C'MON: Monitoring the Compliance of Cloud Services to Contracted Properties

Soha Alboghdady, Stefan Winter, Ahmed Taha, Heng Zhang, Neeraj Suri
DEEDS Group, TU Darmstadt, Germany
{soha|sw|ataha|zhang|suri}@deeds.informatik.tu-darmstadt.de

ABSTRACT

The usage of computing resources "as a service" makes cloud computing an attractive solution for enterprises with fluctuating needs for information processing. As security aspects play an important role when cloud computing is applied for business-critical tasks, security service level agreements (secSLAs) have been proposed to specify the security properties of a provided cloud service.

While a number of approaches for service providers exist to assess the compliance of their services to the corresponding secSLAs, there is virtually no support for customers to detect if the services they use comply to the specified security levels. To close this gap, we propose C'MON, an approach to continuously monitor the compliance of cloud services to secSLAs. Our evaluation of C'MON shows its ability to identify violations of contracted security properties in an IaaS setting with very low performance overheads.

CCS CONCEPTS

•**Security and privacy →Distributed systems security;** •**Computer systems organization →Cloud computing;**

KEYWORDS

Cloud Computing, Security, Compliance, Monitoring

ACM Reference format:
Soha Alboghdady, Stefan Winter, Ahmed Taha, Heng Zhang, Neeraj Suri. 2017. C'MON: Monitoring the Compliance of Cloud Services.. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 6 pages.
DOI: 10.1145/3098954.3098967

1 INTRODUCTION

Cloud computing grants customers on-demand, elastic provisioning of computing resources, that are managed and maintained by Cloud Service Providers (CSPs). Despite the benefits provided by the cloud model, including the reduction of expenses and management overhead, the resultant loss of control over the security of the provisioned resources limits the adoption of the cloud model. The lack of security specifications and guarantees in most cloud service offerings leaves customers uncertain about whether or not the provided service satisfies their security requirements.

In order to address service security between the CSPs and customers, Security Service Level Agreements (secSLAs) are proposed as an extension to existing cloud SLAs that typically only cover performance properties of services [12]. In contrast to SLAs, SecSLAs are used to specify the security properties of the provided service. The specification of a security property defines the security mechanisms needed to achieve this property and also their implementations. Moreover, a secSLA includes Service Level Objectives (SLOs), which are target values for different service levels of the specified security properties. Consequently, secSLAs can serve two purposes. First, according to the security level determined by the specified values of the SLOs, customers can decide if the required security level for the provided service is satisfied. Second, binding the security properties in a contract between the CSP and the customer, obligates the CSP to provide the service with the contracted SLO values. However, to date there is no mechanism to enable customers to verify the CSPs' compliance to such contracts. Thus, the inability to validate if a contract is honoured by both parties renders the utility of such contracts questionable.

In this paper, we propose a framework for secSLA compliance validation which enables cloud customers to validate the compliance of the provided cloud service to the contracted security level in a secSLA. The approach is realized by first defining means to evaluate the SLOs associated with security properties defined in a secSLA. By using these means to monitor the values of the SLOs, the compliance of the service to secSLA is validated against the secSLA throughout the service life cycle and any detected violations are reported. The paper provides the following contributions:

(1) Definition of measurement procedures for SLOs proposed by different security controls frameworks and standards to be included in the secSLA. Additionally, measurements procedures are defined for SLOs that are measurable from the customer side.
(2) Design and implementation of a tool which monitors the values of the SLOs throughout the service life cycle, to ensure their compliance to the secSLA.
(3) Evaluation of the functionality and performance of the approach on two examples of Infrastructure-as-a-Service (IaaS) services using a OpenStack and Amazon Elastic Compute Cloud (Amazon EC2) instances.

The remainder of the paper is organized as follows. Section 2 outlines the state-of-the-art approaches proposed to validate the compliance of CSPs to SLAs. Section 3 introduces the basic concepts and the terminology used in the paper. Section 4 presents

Research supported in part by EC H2020 ESCUDO-CLOUD GA #644579, CIPSEC GA #700378 and BMBF TUD-CRISP

ARES '17, Reggio Calabria, Italy
© 2017 ACM. 978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3098967

the proposed cloud monitoring tool. Section 5 presents the experiments used to validate the proposed approach, and evaluates its performance. Finally, Section 6 concludes the paper.

2 RELATED WORK

Based on the service properties described in SLAs, multiple approaches have been proposed to validate the compliance of the service to the SLA, by verifying the enforcement of the contracted properties. Haq et al. [5] propose a framework to manage SLA validation by defining rules to map high-level SLOs to low-level metrics that the CSP can monitor throughout the service life cycle. Furthermore, Rak et al. [15] and Liu et al. [10] propose cloud application monitoring tools that detect SLA violations. Although these approaches provide effective techniques to detect SLA violations, they are only focused on monitoring *performance* properties rather than *security* properties.

Very few approaches exist for monitoring security properties of cloud services. Ullah and Ahmed [16] proposed a secSLA management solution that installs monitoring agents on the cloud service to measure service security properties. Ullah et al. [17] and Majumdar et al. [13] proposed tools to be used by the CSP to audit the compliance of their services to some security properties by depending on log analysis. Nevertheless, all of these approaches address only CSP side validation by proposing solutions managed/deployed by/at the CSP or require cooperation from the CSP. Hence, the customers are unable to verify the effectiveness of these approaches or ensure the validity of the reported results.

3 BACKGROUND & TERMINOLOGY

In this section we introduce the terminology and concepts used in the paper.

A secSLA is a contractual agreement between the CSP and the customer, which describes the security properties of the service to be delivered [12]. Security control frameworks are used to describe the security properties in the secSLA as a hierarchy of security controls refined into SLOs. SLOs are the targets for service levels that the CSP specifies in the secSLA and commits to achieve. An example of an SLO is the *mean time between incidents*, which refers to the average time between the discovery of two consecutive incidents [2]. The specified value for this SLO can be a lower bound which would imply that the reported average time between incidents should not fall below the defined lower bound at any time of the service life cycle. Measuring the value of an SLO requires defining a measurement mechanism. A measurement may be a formula, a process, a test or any schema usable to assign a value to the SLO.

Hence, the secSLA of a service enables the customers to understand which security properties are implemented in the service and obliges the CSP to deliver the properties with the contracted values of the SLOs. However, a secSLA by itself does not guarantee that the contracted security level is actually met. The CSP might fail to satisfy the contracted security level at any time during the service life cycle. The aim of the proposed approach is to provide customers with means to detect these violations of secSLAs. This is achieved by measuring the SLOs contained in the secSLA and ensuring the compliance of the measured values to the contracted ones.

4 C'MON MONITORING APPROACH

In the following we introduce C'MON, a cloud monitoring approach that enables customers to validate the compliance of a running cloud service to its secSLA. To enable customer-side SLO measurements, we performed an extensive literature survey to identify SLOs that are commonly contained in secSLAs and investigated each SLO to decide whether or not it can be measured by the customer. A total of 142 SLOs were examined for this work[1]. The SLOs considered are defined for IaaS services which also form the basis for ensuring the security of the other models (Software-as-a-Sevice or Platform-as-a-Service).

Based on the SLOs identification survey, we develop measurement mechanisms that allow the assessment of SLOs from the customer side, as detailed in Section 4.1. Using these mechanisms, a framework that enables the continual monitoring of SLOs is presented in Section 4.2.

4.1 SLO Measurement Mechanisms

Our targets are the SLOs whose measurement is achievable by the customers. Hence, in the following, we describe the measurement mechanisms corresponding to the requisite SLOs.

4.1.1 Percentage of Uptime. This SLO specifies a lower bound for the percentage of time, over a predefined measurement period, in which the service should be available. Based on the description of the SLO, the availability of a service is determined by the status of a request sent to the service at the time of measurement. If the CSP fails to deliver the service within a predefined time frame, the service is considered unavailable. The measurement period is divided into timeslots of fixed length *slotSize*. A slot is considered available if the percentage of failed requests within a timeslot is less than a defined percentage *timeslotFailThreshold*. Accordingly, the percentage of uptime is calculated as the percentage of time slots in which the service was considered available.

Measurement. To monitor the availability of the service throughout the full life cycle, requests are sent periodically to the service and the response of the CSP is verified. The status of the requests is used to calculate the percentage of uptime using Equation 1.

$$\frac{\sum Available\ slots}{\sum Slots} \tag{1}$$

4.1.2 Percentage of Timely Recoveries. This SLO specifies a lower bound for the percentage of unavailability events which last less than a predefined delay *maxTime* [2]. The SLO follows the same definition of the service availability, as the previous SLO, defined over time slots of size *slotSize* to determine unavailability events and their duration.

Measurement. The service requests are sent at a predefined frequency to detect service unavailability events throughout the full measurement period. The percentage of the unavailability events with duration less than *maxTime* is then calculated as follows.

[1]Multiple security SLOs from NIST and CIS were surveyed.

$$\frac{\sum Unavailability\ events(T < maxTime)}{\sum Unavailability\ events} \tag{2}$$

4.1.3 Percentage of Processed Requests. A lower bound for the percentage of successfully processed requests by the CSP over a given measurement period is specified by this SLO. A request is considered successful, if the service was delivered without an error and within a predefined time frame.

Measurement. To measure this SLO, service requests are generated at a predefined frequency and the percentage of successful requests is calculated over the measurement period.

$$\frac{\sum Successful\ requests}{\sum Requests} \tag{3}$$

4.1.4 Mean time between failure (MTBF). This SLO is used to specify a lower bound for the mean time between two consecutive failures of processing a request [2]. It is important to note that the failure here is defined as the failure of a single request.

Measurement. Test requests are continuously sent with a predefined frequency to the service. The time between every two consecutive failures is calculated. At the end of the measurement period, the mean time of all the reported times between consecutive failures is calculated using Equation 4.

$$\frac{\sum |Start\ of\ downtime - Start\ of\ next\ uptime|}{\sum Failed\ requests} \tag{4}$$

4.1.5 HSTS. This SLO refers to the usage HTTP Strict Transport Security (HSTS) [11].

Measurement. An HTTP host is declared as an HSTS host by issuing an HSTS policy, which is represented by and conveyed via the Strict-Transport-Security HTTP response header field [7]. To validate the continuous usage of HSTS during the service life cycle, HTTP GET requests are repeatedly sent to the service, checking the header field on every request.

4.1.6 Secure Cookies Forced. The secure cookies SLO [11] reports whether the service enforces the usage of secure cookies.

Measurement. By sending an HTTP GET request to the service and examining the Set-Cookie header in the responses, one can check if the secure attribute is set to true to validate the usage of secure cookies.

4.1.7 Forward Secrecy. The SLO reports whether the cloud service supports forward secrecy in its cryptographic channels.

Measurement. Connecting to a server which supports forward secrecy implies that the session key used during Secure Sockets Layer/Transport Layer Security (SSL/TLS) session establishment is independent from the server's private key [6]. Accordingly, achieving forward secrecy is reliant on the cipher suite chosen when initiating the SSL/TLS session since the cipher suite determines the key exchange algorithm used. The use of Diffie-Hellman key exchange supports this property. Hence, to check for the usage of forward secrecy, an SSL session is initiated with the server, including only the cipher suites which uses Diffie-Hellman key exchange in the client cipher suites. If the handshake was successful then the server supports the forward secrecy. Otherwise, if a handshake failure alert is raised and the session is terminated, then forward secrecy is not supported.

4.1.8 Client Certificates. This SLO refers to whether the service enables the use of client certificate in SSL/TLS-based authentication.

Measurement. An SSL session is initiated with the server to check whether the server uses the client certificate. To test this SLO, first a session is initiated without a client certificate. If a handshake failure alert is raised [3], then the server needs to verify the client certificate for authentication (i.e., a certificate is "required"). Else, if the handshake did not fail then the client certificate may be "optional" or "not required".

4.1.9 OCSP Stapling. Online Certificate Status Protocol (OCSP) stapling [4] reports if the use of OCSP for requesting the status of a digital certificate is enabled or not. A server that uses OCSP stapling sends the OCSP request on the behalf of the client staples the OCSP response to the handshake.

Measurement. To check whether OCSP is supported or not, an SSL session is initiated with the server while including the Certificate Status Request extension in the "client hello" SSL handshake message. If the Certificate Status response extension is attached with the SSL handshake, then the server supports OCSP stapling.

4.2 C'MON Monitoring Framework

C'MON is developed as a solution for the customers to monitor the compliance of the service to the secSLA throughout its life cycle. This is achieved by tracking the values of the SLOs using the measurements defined in the previous subsection. By comparing the measured values to the contracted ones, the framework enables the detection of violations to the secSLA. The architecture of the framework is shown in Figure 1. The components of the framework and their interactions are discussed in the following.

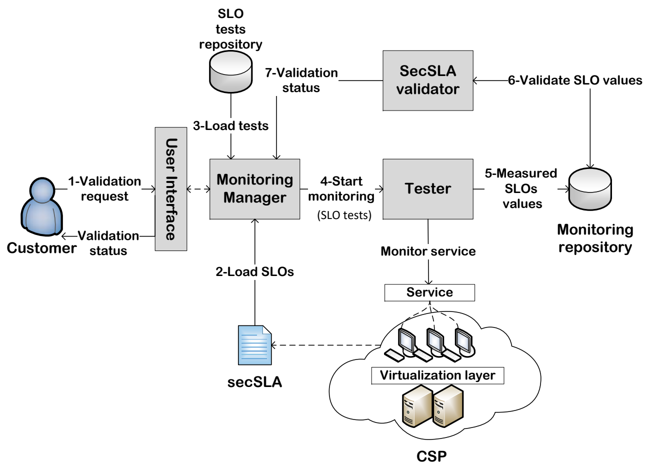


Figure 1: Monitoring Framework Architecture

Monitoring manager: The monitoring manager regulates the validation process. It takes the validation request from the customer as input, starts the monitoring and outputs the validation status.

SLOs tests repository: The measurements defined in Section 4.1 are formulated as tests to be executed during the monitoring to determine the real-time values of the SLOs. The tests are stored in this repository.

Tester: The tester performs the tests loaded from the repository to measure the values of the SLOs of the delivered service. The testing of the service is performed remotely using the provided information about the service with the validation request.

Monitoring repository: The monitoring repository maintains the monitoring results collected by the tester throughout the measurement period.

SecSLA validator: The secSLA validator is responsible for reporting the compliance status of the service. The validator compares the measured value of each SLO to its corresponding contracted value to detect violations. The service is said to be compliant to the secSLA if no violations are detected.

5 EVALUATION

A set of experiments were conducted to evaluate the approach. The first two experiments evaluated the functionality of the approach on two examples of IaaS services: OpenStack and Amazon EC2. The third experiment evaluated the overhead imposed by the proposed framework on both the monitoring server and the monitored cloud service.

5.1 Experiment 1: OpenStack Instance

In the first experiment, the effectiveness of C'MON for managing the compliance validation process has been evaluated by assessing C'MON's ability to measure the SLOs and detect violations of predefined values. The experiment was performed in a controlled environment using an IaaS service provided by a self-hosted OpenStack cloud platform. The platform was used to create a compute instance, which is used as a target for monitoring the SLOs *percentage of uptime, percentage of timely recoveries, meantime between failures* and *percentage of processed requests* by monitoring the availability of the instance using ping messages. In addition, OpenStack's dashboard was used as a web based interface to manage the instance. The SLOs *Secure cookie forced, HSTS, client certificate, forward secrecy* and *OCSP* are monitored for the dashboard to evaluate the session security properties of the management interface. C'MON was run on a server outside the cloud platform. The IP address of the instance and the host name of the dashboard were used to access the corresponding resources.

The experiment was performed by setting up a running instance with the management interface and enabling all the session security properties by editing the configurations of the dashboard. Violations of SLO values were deliberately caused at different times during the experiment to test C'MON's ability to detect them. Two types of violations were used: an instance availability outage, in which the instance was forced into an unavailable state, and a session security violation, in which support for the session security properties was detained (i.e., the usage of secure cookies is disabled after it was originally enabled). Over the entire one month experiment duration, C'MON accurately detected the artificially injected security property violations without any false negatives or false positives.

5.2 Experiment 2: Amazon EC2 Instances

In the second experiment, we evaluated our approach on a commercial cloud service setup using AWS EC2 instances. Three EC2 instances, each in a different geographical region, along with their web based management interfaces (Amazon management console) were monitored. The configurations for the instances are shown in Table 1. C'MON was deployed on a server outside the cloud network. The SLOs monitored for the instances and the management interfaces were the same as in the previous experiment.

Table 1: Instances Configurations

	US_EC2	FRA_EC2	TYK_EC2
Region	US East (N. Virginia)	EU (Frankfurt)	Asia Pacific (Tokyo)
Instance type	t2.nano		
Amazon M/C Image	Amazon Linux AMI 2016.09.0 (HVM) [1]		
Network	Default		
Availability zone	us-east-1d	eu-central-1b	ap-northeast-1a
Tenancy	Shared		

C'MON Configurations. With the aim to detect the shortest possible change and thereby achieve maximal coverage of changes in the monitored SLO values, a fine-grained monitoring was used. For availability, the shortest possible change was assumed to be the reboot time of an instance during which the instance is unavailable. The reboot time of an instance was experimentally assessed to be 5 seconds [2]. Hence, the monitoring frequency is set to one test every 5 seconds. For the session security properties of the management subsystem, a lower monitoring frequency was used, since changes in the configuration of the web server hosting the console are expected to occur less frequently. Moreover, automated access to the console at high rates might be considered malicious and thus the requests might get blocked by the CSP. Accordingly, the monitoring frequency for the consoles was configured as 6 times per hour.

We used the following values to calculate the SLO values in this experiment. The *slotSize* was set to 15 minutes with a *timeslotFailThreshold* equal to 10% [3]. The *maxTime* used to calculate the *percentage of timely recoveries* was one slot, i.e., 15 minutes. The experiment was also run for a month. Table 2 shows the measured values of the SLOs for all three instances. According to the results, the instance deployed in the US East (New Virginia) offered the least *percentage of uptime* with a percentage still greater than 99%. The lowest *percentage of processed requests* is reported for the instance located in Asia Pacific (Tokyo). The instance deployed in the EU (Frankfurt) had the highest *mean time to failure* with a value equal to 17 hours. All instances achieved 100% timely recoveries from outages for a *maxTime* of one slot duration, i.e., no outage lasted for more than 15 minutes for any instance. We analysed the collected data to investigate the frequencies and duration of the outages during the measurement period. The shortest observed outage is the failure of a single request, i.e., an outage duration of less than 10 seconds. The longest outage duration extracted from the results of all instances is 255 seconds experienced in the US_EC2 instance.

[2]5 seconds was assessed as a lower bound for rebooting a machine. This duration needs to be calibrated for each monitored system given its impact on both overhead and completeness.
[3]These values are arbitrarily chosen since they do not influence the actual values achieved by the service or the values measured by the framework.

The reported results for the management interfaces are almost identical for all three consoles. All session security SLOs were enabled and the client certificate was "Not required" for all consoles. The FRA_Console apparently did not support OCSP stapling. The assessment of these SLOs did not change over the measurement period. Therefore, no conclusions can be drawn from our experiment regarding the change frequency or duration for session security SLOs in real world settings.

Table 2: Amazon EC2 Instances Results

	US_EC2	FRA_EC2	TKY_EC2
% of Uptime	99.0915%	100%	99.9303%
% of Processed Requests	99.8088%	99.9930%	99.7571%
% of Timely Recoveries	100%	100%	100%
MTBF	36.41 min	1023.63 min	35.91 min

5.3 Experiment 3: Monitoring overhead

In the third experiment, the performance of the proposed approach was evaluated with the goal to estimate the communication and computational overhead C'mon imposes. This experiment was conducted on the same setup of the first experiment, as it provides higher controllability of the factors that potentially influence our performance measurements. The imposed overhead was evaluated for the monitoring server, the OpenStack instance, and the server hosting the dashboard. To estimate the highest overhead imposed by the monitoring framework, the highest monitoring frequency from our prior experiments was used, i.e., one test every 5 seconds. The monitoring overhead was assessed by measuring communication and computation performance metrics on the specified machines with and without C'mon being active. The comparison between performance with and without C'mon is conducted using a two sample Kolmogorov-Smirnov (K-S) test [14]. If the null hypothesis (that the two sample sets of performance measurements belong to the same population) cannot be rejected, then there was no significant performance overhead imposed by C'mon. Otherwise, an overhead was observed and we report the average difference between the values as the imposed overhead. The conducted performance tests are detailed below.

5.3.1 Communication Overhead. The metric for measuring the communication overhead is *network throughput*. Network throughput was measured using the netperf [8] benchmark. The server under measurement executed a netperf client and the *TCP_STREAM* benchmark profile was used to transfer data and report the throughput of the uplink during the transmission. Three tests were performed, one for the monitoring server and one for each target instance. Each test was performed once while monitoring was active and once while it was inactive. For every test for both cases (monitoring in/active) the test has been performed 30 times yielding 30 throughput samples per test. The K-S tests are performed on the collected samples from each test using a statistical significance level of $\alpha = 0.05$. The results are shown in Table 3. For all three cases the p-value is greater than α. Thus, the null hypothesis cannot be rejected, which shows that there is no evidence of statistically significant communication overhead using C'mon.

Table 3: Network Throughput K-S Tests Results

	Monitoring Server	Instance	Dashboard
p-value	0.236	0.954	0.132
D	0.267	0.333	0.300

5.3.2 Computational Overhead. To measure the performance of computational workloads, two metrics were chosen: *CPU total time* and *file Input/Output (I/O) throughput*. We used Sysbench [9] to measure the chosen metrics. The CPU total time was measured for the monitoring server, the instance, and the machine running the dashboard, whereas file I/O throughput was only measured for the monitoring server and the dashboard, as the ping-based SLO measurements do not impose file I/O overheads on the IaaS instances. Each test has been performed 30 times for each machine and monitoring being active or inactive. K-S tests are used to compare the samples collected for each case using $\alpha = 0.05$ as for the communication overhead assessment before.

The results of the K-S tests are listed in Table 4. According to the p-values of the CPU total time, tests for both the monitoring server and the server hosting the dashboard, the null hypothesis is rejected. This implies an imposed overhead on the server caused by monitoring. Figures 2(a) and 2(b) show the cumulative frequency distributions of the CPU time samples collected for the monitoring server and the dashboard server respectively. We estimate the imposed overhead as the average difference between the measurements with and without monitoring, which is 1.28 seconds, i.e., a 0.0076% increase for the monitoring server. For the dashboard server, the monitoring overhead is 0.072 seconds, i.e., a 0.0003% increase. For the CPU time test performed on the instance, the null hypothesis of the K-S tests could not be rejected. Hence, there is no evidence of any statistically significant overhead.

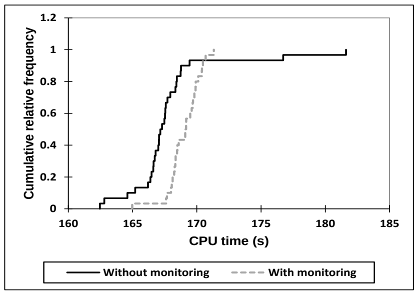
Table 4: Computational Performance K-S Tests Results

	CPU Total Time			I/O Throughput	
	Monitoring Server	**Instance**	**Dashboard**	**Monitoring Server**	**Dashboard**
p-value	< 0.0001	0.007	0.393	0.219	< 0.0001
D	0.633	0.433	0.233	0.267	0.567

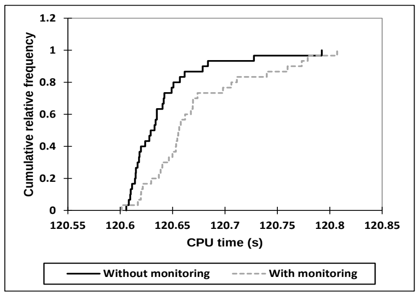
The results of the file I/O throughput test performed for the monitoring server also yields no evidence of significant overhead. However, an overhead is observed on the dashboard for the same test. Figure 2(c) shows the cumulative frequency distributions of the throughput for both cases (monitoring active/inactive). The average difference between the distributions is 0.045 Mb/sec, i.e., a 0.03% decrease resulting from monitoring.

5.4 Threats to Validity

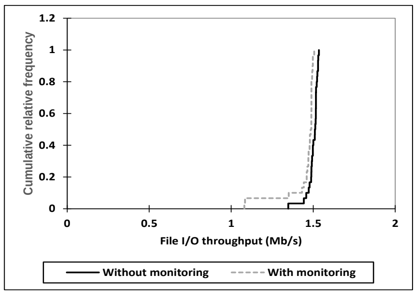
The first threat to validity is the assumption made about the minimum outage duration. The way an instance is managed (launched, scheduled or booted) differs from one CSP to another, hence, the expected outage durations of the services differ. The absence of false negatives observed in our experiments depends on whether the violation duration is longer than the time interval between two consecutive monitoring events. Hence, the coverage of violations



(a) Monitoring Server CPU Total Time



(b) Dashboard CPU Total Time



(c) Dashboard File I/O Throughput

Figure 2: Monitoring Overhead

is dependent on the monitoring frequency. High monitoring frequencies yield high coverage, but also entail high overheads. In our evaluation we conservatively chose high monitoring frequencies and still obtained modest performance overheads for C'MON. We are therefore confident that C'MON provides reasonable performance in any realistic use case.

The second threat is caused by assuming that any request failure is caused by an availability of the service at the CSP side. Although, the availability of the service defined in [2] based on the status of requests sent to it, there is a chance that the failure of the request is caused due to the failure of the path taken to reach the service, not the failure of service itself. On the one hand, this means that CSPs are not necessarily responsible for secSLA violations observed by C'MON, as these may also be caused by perturbations in the network infrastructure. On the other hand, this still accurately reflects the service level as perceived by the user. Therefore, the threat to validity only affects conclusions regarding CSPs' provided service levels. The service level observed by the user is accurately reflected by C'MON's monitoring results.

A third threat concerns the measurement defined to check for the enforcement of forward secrecy. The measurement considers variants of Diffe-Hellman key exchange as the only algorithms which support forward secrecy and the enforcement is decided according. The existence of other algorithms might threaten the validity of the "not enforced" result.

Finally, a fourth threat is the choice of CSPs. The approach was only evaluated on OpenStack and Amazon EC2. The validation of services, which are differently set up and managed, might yield different results.

6 CONCLUSION

SecSLAs have been proposed for managing security assurance in the cloud model by matching the security specifications of the service to the customers' security requirements. However, mechanisms to validate enforcement of these specifications throughout the life cycle of the service are still needed. Although many approaches help CSPs monitor their compliance to the secSLA to take corrective actions in case of violation, existing approaches do not allow customers to manage the compliance validation process themselves, thereby limiting their ability to assess whether contracted security levels are actually provided. In this paper, we proposed C'MON, a customer side monitoring framework for monitoring the compliance of cloud services to the contracted properties in secSLAs.

The proposed approach has been evaluated on both a self-hosted OpenStack platform and a commercial IaaS cloud service. The results have proven our approach suitable for measuring the values of the SLOs and identifying violations of contracted SLO values. Moreover, the results of the overhead evaluation showed that C'MON imposes very low CPU, file I/O, and network bandwidth overheads on both the monitored and the monitoring machines, which makes it a practical solution for IaaS secSLA compliance validation.

REFERENCES

[1] Amazon. 2017. Amazon web services. http://aws.amazon.com/ec2/. (2017).

[2] CUMULUS Consortium. 2013. *D2.1 Security-Aware SLA Specification Language and Cloud Security Dependency model*. Technical Report. Certification infrastrUcture for MUlti- Layer cloUd Services. http://www.cumulus-project.eu/index.php/public-deliverables

[3] Tim Dierks and Eric Rescorla. 2008. The transport layer security (TLS) protocol version 1.2. https://www.ietf.org/rfc/rfc5246.txt. (2008).

[4] Slava Galperin, Stefan Santesson, Michael Myers, Ambarish Malpani, and Carlisle Adams. 1999. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP . https://www.ietf.org/rfc/rfc2560.txt. (1999).

[5] Irfan Haq, Ivona Brandic, and Erich Schikuta. 2010. Sla validation in layered cloud infrastructures. In *Proc. of GECON*. 153–164.

[6] Dan Harkins and Dave Carrel. 1998. The Internet Key Exchange (IKE). https://tools.ietf.org/html/rfc2409#section-3.3. (1998).

[7] Jeff Hodges, Collin Jackson, and Adam Barth. 2012. HTTP Strict Transport Security (HSTS). https://tools.ietf.org/html/rfc6797. (2012).

[8] Rick Jones. 2017. Netperf Homepage. http://www.netperf.org/netperf/. (2017).

[9] Alexey Kopytov. 2017. Sysbench. https://github.com/akopytov/sysbench. (2017).

[10] Duo Liu, Utkarsh Kanabar, and Chung-Horng Lung. 2013. A light weight SLA management infrastructure for cloud computing. In *Proc. of CCECE*. 1–4.

[11] Jesus Luna and Marina Bregu. 2014. *Report on requirements for Cloud SLA negotiation*. Technical Report Deliverable 2.1.2. SPECS Project. http://www.specs-project.eu/publications/public-deliverables/d2-1-2/

[12] Jesus Luna, Ahmed Taha, Ruben Trapero, and Neeraj Suri. 2015. Quantitative Reasoning about Cloud Security Using Service Level Agreements. *IEEE Trans on Cloud Computing* 99 (2015).

[13] Suryadipta Majumdar, Taous Madi, and others. 2015. Security Compliance Auditing of Identity and Access Management in the Cloud: Application to OpenStack. In *Proc. of CloudCom*. 58–65.

[14] Frank Massey. 1951. The Kolmogorov-Smirnov test for goodness of fit. *J. Amer. Statist. Assoc.* 46, 253 (1951), 68–78.

[15] Massimiliano Rak, Salvatore Venticinque, Gorka Echevarria, Gorka Esnal, and others. 2011. Cloud application monitoring: The mosaic approach. In *Proc. of CloudCom*. 758–763.

[16] Kazi Ullah and Abu Ahmed. 2014. Demo paper: Automatic provisioning, deploy and monitoring of virtual machines based on security service level agreement in the cloud. In *Proc. of CCGrid*. 536–537.

[17] Kazi Ullah, Abu Ahmed, and Jukka Ylitalo. 2013. Towards building an automated security compliance tool for the cloud. In *Proc. of TrustCom*. 1587–1593.