On Security SLA-Based Monitoring as a Service

Dana Petcu[1,2(✉)], Silviu Panica[1,2], Bogdan Irimie[1,2], and Georgiana Macariu[1]

[1] Institute e-Austria Timişoara, Timişoara, Romania
petcu@info.uvt.ro
[2] West University of Timişoara, Timişoara, Romania

Abstract. Client-driven monitoring of security service level agreements is not available nowadays in the market of Cloud services. Supposing that security obligations associated with a service will be available soon in the service level agreements, we designed such a monitoring service that can be deployed on Cloud provider premises or as external service. It is a stand-alone component of a larger system that allows the negotiation of service level agreements and their enforcement. The concepts, design and architecture of the proof-of-concept service are presented in this paper.

Keywords: Cloud · Security · SLA · Monitoring

1 Introduction

Resources deployed in a cloud environment grow day by day as companies and governments move from on premises model to the Cloud. Moving to the Cloud implies less responsibilities for the client as the Cloud provider manages different aspects of the infrastructure [1]. However, monitoring responsibilities should not be delegated completely to the Cloud provider, and monitoring from the user perspective should be implemented. The monitoring tools should allow clients to check that the quality of services they agreed with the Cloud provider is maintained. They can provide insights on the security of the system through the specification by the client of security parameters of interest. Such parameters can be the status of the ports or the services running on a specific host.

Client-driven security monitoring in Cloud environments is laging behind other client-driven operational monitoring, like performance monitoring. This fact is sustained by the absence of security obligations associated with a service in current service level agreements (SLAs), hindering the Cloud providers capacity to offer trustworthy services [2]. Currently, the only service aspect included in SLAs is service availability [3]. However, cloud services provider contracts are expected to provide soon detailed and substantial security SLAs [4,5].

In this context, we are interested to provide an open-source SLA-based Cloud security monitoring system that can act as Monitoring-as-a-service. It is deployed together with a customer application on a Cloud provider premises providing infrastructure-as-a-services (IaaSs), resides on the Cloud provider resources, or is offered on third party premises. The role of such an SLA-based Cloud security

B. Mandler et al. (Eds.) IoT 360° 2015, Part I, LNICST 169, pp. 326–336, 2016.
DOI: 10.1007/978-3-319-47063-4_34

Table 1. Challenges, barriers, models, metrics

Category	Short description
Challenges	Mapping between low-level metrics and application-based SLA parameters
	Ability to monitor SLA parameters to multiple Cloud layers (IaaS, PaaS)
	Uncertainty of Cloud environments in event observation (rate of probes)
	Cloud agnosticism (tightly coupling of monitoring tools to the services)
	Big data security is time consuming, instant reaction difficult to achieve
Barriers	Security SLAs are not in place
	Privacy laws that are restricting instant monitoring by Cloud providers
	Security metrics are still vagues
	Virtualization makes monitoring harder
	The ability to monitor services is considered a security risk
Models	Multi-layer model (facility, network, hardware, OS, middleware, appl, user)
	Cloud security control domains (application, interfaces, identity, access etc.)
Metrics	Cloud service measure &metric (scenarios, measure, metric, measurement)
	Service measurement index (category Security and Privacy)
	Security indicators (e.g. rate of compliance with a catalogue of criteria)
	Cloud security properties (core elements: identifier, definition, attributes)
	Security parameters for monitoring (incident, data, change, log, isolation, etc.)

monitoring service in a larger framework, named SPECS, was exposed in [6]: a security SLA (Sec-SLA) that is negotiated with a IaaS provider will be monitored for compliance and alerts will be generated in case of security changes or in case of Sec-SLA violations (leading to its enforcement).

We identified recently in [7] the challenges, barriers, models and metrics for building a SLA-based Cloud security monitoring system. Table 1 summarizes our conclusions. Other opinions related to the challenges of security monitoring in Cloud environment are presented in [8].

In this paper we present the core and support services of the proposed service. The next section is referring to related work and our motivation. The third section is dedicated to core services, while the fourth to support services. The last section is dedicated to conclusions.

2 Related Work

Analyzing the reports about the academic prototypes or commercial services that are available for SLA monitoring or security monitoring in Clouds, we identified in [9] the fact that there is no report until date of an Sec-SLA based Cloud monitoring service. We mention here only few of the monitoring tools that were identified. There are several open-source SLA-oriented Cloud monitoring tools like CloudCompas (available on github.com), or Everest/SLA@SOI (on sourceforge.net), as well as open-source Cloud security monitoring tools like Snorby (on github.com). Commercial Cloud monitoring tools are many, e.g. SLA-oriented ones from nimsoft.com or site24x7.com, or security-oriented ones from ciphercloud.com, cloudflare.com, cloudpassage.com, splunk.com or threatstack.com. Often Cloud monitoring tools are relying upon the services of open-source general monitoring tools, like collectl, Ganglia, Nagios (all three on sourceforge) or MonALISA (monalisa.caltech.edu). A comprehensive study of the Cloud monitoring systems is available in [10].

Security parameters for monitoring systems were classified in [11,12], along with methods and techniques for measuring parameters in practice. Thresholds were established also to indicate when to trigger an event. However, security indicators (observable characteristic that correlates with a desired security property) were not provided. To overcome this problem, a step forward was made in [13] by providing an attribute-based security property vocabulary (security properties in abstract terms and as a properties with a set of defined attributes).

Cloud security monitoring is currently done on-premises, on the monitored infrastructure, or via a SaaS. In the case of monitoring on-premises, a security tool is able to make use of specific APIs as well as to collect logs from Cloud services. In the second case, of monitored IaaS, a security tool is loaded directly into an IaaS (no high bandwidth requirement, possible some high storage costs). In the third case, monitoring data is obtained from the Cloud service (if available), and hand it to a managed security service provider. We are interested to offer a deployable service that supports client-driven monitoring and can be mapped to all three cases.

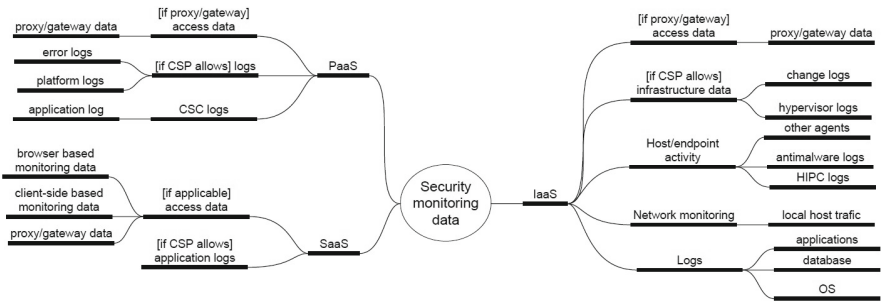

Fig. 1. Security monitoring data

We proposed recently in [14] a taxonomy for the SLA-based monitoring of cloud security. We reproduce here in Fig. 1 the class related to security monitoring data. Note that monitoring specific utilities for collecting information about security are referring to software vulnerabilities or bugs (OS/middleware layer), IDS or firewalls (network), authentication systems or surveillance (facility), workload, voltage or temperature, memory or CPU (hardware).

3 SPECS Monitoring

Monitoring is highly important for SPECS framework. The features offered by its platform(-as-a-service) rely on the information processed by the monitoring module (whether we are talking about to overall platform functional process or about the end-users SLAs that need to be fulfilled).

Monitoring as a Module in the SPECS Framework. The SPECS monitoring module addresses the problem of using monitoring for the fulfillment of security-related user requirements. It enables the users to continuously keep an eye on their applications with respect to certain security properties that might be of interest to them. The module monitors the resources and services and notifies events considered of relevance (according the SLAs) to an enforcement module. The module integrates existing and custom monitoring tools/agents to gather information on SLO metrics, in order to help the enforcement module to detect possible alerts and violations.

Once signed a SLA enters the observed state in which dedicated monitoring agents keep collecting information with regards to the execution of the negotiated services, continuously checking in that way the fulfillment of the specified service level objectives (SLOs). The module focuses on SLO metrics, i.e. the measurable part of the SLAs. This assumption keeps the monitoring and enforcement (diagnosis) functionalities separated, and improve the scalability of the approach in presence of a large number of SLAs to follow. We assume the existence of a monitoring services repository that contains a static mapping between possible SLO metrics and available monitoring components able to gather data on those metrics. An enforcement planning component performs a lookup in such repository to retrieve the monitoring services to activate. Such services are then deployed, configured and activated by an enforcement implementation component. The configuration of the monitoring systems includes setting proper thresholds, intervals and values for the related SLO metrics according to what has been specified in the SLA. The activation of the monitoring components is carried out by a platform functionality (see next section).

Monitoring Workflow. The module deals with large amount of information that needs to be collected, filtered or routed to other components. The monitoring data volume depends on the number of metrics that need to be monitored together and the number of users the platform need to deal with.

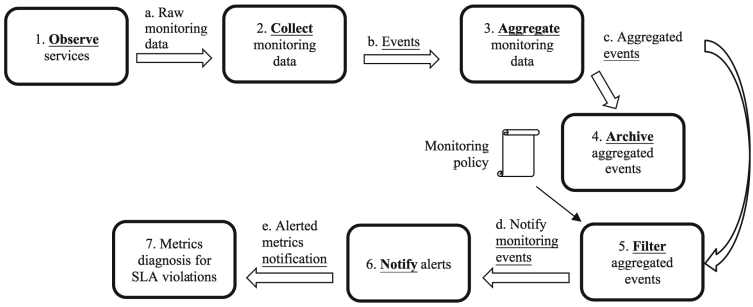

Fig. 2. Simplified monitoring workflow

The monitoring workflow include the following tasks (described in Fig. 2):

observe and collect: the targeted services are continuously monitored and specific data is collected directly from the services log files; this data is then sent to the core monitoring infrastructure from where it is routed to dedicated components that need to analyze the data;

aggregate: the monitoring information is used to compute some statistic data regarding different targeted service behaviour; this aggregation is made based on predefined or dynamically defined aggregation rules described in monitoring policies that are continuously updated during the platform runtime;

filter: the monitoring data received from the targeted services is filtered and routed to specialized services that needs the data to analyze;

archive: all the monitoring data is archived for later use, for example in case of historical statistical computation of some defined metrics;

notify: send out external notifications in case of broken filtering rules set for the monitored metrics.

The monitoring data is split into two types: the *raw monitoring data*, collected by the monitoring adapters from the targeted services and *monitoring events*, the data that is send through the monitoring infrastructure. The raw monitoring data collected by the monitoring adapters is mapped into a standard message called the event format. The event format uses a simple but general structure (Table 2) in order to allow any type of raw monitored data to be mapped. In this way the monitoring core services are independent from the platform and can be reused while not being tied up to a specific set of services that can be monitored.

Core Services. The components of the SPECS monitoring core services are:

Event Archiver: aims to retain all the monitoring data for a defined period of time for later data preprocessing;

Event Aggregator: is responsible with point-in-time observations, transformations of events by showing the global status of the monitored system;

Table 2. Monitoring event format

Field	Description
component	Identifier of the component instance generating the event (i.e., a VM)
object	Hierarchical string pointing event source that generated the event
labels	Hierarchical string that provides a way to give a context to the event
type	Hierarchical string representing the type of event
data	Concrete information specific for each type of event
timestamp	Time of the event, in seconds
token	(optional) Used by some monitoring component for a specific purpose

Event Hub: acts as a router between the monitoring adapters and the other monitoring components; it uses filters to route the monitoring data among the components;
Monitoring Adapter: collects the raw data from the targeted services and sends out the data to the monitoring core in form of monitoring events;
Monitoring Policy Filter: filters the aggregated events and searches for possible violation or alerts of the monitored metrics;
SLO Metrics Exporter: notifies the others platform components in case of violations or alerts set for the monitored metrics.

The Event Hub receives the events collected by the various Monitoring Adapters and routes them towards the Event Aggregator, Event Archiver and the Monitoring Policy Filter (Fig. 3). There are several important sub-components. A HTTP Mux implements an HTTP interface through which the Monitoring Adapters publish events to the Event Hub and clients like the MoniPoli Filter can receive desired events. A Router Multi-Decoder, which accepts as input events

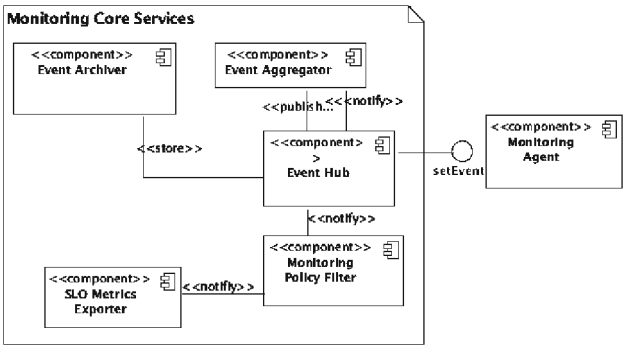

Fig. 3. SPECS monitoring core services architecture

and transforms them into a format which can be handled by the Mozilla Heka[1] stream processing and routing system, used internally by the Event Hub. A Router Multi-Encoder encodes the Hub's internal messages back as events, represented in the platform's internal format (Table 2). A Router Output, together with the HTTP Mux sub-component, streams events to all interested parties. An Archive Output call the Event Archiver in order to store routed events. A Heka Router forwards internal messages to any filter of the Event Hub and to the Router Output and Archive Output sub-components. A Router Input uses the Router Multi-Decoder for decoding received events and delivers the decoded internal messages to the Heka router. Sieve Filters group events based on information contained in them. For example, one may define a filter for grouping all events related to CloudWatch[2] and use this in order to stream all these events to the MoniPoli Filter.

The Event Aggregator consumes events and pushes out the data back into the Event Hub for further consumption. Based on various aggregation rules, it aggregates the events. To fulfill the usual statistical measurements (like min, max, average, standard deviation, etc.) a basic aggregator implementation is currently implemented. More sophisticated, implementations could be implemented to support more complex measurements like identifying trends through various methods like statistical or neural networks. At runtime there can be multiple instances of the same event aggregator implementation, with different aggregation rules, especially for scalability purposes.

The Monitoring Policy Filter has three main components. The Event Filter registers itself to the Event Hub to receive all the stream of events labeled with the security metric labels reported in the corresponding label attribute of the policy. The MoniPoli Rule Filter applies the rule at runtime and export them, through the MoniPoli Output interface. The MoniPoli Rule generator accepts as input a new SLAs and generates new rules, according to the algorithms proposed in the MoniPoli section, communicating them to the MoniPoli Rule Filter.

The Event Archiver retains the monitoring data and events for a defined period of time (when an SLA finished its execution or it is terminated then all the associated archived monitoring data and events are disposed). The communication interface is based on a REST API that supports PUT, GET and DELETE actions (store data and events; retrieves the data from the archiver database at query like event attribute and a time interval; erases the data from the database on query); operation handlers are the actual functions that perform a specific operation. A request pool handles multiple requests that need to be routed to specific internal handlers. A distributed object-store database is used for storing the monitoring data and events.

The SLOM Exported receives only the monitoring events that should be notified. MoniPoli Filter makes the selection of the events and forwards them to the SLOM Exporter. The Exporter generates an XML representation of the monitoring event, made in agreement with the SLA XML framework. It uses the

[1] https://github.com/mozilla-services/heka.
[2] https://aws.amazon.com/cloudwatch/.

SPECS's SLA Platform API in order to notify the event, in the right format to the enforcement module.

The Monitoring Adapter sends out the events to the Event Hub and receives data from Monitoring Agents. The format of this data depend on the type of agent, and the data is transformed by the Monitoring Adapter into the platform internal format to represent events that are finally sent to the Event Hub.

Monitoring Agents. Two agents were tested in SPECS context: OpenVAS, a vulnerability scanner and Nmap, a network security scanner. Three others are on the list for the next integration steps, to prove the concepts feasibility: OSSEC, a host intrusion detection and prevention system, Snort, a network intrusion detection system, and Monit, a general purpose monitoring tool[3]. Security metrics that can be monitored by Nmap, for example, are status of ports, service version, guess OS, time since last restart, ciphers used by TLS Ciphers used by servers that offer TLS.

We proposed in [16] a monitoring system (agent in SPECS context, despite the agent-less internal architecture) based on Nmap that allows to execute multiple Nmap instances at the same time to provide efficiency and fault tolerance. As number and type of resources that should be monitored can quickly become very large, a monitoring system should be scalable, provide small overhead and fault tolerance. The tests performed on Google Compute Engine premises showed the scalability of the proposed system. The event message mentioned in Table 2 includes in this case, as component, the UUID of the component that creates the message, as object, nmap, as labels, client id and job id, as type, metric, as data, Nmap results after they have been processed, and as timestamp, the time when the event message was created.

4 SPECS Enabling Platform

The Enabling Platform creates the execution environment for the SPECS' platform that hosts the SLA services. This component is a bootstrap service that transforms a standard resource (e.g. a Cloud virtual machine, a VM, with an Linux OS) into an execution environment ready to host other resources. The other resources have some special requirements in terms of local libraries or software packages and services or to remote resources to interact with. The Enabling Platform solves this issue and it is also able to acquire the compute resources, from various Cloud providers, where the SPECS Platform is hosted.

The Enabling Platform consists of several core components (Fig. 4):

– *resource allocator*, a standalone service able to acquire resources from the cloud providers based on a resource descriptor document where the end-user specifies the requirements and constraints for platform deployment;

[3] OpenVas: openvas.org; NMAP: nmap.org; OSSEC: ossec.net; Snort: snort.org; Monit: mmonit.com/monit/.

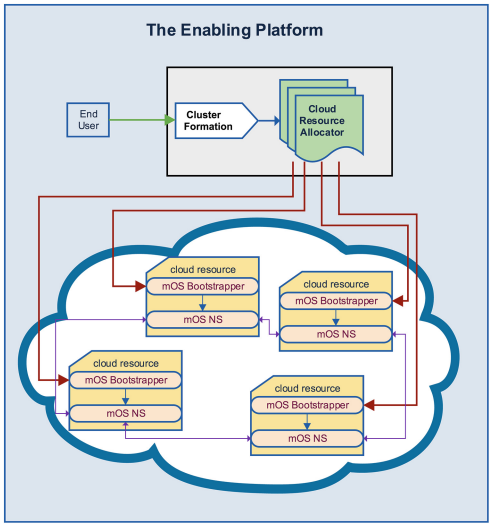

Fig. 4. Enabling platform architecture

- *cluster formation service*, an orchestrator service able to manage the resources scheduling based on the definition given by the end-user (resource descriptor);
- *mOS node bootstrapper*, a simple and autonomous resource clustering solution that transform a standard operating system into a specialized environment that is able to host the platform resources (mOS stands for multi-purpose OS); it uses Chef service-client architecture for package deployment;
- *mOS naming service*, a distributed resource management service able to offer resource management features like resource registration and retrieval in a heterogeneous distributed system.

We focus here on cluster formation service, mOS bootstrapper and naming service (NS), as having the particular ability (not yet encountered in literature) to transform a set of simple Cloud resources (acquired by the resource allocator) into an execution environment ready to host complex applications and services.

The cluster formation service takes a resource descriptor from the user or a third party service and tries to interpret it. The result is a bootstrap configuration that contains, among others, the number of resources that need to be acquired and for each resource a bootstrap plan that mOS node bootstrapper needs to apply on the acquired Cloud resources. The latest is a package that is deployed on the target VM, as a system service that will transform the VM, based on the bootstrap plan, into an execution environment that will host the target platform. It relies on Chef[4] technology for package deployment and on mOS NS for distributed resource identification. Chef server-client architecture is complex to configure in an unattended manner as it implies a lot of security

[4] www.chef.io.

and resource information to be exchanged between the clients and the server. Using the mOS NS service the clients can automatically find the Chef server contact information (like IP address, chef services ports and registration URLs used by the clients to get registered in the Chef server database). When the Chef cluster is setup the mOS node bootstrapper will tell to the Chef server to apply deployment plan on each client based on the initial implementation plan.

mOS naming service is a distributed resource management service that uses a distributed system as an engine based on Paxos algorithm described in [15]. The package is started on each targeted node and the distributed system will be automatically be created based on standard network discovery protocols (multicast or broadcast). When the nodes are synchronised they can start exchange and store information. Each time a Chef client will install a service a trigger is activated and mOS NS client will register the service together with some additional information (IP address, port number or other distinctive information) required for resource discovery. On the other hand when a node needs a specific resource to consume it will query the service to find out if such a service is already registered and what are the details of it. In this way the Chef server-client architecture can be deployed unattended and further the entire platform as well in the same manner. The naming service can be also used by any other resource that needs to store or retrieve other resource information because it offers a standard communication interface (REST).

5 Conclusions

The motivation, the concepts and implementation of a SLA-based Cloud security monitoring system were exposed in this paper. Special particularities are its modularity and ability to integrate external security scanners. The proposed software prototype is a proof of the fact that a interoperability layer can be used in Cloud security monitoring to interconnect various existing tools. The presentation focused on the component design and we neglected the description of the entire SPECS framework which can be found in previous reports.

Acknowledgments. This work is partially supported by the European Commission under grant agreement FP7-610795 (SPECS). We thank also Ciprian Crăciun for his consistent contributions to the design of the core services.

References

1. Spring, J.: Monitoring cloud computing by layer, Part 1. IEEE Secur. Priv. **9**(2), 66–68 (2011)
2. Bernsmed, K., Jaatun, M.G., Meland, P.H., Undheim, A.: Security SLAs for federated cloud services. In: 6th ARES, pp. 202–209 (2011)
3. Ouedraogo, M., Mignon, S., Cholez, H., Furnel, S., Dubois, E.: Security transparency: the next frontier for security research in the cloud. J. Cloud Comput. Adv. Syst. Appl. **4**(12), 1–14 (2015). doi:10.1186/s13677-015-0037-5

4. Wagner, R., Heiser, J., Perkins, E., Nicolett, M., Kavanagh, K.M., Chuvakin, A., Young, G.: Predicts 2013: cloud and services security. Technical report, Gartner ID:G00245775 (2012)
5. Casola, V., De Benedictis, A., Rak, M.: On the adoption of security SLAs in the cloud. In: Felici, M., Fernández-Gago, C. (eds.) A4Cloud 2014. LNCS, vol. 8937, pp. 45–62. Springer, Heidelberg (2015). doi:10.1007/978-3-319-17199-9_2
6. Rak, M., Luna, J., Petcu, D., Casola, V., Suri, N., Villano, U.: Security as a service using an SLA-based approach via SPECS. In: CloudCom 2013, pp. 1–6 (2013)
7. Petcu, D.: SLA-based cloud security monitoring: challenges, barriers, models and methods. In: Lopes, L., et al. (eds.) Euro-Par 2014, Part I. LNCS, vol. 8805, pp. 359–370. Springer, Heidelberg (2014). doi:10.1007/978-3-319-14325-5_31
8. Mazhar, A., Khan, S.U., Vasilakos, A.V.: Security in cloud computing: opportunities and challenges. Inf. Sci. **305**, 357–383 (2015)
9. Petcu, D., Craciun, C.: Towards a security SLA-based cloud monitoring service. In: 4th CLOSER, pp. 598–603 (2014)
10. Aceto, G., Botta, A., De Donato, W., Pescape, A.: Cloud monitoring: a survey. Comput. Netw. **57**(9), 2093–2115 (2013)
11. Hogben, G., Dekker, M.: Procure secure: a guide to monitoring of security service levels in cloud contracts. Technical report, ENISA (2012)
12. Rahulamathavan, Y., Pawar, P. S., Burnap, P., Rajarajan, M., Rana, O.F., Spanoudakis, G. Analysing security requirements in cloud-based service level agreements. In: 7th SIN, pp. 73–76 (2014)
13. Pannetrat, A., Hogben, G., Katopodis, S., Spanoudakis, G., Cazorla, C.S.: Security-aware SLA specification language and cloud security dependency model. Technical report, CUMULUS (2013)
14. Petcu, D.: A taxonomy for SLA-based monitoring of cloud security. In: 38th COMPSAC, pp. 640–641 (2014)
15. Lamport, L.: Paxos made simple, fast, and byzantine. In: OPODIS, pp. 7–9 (2002)
16. Irimie, B.C., Petcu, D.: Scalable and fault tolerant monitoring of security parameters in the cloud. In: 17th SYNASC (2015, in print)