first one (mOSAIC Platform) enables the execution of applications developed using the mOSAIC API. The second one (Cloud Agency) acts as a provisioning system, brokering resources from a cloud provider, or even from a federation of cloud providers.

mOSAIC can be used in three different scenarios:

- when a developer wishes to develop an application not tied to a particular cloud provider;
- when an infrastructure provider aims at offering "enhanced" cloud services in the form of SaaS;
- when a final user (e.g., a scientist) wishes to execute his own application in the cloud because he needs processing power.

A mOSAIC application is built up as a collection of interconnected *mOSAIC components*. Components may be (i) core components, i.e., predefined helper tools offered by the mOSAIC platform for performing common tasks, (ii) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (iii) *cloudlets* developed using the mOSAIC API and running in a *Cloudlet Container*. mOSAIC cloudlets are stateless, and developed following an event-driven asynchronous approach (Petcu et al., 2011c; Petcu et al., 2011a).

The mOSAIC platform offers ready-to-use components such as queuing systems (*rabbitmq* and *zeroMQ*), which are used for component communications, or an HTTP gateway, which accepts HTTP requests and forwards them to application queues, NO-SQL storage systems (as KV store and columnar databases). mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the *Platform Manager*, which makes it possible 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 or to decrease the number of virtual machines dedicated to the mOSAIC application, which will scale in and out automatically.

Defining a new application is very easy: a cloud application is described in a file named *Application Descriptor*, it lists all the components and the cloud resources needed to enable their communication. A mOSAIC developer has both the role of developing new components and of writing application descriptors that connect them together.

3.2 Evaluation of Security

The methodology implemented to evaluate security is the Reference Evaluation Model (REM) (Casola et al., 2007a; Casola et al., 2007b), it defines how to express in a rigorous way the security policy, how

to evaluate a formalized policy, and how to state the provided security level. Any policy is represented through a tree, which contains all the policy provisions (intermediate nodes and leaves).

In Figure 1 the three methodology phases are shown: **Policy Structuring**, **Policy Formalization** and **Policy Evaluation**:

The goal of the **Structuring** phase is to associate an enumerative and ordered data type K_i to the n leave-provisions of the policy. A policy space "P" is defined as P $=K_1 \times K_2 \times \ldots \times K_n$, i.e. the vectorial product of the n provisions K_i. The space is defined according to a policy template that strongly depends on the application context.
In the **Formalization** phase the policy space "P" is turned into an homogeneous space "PS". This transformation is accomplished by a normalization and clusterization process which allows to associate a Local Security Level (LSL) to each provision; after that the provisions may be compared by comparing their LSLs.

The goal of the **Evaluation** phase is to pre-process the "PS" vector of LSLs and evaluate the so called Global Security Level L_{Px} associated to the policy P_x. The Global Security Level has been defined on the basis of an Euclidean distance among matrices and some reference levels:

$$L_{Px} = \begin{cases} L_0 & iff d_{x0} \leq d_{10} \\ L_1 & iff d_{10} < d_{x0} < d_{20} \\ L_2 & iff d_{20} < d_{x0} < d_{30} \\ L_3 & iff d_{30} < d_{x0} < d_{40} \\ L_4 & iff d_{40} \leq d_{x0} \end{cases}$$

where $d_{i,0}$ are the distances among the references and the origin of the metric space (denoted as $\emptyset$). This function gives a numerical result to the security; indeed the idea is to evaluate the security associated to an infrastructure through the evaluation of its security policy.

The GSL is a measure of the security provided by an infrastructure according to its security policy; it is obtained by formalizing the process that is manually performed by security experts while trying to extend trust to other domains. The details of the methodology are out of the scope of this paper, and they can be found in (Casola et al., 2007a).

4 A PRACTICAL EXAMPLE

In order to show in practice how the proposed approach works, in this section we propose the WS Security Policy template and the instances for the three generic providers.

The apply the REM to three Cloud Service Providers, denoted by **ProviderA, ProviderB** and **ProviderC** we need to describe the security mechanisms according to the WS-* models that we created.

To better understand the policy, we first summarize the behaviour of the three providers.
ProviderA applies a redundant security mechanism, it supports HTTPS for the *Authentication of the Service* and the *Message Confidentiality*. At Message Layer it supports HMAC to assure the identity of the sender (with a Secret Key for Encryption) and it also signs the body of the message ensuring the message integrity (operation redundant because it is on HTTPS).
ProviderB supports HTTPS for the REST call invocations, offering:

- *Authentication of the Service* : The user knows that he is sending the message to the right destination.
- *Confidentiality of messages* : No one can be aware of content of a sent message, nor alter it.

At Message Layer it attaches an `AuthenticationToken` that is created on the basis of a shared secret (password), concatenates the secret string to a string already containing a username, a timestamp, the user agent (http field) and then applies the MD5 hash function. The receiver performs the same operations before accepting the request.

ProviderC basically offers the same mechanism of ProviderB : HTTPS and an `AuthenticationToken` for the User Authentication. The only positive difference is that it uses SHA-1 instead of MD5.

4.1 Building the Common Template

In order to represent such features, Ws-SecurityPolicy supports three main types of containers:

- `Transport Binding` the security mechanism is enforced at transport layer, it may contain different subfields as `HttpsToken` and `TrasportToken`;
- `Symmetric Binding` the security mechanism is enforced at SOAP level as specified in Ws-Security, by using symmetric keys; it also allows to express other types of tokens as `SignatureToken` and `EncryptionToken`;
- `AsymmetricBinding` the security mechanism is enforced at SOAP level as specified in WS-Security, by using asymmetric keys.

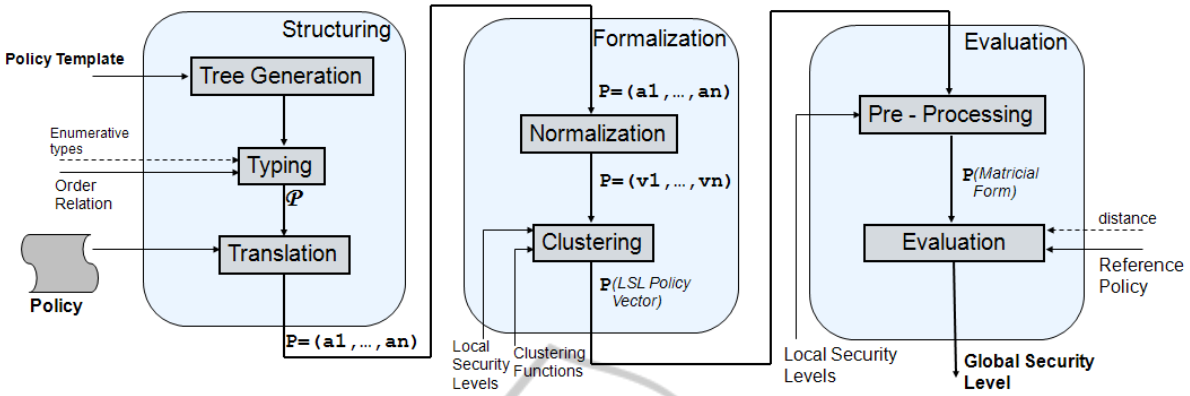

Figure 1: Phases of the evaluation methodology.

In our template we will focus only on the first two containers.

We defined a `TransportBinding` container, giving to the sequence `TransportBinding`/`TransportToken` the semantic meaning of Security at the Transport Layer, and to the `HttpsToken` subfield the semantic meaning of support of the HTTPS protocol.

We gave to the sequence `SymmetricBinding` / `SignatureToken` the semantic meaning of security at the Message Layer (Http message), where there may be some signed message parts. We did not add the sequence `SymmetricBinding`/`EncryptionToken`, because none of the analyzed Providers provides confidentiality on the message at the message layer. Furthermore, in `SignatureToken` we added predetermined fields including `AlgorithmSuite` and *SignedParts*.

In particular, among the possible `AlgorithmSuite` alternatives we used: `Md5`, `SHA1`, `HMAC`, `No` (from `XMLSignature` Specification).

Among the possible *SignedParts* subfields we added :

- `TimeStamp` against replay attacks (WS-Trust Specification);
- `Password` obtaining thus Authenticity of the Sender (Ws-Security Specification);
- `Body of Message` obtaining thus Integrity of the message;

By combining WS-Policy and WS-SecurityPolicy to define the template, we built the template presented in the following listing:

Listing 1: The Ws Security Policy Template.

```
<?xml version="1.0" encoding="utf-8"
    ?>
<wsp:Policy xmlns:wsp="http://www.w3
    .org/ns/ws-policy"
xmlns:sp="http://docs.oasis-open.org
    /ws-sx/ws-securitypolicy/200702"
xmlns:ds="http://www.w3.org/2000/09/
    xmldsig#"
xmlns:wsse="http://docs.oasis-open.
    org/wss/oasis-wss-wssecurity
secext-1.1.xsd"
xmlns:wsu="http://schemas.xmlsoap./
    ws/2005/trust
/Ws-Trust.xsd">
<wsp:ExactlyOne>
<wsp:All>
<sp:TransportBinding>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:TransportToken>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:HttpsToken sp:IncludeToken="
    http://docs.oasis
open.org/ws-sx/ws-securitypolicy
    /200702
    IncludeTokenAlwaysToRecipient"
wsp:Optional="true"/>
<!--NotPresent < Present-->
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:TransportToken>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:TransportBinding>
<sp:SymmetricBinding>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:SignatureToken sp:IncludeToken="
    http://docs.oasis-
```

```
open.org/ws-sx/ws-securitypolicy
    /200702
    IncludeTokenAlwaysToRecipient">
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:SignedParts>
<wsu:Timestamp wsp:optional="true"/>
<!--NotPresent < Present-->
<wsp:Body wsp:optional="true"/>
<!--NotPresent < Present-->
<wsse:Password wsp:optional="true"/>
<!--NotPresent < Present-->
</sp:SignedParts>
<sp:AlgorithmSuite>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<wsp:ExactlyOne>
<ds:SignatureMethod ds:Algorithm=
"http://www.w3.org/2000/09/xmldsig#
    hmac-sha1"/>
<ds:SignatureMethod ds:Algorithm=
"http://www.w3.org/2000/09/xmldsig#
    sha1"/>
<ds:SignatureMethod ds:Algorithm=
"http://www.w3.org/2000/09/xmldsig#
    md5"/>
<!--md5 < sha1 < hmac-sha1-->
</wsp:ExactlyOne>
</wsp:All>
<wsp:All>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:AlgorithmSuite>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:SignatureToken>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:SymmetricBinding>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Note that a security expert can perform different kind of analysis on the policies. For example, being ProviderA the only one supporting HTTPS, one will expect it will perform better than the others. On the other hand, ProviderB and ProviderC do not ensure the integrity of the message, nor its confidentiality. If an attacker applies the man in the middle technique, he can replace the message with a different body. At that point even the identity of the sender loses meaning. If the attacker is not able to make the man in the middle technique but only sniff the messages, he could retrieve the md5 hash and crack it in seconds obtaining the user password. The SHA1 is stronger than md5 and he can only guess the password by a bruteforce attack. If the communication is on HTTPS, the attacker should perform a bruteforce attack and the longer is the password, more secure will be the mechanism.

About ProviderA, the attacker can not use the man in the middle technique, because the user has signed the content with HMAC, but he can observe it: the Security is in the difficulty with which the attacker can guess the key with a offline bruteforce attack. In particular, with HTTPS, the attacker should force with a brute force online attack which is impractical. From these considerations, it could be said that: the TransportBinding/TransportToken branch should leveling out differences however it makes appear the ProviderA the most secure. On the SymmetricBinding/SignatureToken branch the ProviderA should provide a Security Level quite high compared to the others.

Once made these preliminary actions, and applying the methodology, the results we obtain in the evaluation give three different Global Security Level. For brevity's sake we cannot report the whole evaluation process but the final result is:

- *P*roviderA GSL := 3;
- *P*roviderC GSL := 2.67;
- *P*roviderB GSL := 2.5;

Such results are coherent with an intuitive analysis of the behaviours described at start of the section.

Even if such analysis can be done by an independent expert, the approach we propose offers a clear step toward an automatization of the procedure, assuming a collection of Ws-Security-Policy descriptions of the services offered by different Cloud Providers, which can be done even by third parties.

5 THE SECURITY EVALUATION APPLICATION

In this section we will illustrate the functionalities that the cloud Evaluation Application can offer to a security evaluator.

The evaluation of the Security Level of a SecLA is an action that can be carried out by Security Administrators who will have the task of managing their Platform. They can be Developers who want to test an application or Methodology Researchers who want to use the application to run tests in order to study possible alternatives in the evaluation methodology field. The role of the security evaluator can be clearly hired

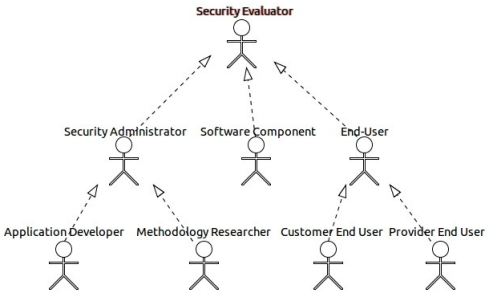

Figure 2: The Application Users.

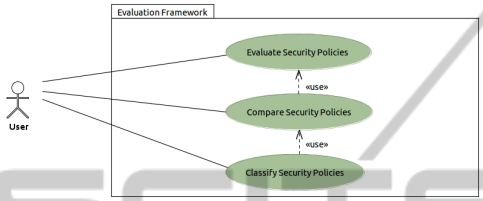

Figure 3: The Application Use Case diagram.

by End Users who use the application to choose a Provider. It in turn may be classified as a Customer End User, or as a Provider End User who wants to use the service to test the Security Level of their standards before entering the market. Figure 2 clarifies the relationships among all these actors.

The Application functionalities are reported in Figure 3 with the UML Use Case diagram.

The user can request the evaluation of the security requirements according to the REM Methodology (Casola et al., 2007a), compare them with other security policies (eventually supplied by cloud providers) and classify the results in order to choose the best solution.

In the next sections we will illustrate at first a Java library that enables the application of the REM Methodology, then we will present the mOSAIC components that enable its execution in a cloud environment.

5.1 The Java Library

The *Java Library* consists of a set of *Java Components* organized in packages.

The *Representation Package* supplies the data structures and all the mechanisms required to represent data. In particular, the *parsers package* contains a set of Java Classes to transform security policies expressed in an interoperable format (XML, for information tranfers) into a format suitable for the elaborations (Java Jung Tree) (Jun,) being the latter defined in the *data_structure package*. The ”parsers” and ”data_structures” packages can be extended with new Java Classes in case a new type of Security Policy (corresponding to a new template) should be processed by the system. The *distance_evaluator package* implements the specific evaluation technique. The *difference_evaluator* package provides classes able to compare two instances of Security Policies. These packages can be extended in order to implement new evaluation techniques which extend those already defined.

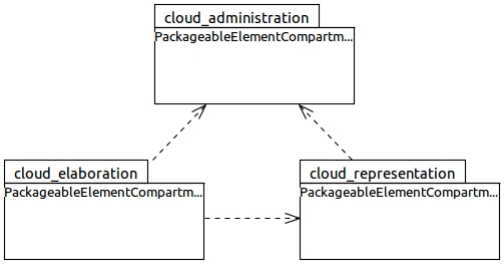

Figure 4: Cloudlets organization.

5.2 The Cloud Application Components

The Cloud application, based on the mOSAIC framework briefly described in section 3.1 is composed of a set of ready-to-use Cloudlets, that embeds the Java Components. The Cloudlets can be reused and composed by a developer in order to realize its custom evaluation application.

Figure 4, shows the implemented cloudlets organized by packages, depending on offered functionalities. The *cloud_representation* package contains the cloudlets offering to developers and final users a way to convert their Security Policies from an human readable format to a platform readable representation (and vice versa), ready to be elaborated. These cloudlets implement, using the mOSAIC API, the classes defined in the *representation package* of the Java Library; they are:

- *QSTRepresentation* (converts and stores the policy from readable format via user interface to internal representation);
- *WeightsRepresentation* (converts and stores informations about possible weights from user interface to internal representation);
- *ResultRepresentation* (converts and stores results internally represented to an human readable format from the interface);

The *cloud_elaboration* package contains the cloudlets offering to developers the Security Policy evaluation functionalities, with likelihood to choose between the various elaboration techniques. The cloudlets contained in this package therefore implement, using the mOSAIC API, the classes defined

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

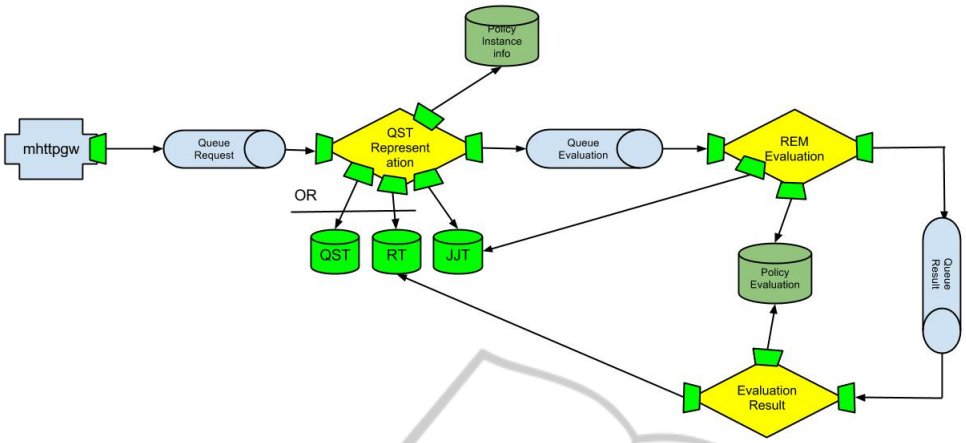

Figure 5: The mOSAIC Evaluation cloudlets.

in the *elaboration package* of the Java Evaluation Framework.

The cloudlets contained in the package are:

- *REMEvaluation* (offers the evaluation functionalities. It implements, using the mOSAIC API, the "distance_evaluator" package of Java Evaluation Framework);
- *WeightsEvaluation* (applies the weights to security policies in accordance with defined techniques and the type of representation. It implements using the mOSAIC API the "weights_evaluator" package of the Java Evaluation Framework);
- *RemCompare* (offers comparation functionalities between two instances of a Security Policy. It implements, using the mOSAIC API, the "difference_evaluator" package of the Java Evaluation Framework);
- *Ranking* (rankes the Security Policies conveniently evaluated and compared);

The *cloud_administration* package contains essential cloudlets for the administration, the messages management, and the interoperability of cloudlets defined in the above packages.

Furthermore, in addition to defined packages, a set of *mangement kv-stores* is managed by the system. They keep the necessary informations for the correct execution of the mOSAIC Evaluation Applications.

Figure 5 shows the mOSAIC application that evaluates a Security Policy submitted by an user.

To describe in greater detail the modus-operandi of cloudlets, we proceed to illustrate in detail the operations of two cloudlets of the mosaic application.

The *QSTRepresentation* takes in input a message from an HTTP RESTful interface, extracts data, stores them (eventually) in the QST kv-store, and converts them in two different representations:

- RT (XML representation) (optional, for user retrieval);
- JJT (Java Jung Tree Data Structure representation);

The cloudlet sends the JJT representation of the Security Policy on an exit queue.

The *REMEvaluation* takes in input a message from a the representation cloudlet, containing the JJT representation of a Security Policy, evaluates it and stores the JJT evaluated. The cloudlet can send the JJT list of evaluated Security Policies on an exit queue.

Summarizing, the user defines the istance of his Security Policy (in the QST format) and submits it to the system through an HTTP interface (mhttpgw). This transforms the data into a suitable elaboration format through the QST Representation Cloudlet and sends it to REMEvaluation Cloudlet, which executes the evaluation according to one of techniques described in the early section. The chosen technique can be specified by the user through the same interface. The Evaluation Result Cloudlet stores the result in a manner that it can be referred later by the user.

6 CONCLUSIONS AND FUTURE WORKS

The *loss of control* over the resources is the first cause of the perception of low security in cloud computing. In this paper we made a fist step in the direction of offering to customers a new way to control the level of security offered by provider over the resources they control. We proposed a way to use standard languages

to build up comparable policies that enable customers to compare different provider on the basis of the technologies adopted using an automated and quantitative way. We proposed the adoption of WS Policy and WS Security Policy as standard languages to describe the mechanisms adopted by provider and, in future works, we aims at building a collection of policies describing the technological solutions adopted by different cloud providers. We have shown on real case studies the application of the proposed technique over real cloud providers, showing that the results obtained are coherent with a manual evaluation of the technology adopted. Moreover we proposed a Cloud application able to automatize the process and support the security evaluators to apply the technique. The results in this paper will be improved in the future, building a dedicated framework which suppot the development of custom evaluation application and able to fully exploit the flexibility of the underling methodology, which will be further extended.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

REFERENCES

Jung - java universal network/graph framework toolkit.

Specs project.

Amato, A., Liccardo, L., Rak, M., and Venticinque, S. (2012). Sla negotiation and brokering for sky computing. In *CLOSER*, pages 611–620.

Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., et al. (2006). Web services policy 1.2-framework (ws-policy). *W3C Member Submission*, 25:12.

Casola, V., Mazzeo, A., Mazzocca, N., and Vittorini, V. (2007a). A policy-based methodology for security evaluation: A security metric for public key infrastructures. *Journal of Computer Security*, 15(2):197–229.

Casola, V., Mazzocca, N., Luna, J., Manso, O., and Medina, M. (2007b). Static evaluation of certificate policies for grid pkis interoperability. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 391–399. IEEE.

Della-Libera, G., Gudgin, M., Hallam-Baker, P., Hondo, M., Granqvist, H., Kaler, C., Maruyama, H., McIntosh, M., Nadalin, A., Nagaratnam, N., et al. (2002). Web services security policy language (ws-securitypolicy). *Public Draft Specification (Juli 2005)*.

Liccardo, L., Rak, M., Di Modica, G., and Tomarchio, O. (2012). Ontology-based negotiation of security requirements in cloud. In *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, pages 192–197.

Petcu, D., Craciun, C., Neagul, M., Lazcanotegui, I., and Rak, M. (2011a). Building an interoperability api for sky computing. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 405–411. IEEE.

Petcu, D., Crăciun, C., Neagul, M., Panica, S., Di Martino, B., Venticinque, S., Rak, M., and Aversa, R. (2011b). Architecturing a sky computing platform. In *Towards a Service-Based Internet. ServiceWave 2010 Workshops*, pages 1–13. Springer.

Petcu, D., Craciun, C., and Rak, M. (2011c). Towards a cross platform cloud api. components for cloud federation. In *Procs. 1st International Conference on Cloud Computing and Services Science, SciTePress–Science and Technology Publications, Portugal*, pages 166–169.