SLA-based Secure Cloud Application Development: the SPECS Framework

Valentina Casola, Alessandra De Benedictis
DIETI
University of Naples Federico II
Napoli, Italy
{casolav,alessandra.debenedictis}@unina.it

Massimiliano Rak
DIII
Second University of Naples
Aversa, Italy
massiliano.rak@unina2.it

Umberto Villano
DING
University of Sannio
Benevento, Italy
villano@unisannio.it

Abstract—**The perception of lack of control over resources deployed in the cloud may represent one of the critical factors for an organization to decide to cloudify or not their own services. Furthermore, in spite of the idea of offering security-as-a-service, the development of secure cloud applications requires security skills that can slow down the adoption of the cloud for non-expert users. In the recent years, the concept of Security Service Level Agreements (Security SLA) is assuming a key role in the provisioning of cloud resources. This paper presents the SPECS framework, which enables the development of secure cloud applications covered by a Security SLA. The SPECS framework offers APIs to manage the whole Security SLA life cycle and provides all the functionalities needed to automatize the enforcement of proper security mechanisms and to monitor user-defined security features. The development process of SPECS applications offering security-enhanced services is illustrated, presenting as a real-world case study the provisioning of a secure web server.**

I. Introduction

Currently cloud security is considered one of the enabling factors for the widespread adoption of the cloud computing paradigm. Cloud computing relies on the idea of accessing every kind of resource through an "as-a-service" interface, and on the adoption of a *pay-per-use* business model. According to such model, cloud resources are not permanently assigned to users, and are not under the control of user software, but they are just acquired *on-demand*. As a matter of fact, this behavior is in contrast with the view of security as a *vertical layer*, acting independently of the adopted software stacks and provided with full control over all the resources involved in service delivery.

Cloud Service Providers (CSPs), who are the owners of the physical computing, storage and network resources, manage huge datacenters where security is administered according to common best-practice rules. They provide exactly the same security features, which are simply *the best they can offer*, to a variety of Cloud Service Customers (CSCs) with different security requirements. As an example, consider the case of a web developer that aims at building the web portal of a publicly funded health-care system. Even if the design and deployment of a portal is a technical task, the developer needs to respect a set of security rules, related to the criticality of data managed and to the public role of the health care system. For example, the developer has to grant that data are stored according to predefined policies to ensure integrity and confidentiality. A CSP has no interest in offering a security solution that grants such features to *every* CSC, due to the high involved costs. Moreover, it is not able to differentiate security features on a customer basis. As a result, the web developer cannot simply turn to the cloud approach, and is stimulated to consider alternative solutions.

The above example outlines the gap that exists between CSCs, which look for security offered *as-a-service*, exactly as any other cloud resource, and CSPs, which offer security *as-a-whole*, integrated in the cloud services and transparently granted in the same way for all customers. In such a context, Security Service Level Agreements (SLAs) assume a key role, as they allow, among other things, to declare clearly the security level granted by providers to customers, as well as the constraints posed to both parties (providers and customers). Despite the strong interest recently shown in Security SLAs in the context of both academical research and industry and government-driven initiatives, their concrete adoption is not yet a reality. In 2011, ENISA published a report analyzing the use of security parameters in Cloud SLAs (mostly focused on the EC public sector) [1], which pointed out that, although security was considered by most respondents as a top concern, existing SLAs addressed only availability and other performance-related parameters, while security-related parameters were not included. Since then he situation has not changed, and Security SLAs are still far from being adopted by existing CSPs.

The main goal of the on-going SPECS project[1] is to foster the adoption of Security SLAs by building a framework to develop applications that deliver cloud services controlled by Security SLAs. Every cloud service is covered by a Security SLA that specifies the security grants offered, and that can be negotiated before cloud service delivery. Security features are automatically implemented by the SPECS framework according to the agreed SLA, and can be continuously monitored to verify that the SLA terms are actually respected.

The aim of this paper is illustrate the development of a SPECS application. In Section II, we briefly introduce the SPECS framework and in Section III its SLA model. Section IV shows how a SPECS application looks like, while Section V briefly illustrates the process to be followed for its development. The methodology presented is applied in Section VI, which gives insight into the development steps through a case study. Finally, we present our conclusions and future work in Section VIII.

[1]http://www.specs-project.eu

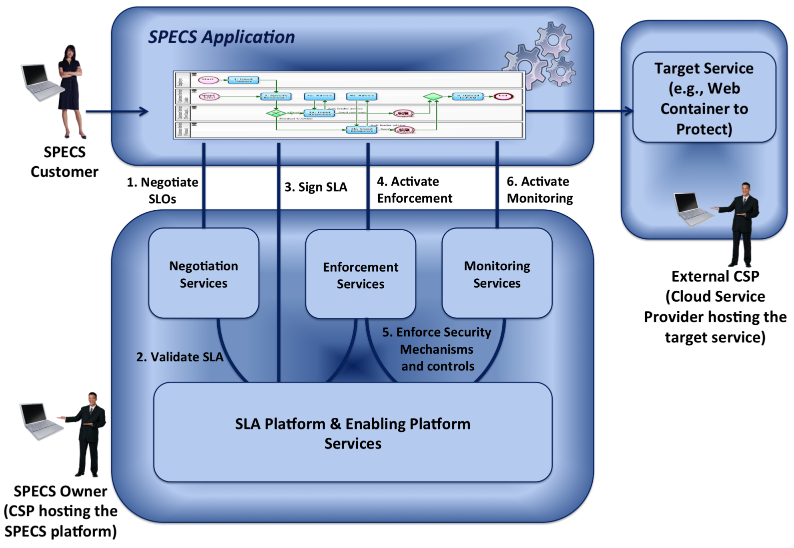
SPECS Application
Target Service (e.g., Web Container to Protect)
SPECS Customer
1. Negotiate SLOs
3. Sign SLA
4. Activate Enforcement
6. Activate Monitoring
External CSP (Cloud Service Provider hosting the target service)
Negotiation Services
Enforcement Services
Monitoring Services
2. Validate SLA
5. Enforce Security Mechanisms and controls
SLA Platform & Enabling Platform Services
SPECS Owner (CSP hosting the SPECS platform)

Fig. 1. Overview of the SPECS architecture

II. The SPECS Framework

The SPECS project aims at designing and implementing a framework for the management of the whole Service Level Agreement life cycle, intended to build applications (SPECS applications) whose security features are stated in and granted by a Security SLA [2], [3].

The SPECS framework provides techniques and tools for: a) enabling user-centric negotiation of security parameters to be included in a Security SLA; b) enforcing an agreed Security SLA by automatically putting in place all security features needed to meet user requirements; c) monitoring in real-time the fulfillment of Security SLAs and notifying both users and CSPs of possible violations; d) reacting and adapting in real-time to fluctuations in the provided level of security (e.g., by applying proper countermeasures in case of an SLA violation).

The following four main parties are involved in the SPECS operation scenario:

- **End-user**, the customer of the cloud services covered by Security SLAs;
- **SPECS Owner**, the provider of the cloud services covered by Security SLAs;
- **External CSP**, an independent (typically public) cloud service provider, which is unaware of the SLAs and offers just basic cloud resources and infrastructural services;
- **Developer**, a cloud service partner that supports the SPECS Owner in the development of SPECS applications and in the delivering of security-enhanced cloud services.

The End-user negotiates his security requirements with the SPECS Owner, who acts as a broker by acquiring resources from External CSPs and by reconfiguring/enriching them in order to fulfill the End-user's requests. This is accomplished by the activation and configuration of suitable security mechanisms, provided in an *as-a-service* mode by the Developer. The security mechanisms are automatically enforced on top of acquired resources, according to what has been agreed in a Security SLA. In the above process, the security-enhanced services are delivered to End-users by a SPECS application, developed and deployed by exploiting the SPECS framework services, and running on top of the SPECS Security Platform-as-a-Service (SPECS PaaS).

In light of the above, the idea is to offer a framework that enables to enrich easily an existing cloud service with Security SLAs, by re-using a set of available security mechanisms and by exploiting a set of services (*Core services*) devoted to the management of the SLA life cycle.

As shown in Figure 1, a SPECS Application orchestrates the SPECS Core services dedicated to Negotiation, Enforcement and Monitoring, respectively, to provide the desired service (referred to as "Target Service" in the picture) to the SPECS Customer (i.e., to the End-user). The Core services run on top of the SPECS Platform, which provides all the functionalities related to the management of Security SLA life cycle and needed to enable the communication among Core modules. In addition to this functionalities, referred to as "SLA Platform services", the SPECS Platform also provides support for developing, deploying, running and managing all SPECS services and related components. These services are referred to as "Enabling Platform services".

In Figure 1, security-related SLOs are negotiated *(Step 1)* based on the SPECS Customer's requirements. A set of compliant offers, each representing a different supply chain to be implemented, is identified with the help of an interoperability layer (represented by the SPECS SLA Platform services), which is also responsible for their validation (e.g., to verify their actual feasibility based on the current system configuration) *(Step 2)*. Of course, given a set of security requirements expressed by the SPECS Customer, multiple

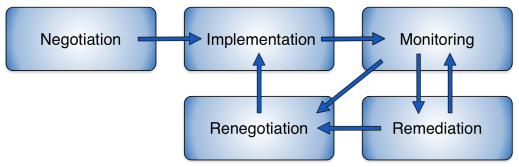

Fig. 2. The SLA Life Cycle phases

supply chains may be identified, each characterized by its own cost and associated security level. The resulting supply chains may be ranked to help the SPECS Customer choose the desired configuration. The agreed terms are included in a Security SLA that is signed by the SPECS Customer and the SPECS Owner *(Step 3)*. Afterwards, the agreement is implemented through the Enforcement services, which acquire resources from external CSPs and activate suitable components that provide, in an *as-a-service* fashion, the security capabilities needed to fulfill the SLOs included in the signed Security SLA *(Steps 4 and 5)*. At the same time, suitable services and agents are activated for monitoring the specific parameters included in the Security SLA *(Step 6)*. Monitoring data are collected by the SPECS Monitoring module and analyzed based on a monitoring policy: if an incoming or occurred violation of the signed SLA is suspected, they are forwarded to the Enforcement module, which performs a diagnosis. In the case of an actual violation, suitable countermeasures may be adopted, consisting in re-configuring the service being delivered, or in applying remediation actions.

III. The Security SLA model

As discussed in the previous section, the SPECS approach for *Security-as-a-Service* provisioning relies upon the idea that each cloud service is covered by a Security SLA, which specifies security-related terms and conditions about delivered services, and that the cloud service delivery can be controlled according to the security features negotiated with the customer. In order to manage such an approach, we proposed a Security SLA life cycle, built according to current standards on cloud SLAs (WS-Agreement [4], ISO19086 [5]). Our SLA life cycle is made of five phases (Figure 2): *Negotiation*, *Implementation*, *Monitoring*, *Remediation* and *Renegotiation*.

During the *Negotiation* phase, a cloud service customer and a cloud service provider carry out a (possibly) iterative process aimed at finding an agreement that defines their relationship as regards the delivery of a service. During the *Implementation*, the CSP provisions and operates the cloud service, but also sets up the processes needed for the management and monitoring of the cloud service, the report of possible failures and the claim of remedies. After the implementation of an SLA, the *Monitoring* phase takes place. If any SLA violation occurs, i.e., if one of the agreed terms of the SLA is not respected, the cloud service customer may be entitled to a remedy (*Remediation* phase). Remedies can take different forms, such as refunds on charges, free services or other forms of compensation. Finally, during the *Monitoring* and/or *Remediation* phases, either the cloud service customer or the cloud service provider may require a change in the SLA. This may lead to a *Re-negotiation* phase, changing the original SLA terms. The interested readers are referred to [6] for a deeper analysis of the SLA life cycle and the description of REST API developed within the SPECS project, and to [7] for an illustration of some of the tools used in SPECS to monitor the SLA during the execution of a cloud service.

The life cycle introduced above makes it possible to control cloud services according to SLA phases (and states). However, it is also necessary a conceptual model to represent security concepts in the SLA. We developed a Security SLA model that enables to clearly state security in cloud services, also taking into account End-user requirements.

In addition to the information about involved parties and provided services, a Security SLA must include security-related guarantees. Unfortunately, despite the strong interest in security and the existing efforts towards standardization, a shared format for Security SLAs that includes the representation of security attributes and security guarantees is not available. Our machine-readable format relies on WS-Agreement (WSAG) [4], born in the context of Grid computing, which is currently the only standard supporting both a formal representation of SLAs and a protocol for their automation and has been recently widely adopted to represent SLAs in cloud environments. In its original definition, WS-Agreement does not allow to specify security-related attributes. Hence, aiming at managing automatically the Security SLA life cycle, we have introduced a Security SLA model and a machine-readable format based on the WS-Agreement's XML schema and extended it with all security-related information.

A high-level view of the proposed SPECS Security SLA model is represented in the UML diagram in Figure 3. In the figure, we report the relationships among the SPECS application, the Target Service, the SLA model and the other main software components involved. As shown in the figure, a SPECS application is linked to a Security SLA template, which summarizes all the features that can be offered to End-users through the application, under the guarantees stated in a Security SLA. When the End-users negotiate security features with a SPECS application, they basically select a subset of the available features and obtain corresponding SLA Offers built according to what can be actually delivered. Hence, an SLA Offer can be seen as an instance of the set of possible SLAs represented by a template.

A Security SLA is provided with basic information such as the agreement name and context data (including the agreement initiator and responder). The concrete description of the SLA is made of `service description terms`, which describe the offered service, and of `service guarantee terms`, which describe the grants guaranteed in the form of Service Level Objectives. Cloud and security-related information are found in a dedicated security-based domain-specific `service term description`, composed of the following three sections:

- `Resources Provider`: this section describes the available infrastructure resource providers (id, name, zone, and maximum number of allowed instances reservations, if applicable) and the available appliances (i.e., VMs) offered by each provider (type of appliance, HW/SW features and description);
- `Capabilities`: this section describes the security

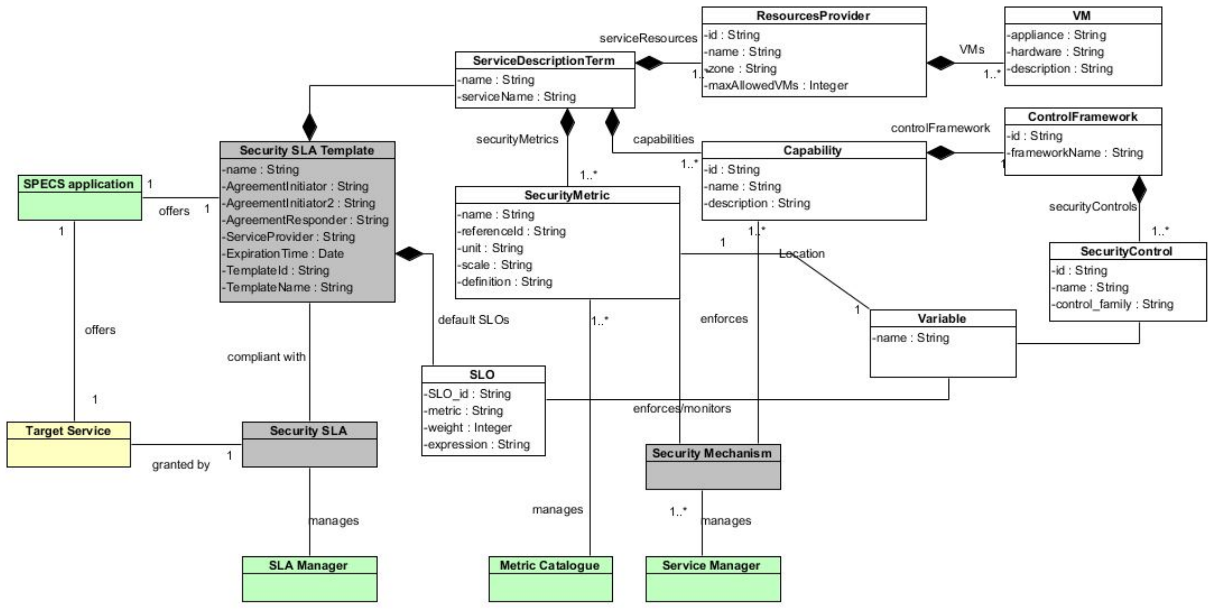

Fig. 3. The SPECS Security SLA model

capabilities[2] offered/required on top of the services covered by the agreement. Each capability is defined as a set of security controls belonging to a Security Control Framework, such as NIST's Control Framework [8] or Cloud Security Alliance's Cloud Control Matrix [9];

- `Security Metrics`: this section includes the specification of the security metrics referenced in the `service properties` section and used to define Security Service Level Objectives (SLOs)[3] in the `guarantee terms` section. A metric specification includes all information needed to identify it and to correctly process the SLOs in which it is involved, such as the metric name, its definition, its unit and scale of measurement, and the expression used to compute its value.

The association among security metrics and security capabilities (more specifically, security controls) is made through the `Variable` fields, which are used to define measurable and exposed properties associated with a service. Finally, `guarantee terms` include the conditions that must be verified to fulfill the agreement. We adopted the `CustomServiceLevel` item of the WSAG specification to define our custom Security SLOs, identified by an SLO id, a reference to the metric involved in the SLO, and the related expression, along with a weight assigned by the service customer and representing the related level of importance.

In Figure 3, the SPECS Platform components are colored in green: the Security SLA life cycle is managed through the `SLA Manager`, and a SPECS application can use the SLA Manager APIs to store/retrieve/update Security SLAs.

The enforcement of security capabilities and the monitoring of related security metrics (as specified in the SLOs) is performed by software tools called `Security Mechanisms`: they are selected, deployed and configured during the *Implementation* phase. The `Service Manager` maintains all the information associated with available security mechanisms that are needed to automate their deployment and execution together with the target services (i.e., mechanism *metadata*). Finally, the `Security Metric Catalogue` maintains, in machine-readable format, the information used for the definition of the security metrics. In our model, metrics are defined according to the NIST RATAX framework [10].

IV. Anatomy of a SPECS Application

As introduced in the previous section, a SPECS application is in charge of delivering cloud services according to SLA life cycle. SPECS applications could be developed in two different ways, namely *(a)* by simply reusing the public interfaces offered by the SPECS Platform's main modules (SLA Platform, Negotiation, Enforcement and Monitoring), or *(b)* by exploiting the tools offered by the SPECS framework to build a custom SPECS application. This second option is more interesting, and will be presented in the following.

The SPECS framework provides a *default SPECS application* in the form of servlets for Apache Tomcat, which offers to customers services that are orchestrated according to the SLA phases. The default application is easily customizable, in that it is independent of the actual services offered (both target services and security services), which can be specified inside a WS-Agreement-compliant template. In practice, the default SPECS application can be used as a basis for straightforward development of any application, since the only effort required

[2]Security capabilities are defined by NIST as combinations of mutually-reinforcing security controls (i.e., safeguards and countermeasures) implemented by technical means (i.e., functionalities in hardware, software, and firmware), physical means (i.e., physical devices and protective measures), and procedural means (i.e., procedures performed by individuals) [8].

[3]Security SLOs are conditions involving suitable security metrics, which allow to measure the level of security being delivered with a service.

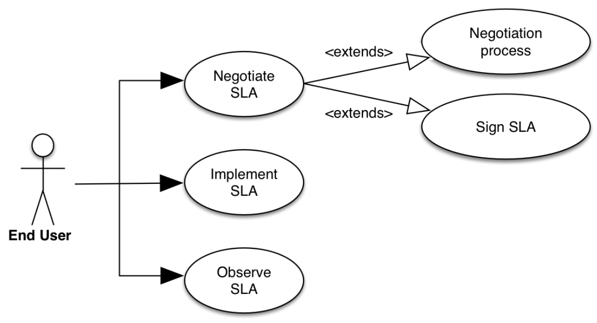

Fig. 4. The SPECS application use cases

from the developer is the provision of suitable metadata, without any code development. This process is described in the next section.

The implementation of the default SPECS application automates the invocation of the SPECS Core services, by offering to customers a dedicated page for each of the use cases illustrated in Figure 4:

- the **Negotiate SLA** use case activates a Negotiation wizard that builds up the Security SLA by identifying the security capabilities, the security metrics and the SLOs the End-user is interested in. When the process ends, the End-user can sign the SLA, completing the process.
- the **Implement SLA** use case starts up the SLA Implementation process: the cloud service is delivered, along with related security mechanisms, according to the previously signed SLA.
- the **Observe SLA** use case refers to the implemented SLA, and enables an End-user to monitor the agreed SLA and to verify that it is respected.

The *Negotiation process* is based on the adoption of an SLA template compliant with the format discussed in Section III, which the default SPECS application retrieves from the Negotiation module and uses to automate the full negotiation process, up to the definition of a complete SLA. It should be noted that the Negotiation module also offers the features needed to guarantee that the SLOs offered in the SLA can be actually implemented and granted (SLA validation). Once the End-user signs the SLA, this is stored into the SLA Platform, which will maintain its status. The default SPECS application implements the interface between the End-user EU and the SPECS platform. It performs the actual negotiation, in a way similar to a wizard on the top of a WS-Agreement template. Currently the Negotiation focuses only on the SPECS-supported Security SLOs. However, it is possible to extend it to other non-functional SLOs.

The *Implementation process* accepts as input a Security SLA Offer, filled according to the template, and automatically identifies the security mechanisms needed to grant the security capabilities and the SLOs reported in the SLA. When the End-user asks to implement a signed SLA, the SPECS application automates the execution of the offered service, by acquiring all the required cloud resources from an external CSP chosen according to the SLA, and by deploying on top of them all the security mechanisms able to implement the security capabilities and to enforce the SLOs agreed in the SLA. Also in this case, the process is completely automatic. The SPECS application has just to invoke the implementation REST API offered by the Enforcement module. Our default SPECS application already offers all the required features: both the cloud service offered to customers and the security mechanisms adopted to grant security capabilities rely on *Chef* deployment solution[4]. Chef automates the process of building, deploying, and managing software over an ICT infrastructure. It includes three main components: *Chef Server*, *Chef Client* and *Chef Workstation*. The Chef server stores software in form of *cookbooks* organized in *recipes*. The Chef client is installed on cloud resources (virtual machine, container, or networking device). SPECS periodically polls the Chef server for the latest policy and state of the software infrastructure. If something on the node is out of date, the client brings it up to date. The last component, the *Chef workstation*, is simply the machine that the system administrator uses to control the Chef Server. Our SLA Implementation module acts as a Chef Workstation, by automating the configuration process of the Chef server (usually made by human operators) according to the Security SLAs.

In order to enable the above-described automation process, each mechanism is enriched with a metadata file, which reports in detail all the information that SLA implementation will use to automatically configure the mechanism itself. Moreover, each mechanism declares the security capabilities that can be granted over the target service and the metrics that can be enforced/monitored. The description of the security mechanism metadata is out of the scope of this paper.

Finally, the SPECS default application enables End-users to continuously monitor the agreed SLAs (*Observe SLA*). To this aim, it offers through a servlet an *Observe* web page that summarizes the acquired resources and their status, reporting the agreed metrics and their current values.

V. DEVELOPMENT PROCESS

As outlined in the previous section, the default SPECS application automates the most part of the cloud service delivery following the SLA life cycle. However, such automation must be supported by the correct development and deployment of the mechanisms able to grant the features agreed upon and contained in the SLA.

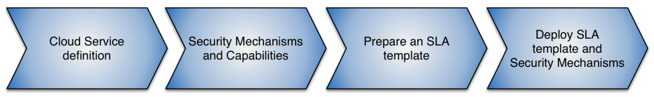
Cloud Service definition
Security Mechanisms and Capabilities
Prepare an SLA template
Deploy SLA template and Security Mechanisms

Fig. 5. SPECS application development process

Figure 5 summarizes the four main steps for the development of a new SPECS application:

- **Cloud Service Definition**: during this phase, the developer builds the cloud services to be delivered

[4]https://www.chef.io/chef/

through the SPECS application (e.g., web servers, storage services) and prepares the related cookbooks.

- **Security Mechanisms Definition**: during this phase, the developer prepares (or selects among those already available) the security mechanisms to offer on top of the cloud services defined at previous step. Each security mechanism comes with its cookbook, declaring the granted security capabilities (and related security controls) and the enforced/monitored security metrics, and with its metadata, which include all related recipes (i.e., the software components implementing the mechanism) and deployment constraints (e.g., incompatibility or dependency of software components implementing the mechanism).
  Security mechanisms can be specific for the service to be offered or generic ones. In this case, they are simply applied to the cloud resource involved (as an example, vulnerability assessment for the virtual machines acquired).
- **Preparation and deployment of the Security SLA template**: once the security mechanisms needed for the target cloud service have been defined, the developer has to compile a WS-Agreement-compliant template, which summarizes the possible features that can be offered to End-users, and has to make it available in the Negotiation module.
- **Deployment of Security Mechanisms**: the last development step is the deployment of the security mechanisms to make them available to the SPECS application. All the cookbooks must be registered with the Chef server, in order to enable the enforcement module to implement the SLA. The metadata of mechanisms must be registered in the SLA Platform in order to enable the SPECS application to retrieve the information and to implement the SLA.

It is worth noting that the SPECS application development mainly focuses on the development of ad-hoc Chef cookbooks for the security mechanisms to be offered. When cookbooks are already available (there are many archives of already-developed cookbooks), the only additional work consists in the preparation of the metadata and SLA templates used to automate the SLA implementation.

VI. The Web Container Example

In order to illustrate better the process of developing a SPECS application, in this section we focus on a simple example related to the acquisition by a web developer (the End-user, EU) of a web container enriched with a set of security features. It is reasonable to suppose that the EU is not a security expert, in that he is aware of the technologies that may be involved (SSL, authentication and authorization protocols, etc.), but has no cognition of the best practices and of how to protect his application from malicious attacks. For this reason, the acquisition of VMs hosting the web container and the enforcement of security features are accomplished through SPECS.

It should be noted that, at the state of the art, even if the web developer acquires VMs from a public CSP, he is the only person in charge of setting up any security configuration. Existing appliances offer predefined services (for example, a pre-configured web server), but checking and comparing the security features offered by different CSPs is not an easy task. The web developer has to (i) manually find the security features provided by each CSP, (ii) evaluate and compare existing offers, (iii) apply a suitable configuration, if not natively supported, and (iv) implement a monitoring solution to verify at runtime the respect of the security features.

The SPECS ecosystem provides a turnkey solution to the above issues, as it (i) offers a single interface to choose among multiple offerings on multiple providers, (ii) enables the web developer to specify explicitly the needed security capabilities on the target web container, (iii) automatically configures the VMs in order to enforce the security controls requested, (iv) offers a set of security metrics to monitor the respect of the security features requested, (v) enables continuous monitoring of the security metrics negotiated, and (vi) can automatically remediate to (some of the) alerts and violations that may occur to the SLA associated to the web container.

Below we will present the development of the Web container as a SPECS application, following the steps dealt with in the previous section.

Cloud Service Definition: The main goal of our case study SPECS application is to deliver web servers, pre-configured according to security best practices. To this aim, we developed a mechanism named *WebContainerPool* devoted to automatically deploying and configuring a pool of web servers over a set of virtual machines acquired from an external CSP (Amazon in our implementation). The developed mechanism not only provides the web server cloud service, but it also offers some reliability features, measured in terms of the two metrics (i) `LevelofRedundancy` and (ii) `LevelofDiversity`. The former ensures resiliency to failures through replication of the web container instances (used transparently by the End-user), while the latter ensures resiliency to vulnerability-based attacks by employing different software and/or hardware instances of the same web container.

In practice, the *WebContainerPool* mechanism has been developed as a security mechanism, and includes a set of pre-configured web servers (at the state of the art, Apache and Ngnix) and a load balancer (based on HAProxy). Web servers are synchronized through Memchached, so that the accesses to the web application are synchronized. The Chef cookbook associated to the *WebContainerPool* can be used also independently of the SPECS framework. It is worth pointing out that, if the aim is to apply the same process to a different cloud service (e.g., a Secure CMS), it is first necessary to develop a Cloud service cookbook dedicated to offer the CMS, and later on to select the security mechanisms that can be offered for it, possibly developing custom ones.

Security Mechanisms Definition: The proposed service (web container), as outlined above, relies on (a pool of) virtual machines, hosting synchronized web servers. The service offers some integrated security features (redundancy and diversity), but a lot of additional security capabilities can be provided. In SPECS three main security mechanisms are already available:

- **TLS**: it is a preconfigured TLS server, configured according to security best practices.
- **SVA (Software Vulnerability Assessment)**: it regularly performs vulnerability assessment over the virtual machines, through software version checking and penetration tests.
- **DoSprotection**: it consists in a solution for denial of service attacks detection and mitigation based on the OSSEC tool.

The main role of the developer is to select the mechanisms to be provided together with the web server mechanism from the catalogue of available security mechanisms (mantained by the SPECS Service Manager). If the developer is interested in offering additional security mechanisms, enforcing security metrics and capabilities not yet supported in SPECS, he has to develop a new cookbook, and enrich it with the SPECS metadata. This activity is out of the scope of this paper.

Preparation and deployment of the Security SLA template: This is the main developer task, as it summarizes all the possible offers to the End-user. Once the template is available, the SPECS application execution is fully automated. WS-Agreement templates are written according to the SLA model proposed in Section III, following the WS-Agreement schema and the SPECS security extensions. The XML schema corresponding to our Security SLA model is available at http://www.specs-project.eu/schema.

Deployment of Security Mechanisms: To complete the deployment of the SPECS application, the security mechanisms have to be deployed. This entails:

- All cookbooks of the chosen security mechanisms are added to the Chef repository associated to SPECS implementation component.
- All cookbook metadata are made available on the SLA Platform, which offers a simple REST API to upload such data and check them.
- The application template is uploaded to the Negotiation module (by using the dedicated REST API).

The above example is available online as demonstrator application at http://apps.specs-project.eu/specs-app-webcontainer-demo_CCM/. All the mentioned cookbooks can be found in the SPECS bitbucket repository (https://bitbucket.org/specs-team/).

VII. Related Work

It is widely recognized that the lack of security perceived by customers and business owners prevents the large-scale adoption of clouds solutions. As a matter of fact, due to the delegation of the management of all resources to the cloud, customers lose control over the status of their data and applications, making it impossible to be in charge of the security level of their applications. Currently there is a lot of ongoing work of the security research community, aiming to let cloud users trust that their applications are securely executed, suitably protecting their data. This has led to the multiplication of *ad hoc* security solutions, not portable or even not useful in different contexts [11].

Only recently it has been recognized the need to tackle security problems from the start, i.e., at application design time, as the addition of security features to an existing application is complicated and partly ineffective. Mohammadi *et al.* are working on the development of applications that can provide *trustworthiness* (the assurance that the system will perform as expected [12]) by design [13], [14]. The idea is to design the software in a way so that there will be mechanisms to ensure, evaluate and monitor trustworthiness, relying on reusable development process building blocks, consisting of method descriptions (guidelines, patterns and check-lists) ensuring that the right mechanisms are put in place to ensure trustworthiness.

The solution discussed in this paper instead relies on the use of SLAs. As the guarantees on the provisioning of a service is customarily managed through *Service level Agreements* between service providers and customers, an almost obvious solution is to use SLAs also for security, specifying suitable *Service Level Objectives* (SLOs) dedicated to security. Standards for the definition of the security terms in an SLA are still lacking, but there is currently a lot of ongoing work by dedicated standard groups (as the SLA C-SIG from the European Commission, the CSCC SLA group [15]) and research projects (see CUMULUS [16], A4Cloud [17], and SPECS [2]).

As mentioned before, the FP7-ICT programme project SPECS addresses cloud security through SLAs. Its objective is to improve the state-of-the-art in cloud computing security by creating, promoting and exploiting a user-centric framework and a platform dedicated to offer Security-as-a-Service using a SLA-based approach, in particular with respect to negotiation, continuous monitoring and enforcement [2], [3].

WS-Agreement [4] is the only standard supporting a formal representation of SLAs and a protocol that aims at their automation. The main limit of such solution is that it was devised in a grid-oriented technological context, and that it is not completely fit in other contexts, such as clouds.

The majority of the cloud-oriented FP7 projects (Contrail[5], mOSAIC[6], Optimis[7], PaaSage[8]) are inclined to adopt WS-Agreement representations, suitably adapted to the cloud context. In one case [18] the implementation approach followed is similar to the one adopted by SPECS [6], in that a REST API is developed to support the WS-Agreement protocol. However, to the best of our knowledge, none of the main commercial IaaS providers (Amazon, Rackspace, GoGRID, ...) currently offers negotiable SLAs. A survey of the SLAs offered by commercial cloud providers can be found in [19].

Not so much work has been done in the area of configuring security requirements specified through SLA documents. Karjoth et al. [20] introduce the concept of Service-Oriented Assurance (SOAS). 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. [21] present a WS-Agreement approach for

[5]http://www.contrail-project.eu
[6]http://www.mosaic-cloud.eu
[7]http://www.optimis-project.eu
[8]http://www.paasage.eu

a fine-grained security configuration mechanism to allow an optimization of application performance based on specific security requirements. Brandic et al. [22] present advanced QoS methods for meta-negotiations and SLA-mappings in Grid workflows.

VIII. Conclusions and Future Work

In this paper, we have introduced the SPECS framework as a solution for developing secure cloud applications covered by a Security SLA. The main benefit of using the SPECS approach is the opportunity to offer security assurance to cloud customers, by managing the life cycle of agreed security parameters contained in Security SLAs. We have described the Core services and APIs offered by the framework and a Security SLA model, an extension to the WS-Agreement standard, compliant with current standards and guidelines provided by the NIST and by CSA. The development process has been presented in detail to show how a security-enhanced application can be straightforwardly developed making use of the default SPECS application provided, and a case study has been discussed to illustrate the functionalities and potentialities of the framework.

Future versions of the default SPECS application will integrate the security evaluation techniques that we are developing in SPECS, in order to rank the Security SLA according to customer requirements. Additional functionalities (e.g., end-to-end encryption, vulnerability assessment, etc.) will be provided in the future to enrich the set of APIs and the set of available security mechanisms and monitorable metrics.

The SPECS applications may be deployed and offered by Cloud Service Providers, in order to define and agree on SLAs with their customers, but even by third-party providers that can act as brokers of services to enrich security capabilities of larger providers. This last delivery model may open new business opportunities, especially in those contexts (i.e., public sectors) where security represents the key factor to decide to cloudify a service.

References

[1] M. Dekker and G. Hogben, "Survey and analysis of security parameters in cloud SLAs across the european public sector," 2011.

[2] M. Rak, N. Suri, J. Luna, D. Petcu, V. Casola, and U. Villano, "Security as a service using an SLA-based approach via SPECS," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2, 2013, pp. 1–6.

[3] V. Casola, A. De Benedictis, M. Rak, and U. Villano, "Preliminary design of a platform-as-a-service to provide security in cloud," in *CLOSER 2014 - Proc. of the 4th Int. Conf. on Cloud Computing and Services Science, Barcelona, Spain, April 3-5, 2014.*, 2014, pp. 752–757.

[4] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (WS-Agreement)," in *Global Grid Forum.* The Global Grid Forum (GGF), 2004.

[5] International Organization for Standardization, "ISO/IEC NP 19086-1. Information Technology–Cloud Computing – Service Level Agreement (SLA) Framework and Technology – Part 1: Overview and Concepts," 2014.

[6] A. De Benedictis, M. Rak, M. Turtur, and U. Villano, "REST-based SLA Management for Cloud Applications," in *Proc. 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2015)*, 2015, pp. 93–98.

[7] V. Casola, A. De Benedictis, and M. Rak, "User-centric security monitoring in the cloud: a SLA-based approach," in *To Appear in Proc of The 1st International Workshop on Security Testing And Monitoring*, 2015.

[8] NIST, "NIST Special Publication 800-53 Revision 4: Security and Privacy Controls for Federal Information Systems and Organizations," 2013.

[9] Cloud Security Alliance, "Cloud Control Matrix v3.0," https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3/.

[10] NIST, "NIST Special Publication 500-307 Draft: Cloud Computing Service Metrics Description," 2015.

[11] C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu, "From security to assurance in the cloud: a survey," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 2:1–2:50, Jul. 2015.

[12] A. Aviienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[13] F. Di Cerbo, P. Bisson, A. Hartman, S. Keller, P. Meland, M. Moffie, N. Mohammadi, S. Paulus, and S. Short, "Towards trustworthiness assurance in the cloud," in *Cyber Security and Privacy*, ser. Communications in Computer and Information Science, M. Felici, Ed. Springer Berlin Heidelberg, 2013, vol. 182, pp. 3–15.

[14] N. Mohammadi, T. Bandyszak, S. Paulus, P. Meland, T. Weyer, and K. Pohl, "Extending software development methodologies to support trustworthiness-by-design," vol. 1367, 2015, pp. 213–220.

[15] CSCC, "The CSCC practical guide to cloud service level agreements," Tech. Rep.

[16] A. Pannetrat, G. Hogben, S. Katopodis, G. Spanoudakis, and C. Cazorla, "D2.1: Security-aware SLA specification language and cloud security dependency model. technical report, certification infrastructure for multi-layer cloud services (cumulus)." 2013.

[17] S. Pearson, "Toward accountability in the cloud," *Internet Computing, IEEE*, vol. 15, no. 4, pp. 64–69, July 2011.

[18] R. Kübert, G. Katsaros, and T. Wang, "A RESTful implementation of the WS-Agreement specification," in *Proceedings of the Second International Workshop on RESTful Design*, ser. WS-REST '11. New York, NY, USA: ACM, 2011, pp. 67–72.

[19] L. Wu and R. Buyya, *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions.* IGI Global, USA, 2011, ch. Service Level Agreement (SLA) in Utility Computing Systems.

[20] G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner, "Service-oriented assurance, comprehensive security by explicit assurances," in *Quality of Protection*, ser. Advances in Information Security, D. Gollmann, F. Massacci, and A. Yautsiukhin, Eds., vol. 23. Springer US, 2006, pp. 13–24. [Online]. Available: $http://dx.doi.org/10.1007/978-0-387-36584-8_2$

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

[22] I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya, "Advanced QoS methods for Grid workflows based on meta-negotiations and SLA-mappings," *2008 Third Workshop on Workflows in Support of LargeScale Science*, 2008.