A Security SLA-Driven Moving Target Defense Framework to Secure Cloud Applications

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

Massimiliano Rak
University of Campania Luigi Vanvitelli
Aversa, Italy
massimiliano.rak@unicampania.it

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

ABSTRACT

The large adoption of cloud services in many business domains dramatically increases the need for effective solutions to improve the security of deployed services. The adoption of Security Service Level Agreements (Security SLAs) represents an effective solution to state formally the security guarantees that a cloud service is able to provide. Even if security policies declared by the service provider are properly implemented before the service is deployed and launched, the actual security level tends to degrade over time, due to the knowledge on the exposed attack surface that the attackers are progressively able to gain. In this paper, we present a Security SLA-driven MTD framework that allows MTD strategies to be applied to a cloud application by automatically switching among different *admissible* application configurations, in order to confuse the attackers and nullify their reconnaissance effort, while preserving the application Security SLA across reconfigurations.

KEYWORDS

Security SLA, secure cloud applications, security reconfiguration, cloud moving target defense

ACM Reference Format:
Valentina Casola,, Alessandra De Benedictis, Massimiliano Rak, and Umberto Villano. 2018. A Security SLA-Driven Moving Target Defense Framework to Secure Cloud Applications. In *5th ACM Workshop on Moving Target Defense (MTD '18), October 15, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3268966.3268975

1 INTRODUCTION AND MOTIVATION

Today the cloud computing paradigm is widely adopted in many application domains and has quickly become a must-have enterprise technology. According to the report recently published by the influential American market research Forrester, the total global public cloud market will be $178B in 2018, up from $146B in 2017, and will continue to grow at a 22% compound annual growth rate, and more than 50% of global enterprises will rely on at least one public cloud platform to drive their business. This notable growth has led to the explosion of research studies targeting cloud infrastructures and focusing on the related technical challenges, ranging from architectural and structural aspects to security and performance issues. With regard to security, despite the high interest shown by the research community in the last years, and the variety of solutions that have been proposed to protect cloud infrastructures, many challenges remain open, and there is still some reluctance, in prospective cloud customers, to trust cloud providers when sensitive data management is involved.

The adoption of Security Service Level Agreements (Security SLAs), formally stating the security guarantees offered to customers by a given cloud service, represents an effective solution to the above problem, as pointed out by several recent research projects (e.g., CUMULUS [23], A4Cloud [24], SPECS [27], MUSA [20]). In previous work ([3–5]), we showed how to automate the development and deployment of a cloud application subject to a given Security SLA, and we highlighted how the security level offered by an application depends not only on how the application has been developed internally, but also on the cloud resources selected for deployment and on their intrinsic security. The introduction of Security SLAs implies that the granted level of security is preserved during service operation, in order not to incur penalties due to SLA violation. Even if security policies declared by the service provider are properly implemented before the service is deployed and launched, security incidents may still happen, due to existing vulnerabilities, resulting in a security degradation.

Incident response strategies may be applied to mitigate the effects of security incidents, but the adoption of proactive strategies aimed at avoiding, as much as possible, that perpetrated attacks result successful, are highly desirable. Moving Target Defense (MTD) [17] is a very promising proactive defense approach, which aims at anticipating the attackers' moves by continually changing the attack surface of the system, defined in [19] as the set of system methods, channels, and data that can be exploited to perpetrate an attack. With a dynamic attack surface, the reconnaissance effort that typically precedes the actual attack and that enables the attacker to gain knowledge on the vulnerabilities of the system is almost nullified, and the uncertainty on the new target to attack is increased.

As discussed in Section 2, several researchers have proposed the adoption of MTD techniques to secure cloud environments, and many of the existing proposals focus on dynamically changing the deployment configuration (i.e., the virtual machines - VMs - used to deploy a cloud service) in order to counteract denial of service attacks. In fact, cloud applications and in particular those relying

MTD '18, October 15, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-6003-6/18/10…$15.00
https://doi.org/10.1145/3268966.3268975

upon Infrastructure-as-a-Service (IaaS) services (i.e., leasing VMs from a provider) are particularly prone to attacks targeting VMs due to the relatively limited VM offer that is available today: VM instances offered by different providers are quite homogeneous and, as a matter of fact, help decrease the attacker's uncertainty on potential targets.

In this paper, we aim at investigating how to exploit the potential of MTD strategies in a cloud system that relies upon Security SLAs to deliver security-guaranteed services, by overcoming existing practical limitations to their wide adoption. As mentioned before, the level of security that can be ascribed to a cloud service (represented by a formal Security SLA) is a combination of the policies implemented internally by the service, of the intrinsic security of the cloud resources used for deployment, and of the relationships existing among all the involved system components: how to preserve a Security SLA in presence of MTD techniques, which require a change in the system configuration?

To answer this question, in this paper we propose a *Security SLA-driven MTD framework* that enables (i) to determine, in an automated way, the set of admissible service configurations that respect a given Security SLA and (ii) to switch, automatically, from an admissible configuration to another according to a reconfiguration strategy. Admissible configurations are obtained starting from a high-level model of the application and by applying novel techniques that enable to take into account all the relevant aspects discussed above to determine the security level of a cloud application. The remainder of the paper is organized as follows. In Section 2, we discuss some relevant existing work on the application of MTD techniques to cloud environments. In Section 3, we introduce our approach to MTD for cloud applications, and in Section 4 we sketch the proposed Security SLA-driven MTD framework. Finally, in Section 5, we draw our conclusions and discuss future research directions.

2 RELATED WORK

The application of moving target defense strategies in the cloud environment has been addressed by several research studies in the last years. Most of the proposed strategies entail a reconfiguration of the VMs used for the service deployment, and consider proper migration techniques to ensure the service correct operation.

The authors of [25] consider a scenario where a cloud service is deployed on one or multiple VM instances belonging to a pool, and the attacker repeatedly probes and attacks the VM pool in order to disrupt the service with the aim of capturing the VM instances on which the service is actually deployed. In order to thwart this attack, the authors propose an MTD strategy according to which, at pre-determined time intervals, an active VM decides both whether to migrate and the migration target based on a probability function that takes into account an estimate of the attacker's capability and of the exposed attack surface of all the instances, defined in terms of their externally accessible resources.

The work presented in [18] specifically addresses denial of service and distributed denial of service attacks, by proposing an MTD strategy consisting in a selective replication of attacked server instances at different network locations, with the aim of isolating persistent bots that follow the moving replica servers to continue a DDoS attack. The affected client sessions are migrated to the replacement replica servers and the attacked servers are taken offline and recycled after client migration is completed. The authors designed a family of algorithms for optimally reassigning clients from the attacked replicas to the new shuffling replica servers.

Live VM migration is considered also in [2], where the authors propose the adoption of the ANCOR framework to create an MTD platform in which any running component of an IT system (i.e., any instance or cluster of instances) can be replaced with a pristine version by leveraging configuration management tools.

The authors of [8] propose to use the network programming capabilities of Software-Defined Networking (SDN) to implement MTD strategies. The presented system architecture includes four modules, devoted respectively to attack and information gathering, attack analytics, countermeasure selection and enforcement, and policy conflict checking and resolution. In particular, the last module ensures that the implemented reconfigurations (consisting in VM migration and subsequent reshaping of flow rules) are consistent with the overall security policy and do not cause unexpected side-effects as a result of conflicts with other flow rules present in the system. The adoption of SDN capabilities to efficiently implement node migration is also proposed by the authors of [30], where cloud monitoring tools are used to detect deviations from the expected behavior in order to trigger reconfiguration.

A more general approach is presented in [16], where the application of Software Behavior Encryption (SBE) to cloud environments is discussed. SBE consists in obfuscating the system to attackers by means of software diversity and redundant version shuffling, and the authors of this paper propose to apply diversity in the implementation of the software (Java, C, C++, etc), the operating system in which it runs (Windows, Linux, etc.), and in the physical location of the virtual machines in which the software is running. The authors propose to evaluate the effectiveness of the above strategy by measuring the resiliency of the system in terms of attack surface, confidentiality, integrity, availability and survivability.

An interesting approach to the evaluation of MTD strategies is presented in [15], where a hierarchical attack representation model (HARM) is used to model the different MTD techniques. In particular, VM live migration, OS diversity and VM redundancy are analyzed to measure the related effects on performance and security. Based on the same approach, combinations of MTD strategies are analyzed in [1].

To the best of our knowledge, independently on the specific suggested MTD technique, existing approaches do not take into account the intrinsic security level that unavoidably characterizes a certain system configuration when determining what are the admissible configurations. When present (in very few cases), the discussion on the admissible configurations to activate only refers to the technical issues that possibly exist. Our approach, on the contrary, takes into account the level of security provided by each possible cloud application configuration, modeled in terms of Security SLAs, and ensures that an SLA is preserved across reconfigurations.

3 APPLYING MTD TO CLOUD-BASED SERVICES

In this section, we introduce our approach to implementing moving target defense within cloud-based services subject to Security SLAs. In particular, we first illustrate the models and related modeling formalisms adopted to represent cloud applications and Security SLAs in Section 3.1, and then we summarize the main MTD techniques and discuss their application to cloud-based services in Section 3.2.

3.1 Cloud applications and Security SLAs

At a high level of abstraction, a *cloud application* can be seen as a collection of cooperating software components. In general, each application component offers a specific service and may be mapped to a cloud resource belonging to one of the three well-known cloud service types, namely IaaS (Infrastructure-as-a-service), SaaS (Software-as-a-service) and PaaS (Platform-as-a-service). For simplicity sake, in this paper we will only consider the case in which application components are COTS or custom pieces of software that must be deployed on a virtual machine (IaaS) to be executed.

In order to model cloud application deployments and enable their automated security assessment, which is fundamental to determine the admissible configurations for MTD application, we exploit a simple yet powerful modeling formalism introduced in [26], i.e., the *Multi-cloud Application Composition Model (MACM)*. MACM is a graph-based representation that allows to describe, in a simple and immediate way, the logic building blocks of an application, the associated security guarantees (in terms of Security SLAs), the relationships existing among application components and the information related to their deployment.

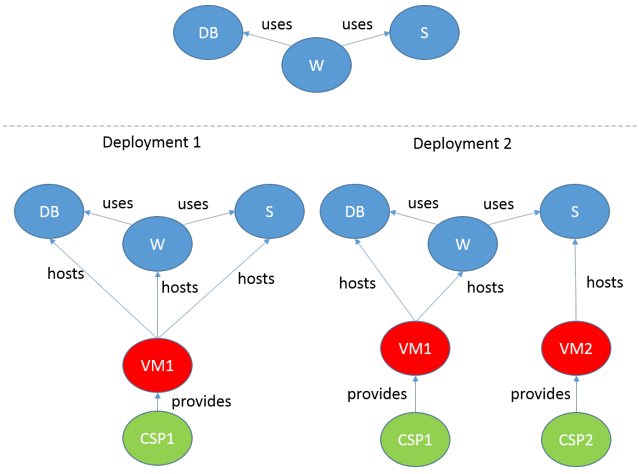

Figure 1: An example of MACM representation

Figure 1 shows, on the top, the *deployment-independent MACM representation* of a simple cloud application made of three logic components, namely a web application `W`, a storage component `DB` and a generic service `S`, where the *use* relationship is outlined between `W` and `DB` and between `W` and `S`. On the bottom, the figure shows the *deployment-specific MACM representation* of the same application in two different deployment configurations. It is worth noting that the two deployment configurations expose a different attack surface and are able to provide a different set of security guarantees, even if the implementation of components does not change. The formalism makes it possible to model cloud service providers (`CSP1` and `CSP2` in the figure), virtual machines (`VM1` and `VM2` in the figure), and the relationships existing among these nodes: a CSP node *provides* a VM, and a VM *hosts* a component. In addition, the formalism allows to specify properties (not shown in the figure) of both nodes and edges, which provide additional information (e.g., the type of a component and its resource demand, the protocol used for components' communication, the IP address of a VM, etc.) used for assessment and deployment operations.

As said, our approach relies upon the adoption of Security SLAs as a means to model and assess the level of security granted by a given cloud application deployment configuration. The Security SLA model we adopt in this paper has been formalized in [3]. According to this model, a Security SLA includes a *declarative* section and a *measurable* section. In the declarative section, the security policies implemented by the service, referred to as the *security capabilities*, are specified in terms of standard *security controls*, belonging to a standard security control framework (such as the NIST Security Control Framework [22] or the Cloud Security Alliance's Cloud Control Matrix [11]). In the measurable section, the actual security guarantees offered by the service are specified in terms of Security Service Level Objectives (SLOs). Security SLOs are represented by boolean conditions involving suitable *security metrics*, which are directly mapped to the security capabilities declared in the SLA and that can be measured, by means of proper monitoring tools, in order to verify the fulfillment of the SLA.

With regard to security, the MACM formalism introduced above allows to distinguish among the *actual* security guarantees associated with each application component, represented by the respective Security SLA, and the *potential* security guarantees that each component is able to grant only based on its internal implementation, without taking into account the impact of deployment on the final security level. These potential guarantees are stated in an *SLA Template*, which is explicitly considered by the MACM formalism to annotate the application model with security information.

Figure 2 reports an example of *security-annotated deployment-specific MACM representation* for one of the deployment configurations of the simple application introduced above. In particular, the example refers to a deployment configuration whose actual level of security has not been assessed yet, since only the SLA *granted* by `CSP1` (i.e., SLA_{CSP1}) is available, while for the other components only the SLA Templates they *support* are depicted. As briefly summarized in the next section and widely discussed in [26], by suitably processing the MACM representation of an application, it is possible to assess the actual level of security granted by each component of the application and by the application as a whole, by taking into account the structure of the application, the enforced security policies (i.e., the available SLAs and SLA Templates) and the impact of the deployment choices.

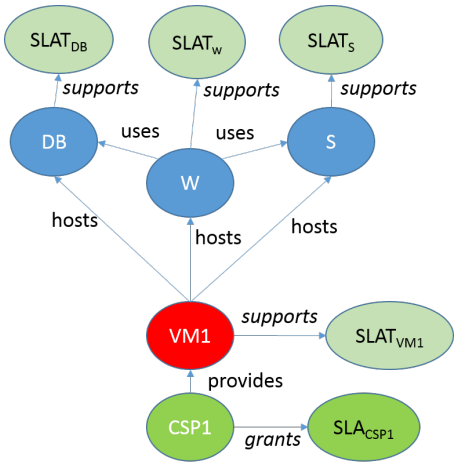

Figure 2: An example of security-annotated MACM representation

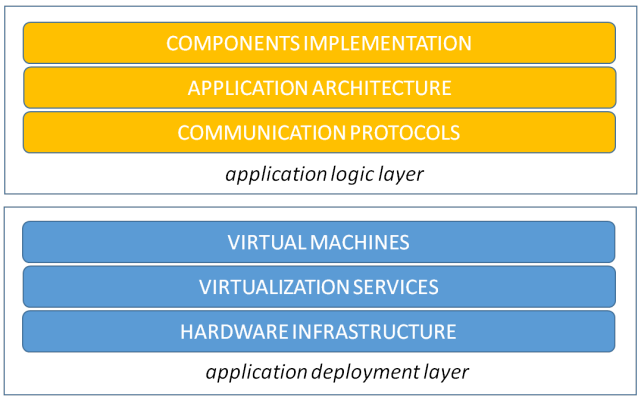

Figure 3: Reconfigurable layers of a cloud application

3.2 MTD strategies and cloud services

MTD can be applied to a generic system in several different ways. In particular, as outlined in [15], three main different MTD techniques can be identified:

(1) *diversity*: different implementations of the system (e.g., in terms of operating systems, application servers, binary image, etc.) are used to deliver the same functionality. The different implementations are not affected by the same vulnerabilities and weaknesses, and it is difficult for an attacker to use the knowledge possibly gathered on a system implementation to damage also the other versions. We include in this technique not only the usage of different versions of the same system, but also the adoption of different input parameters (e.g., a different key used by the same cipher component) and of different algorithms and protocols that slightly change the internal behavior of the system while preserving the same external functionalities.
(2) *redundancy*: multiple replicas of the system (services, nodes, paths) are used to deliver the final service to customers. The way replicas are actually invoked/activated is unpredictable for the attacker, which cannot target a specific replica to attack. This technique is particularly suited against denial of service attacks.
(3) *shuffle*: system settings at various layers are rearranged during system operation (e.g., address randomization, instruction set randomization, live VM migration, topology rearrangement). This technique is maybe the most effective one, but it is also the most expensive and complex and it is not always viable in real systems.

When dealing with a cloud application, all of the above three techniques can be applied, even in combination, at several layers. In particular, as shown in Figure 3, we can consider two main logic layers for MTD application to cloud-based services, namely the *application logic layer* and the *application deployment layer*.

When considering the application logic layer, the attack surface may be changed by acting on the application architecture in terms of the nature and number of components, the internal behavior and/or implementation of components, and the protocols used for communication. On the other side, applying MTD at the application deployment layer means updating dynamically the *deployment configuration*, namely the mapping of components to VMs. This includes updating the VM settings, the virtualization technologies and the underlying hardware layer. Note that this may even imply the migration over VMs hosted on different hardware infrastructures and managed by different providers.

The flexibility and automation level provided by the cloud technologies make the application of redundancy (a cloud application component is replicated over multiple VMs) and diversity (multiple types of VMs are used in the deployment both during time and in parallel) strategies quite easy from the technical point of view, as also demonstrated by the high number of solutions of this kind existing in the literature. Nevertheless, in the application of these strategies, reconfiguration must be carefully designed, since the activation of a new configuration may actually reduce the level of security achieved by the system instead of increasing it, due to the specific security policies implemented by each provider, to the peculiar features of the hosting VMs, and to the specific relationships existing among the application components.

Although presenting a specific reconfiguration strategy is not the objective of this paper, for the sake of completeness hereafter we will refer to concrete strategies for the application of MTD both at the application logic layer and at the deployment layer. As for the application logic layer, we will consider the combination of diversity and redundancy techniques based on the contemporary adoption of different implementations of the same application component, presenting the same external interface but realizing a different internal behavior. More specifically, with reference to the example application introduced above and depicted in Figure 1, we designed an MTD strategy consisting in replicating the web application component, installing different application servers on each replica, and periodically switching among the different replicas to serve client requests. Such MTD technique can be easily implemented by setting up a pool of synchronized web servers, managed

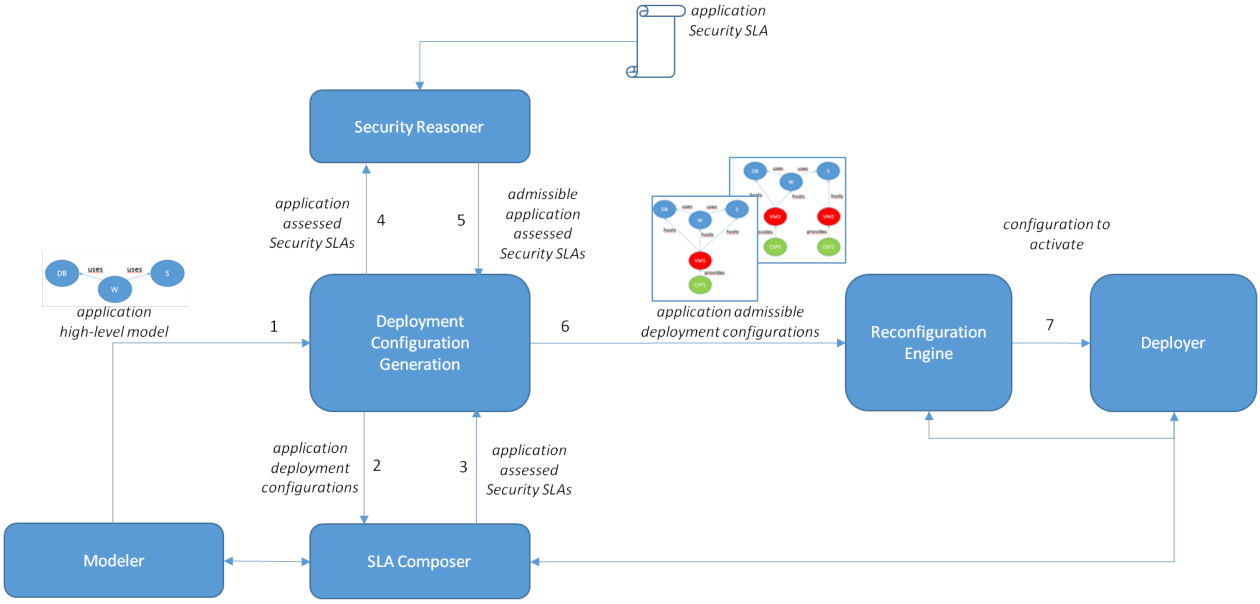

Figure 4: Security SLA-driven MTD framework

by a proxy component acting as a load balancer and request forwarder. At current state, our implementation supports two different web servers, namely Apache [29] and Nginx [21], and relies upon HAProxy [31] to perform load balancing. Since it is not guaranteed that the requests starting from the same client are handled by the same web server, we used Memcached [13] (a general-purpose distributed memory caching system) to store session information.

At the deployment layer, we will consider the adoption of different deployment configurations involving different VM instances to host the web server replicas, leased from multiple providers and characterized by different capabilities and security properties. The acquisition and configuration of VMs is managed by a component of our framework, as discussed in Section 4.

In the next section, we will sketch the architecture of our Security-SLA driven MTD framework, which enables to identify the admissible deployment configurations able to fulfill a given Security SLA and to automatically switch among them in order to ensure proactive defense in line with the MTD philosophy.

4 A SECURITY SLA-DRIVEN MTD FRAMEWORK

The MTD framework discussed in this section has been devised to foster the application of MTD techniques to the cloud computing paradigm, by coping with the need for assessing the level of security provided by each cloud application configuration and for identifying, in an automated way, the application configurations that allow to fulfill a given Security SLA. The proposed framework is composed of six main building blocks (see Figure 4):

- *modeler*: manages and enhances the model of the application in the different stages of its execution, since the initial deployment and during all reconfiguration stages;
- *deployment configuration generator*: generates admissible deployment configurations, i.e., deployment configurations that enable to fulfill the application Security SLA; it retrieves the information on available cloud (IaaS) services and on security guarantees published by service providers from a *service catalogue* and an *SLA repository*, respectively (not shown in the figure);
- *SLA composer*: computes the Security SLA that a given cloud application deployment configuration is able to fulfill;
- *security reasoner*: evaluates the global security level associated with a given Security SLA in terms of a score and compares and ranks different Security SLAs (associated with different deployment configurations) based on the respective scores;
- *reconfiguration engine*: applies a reconfiguration strategy by iteratively selecting the next configuration to activate among the admissible ones;
- *deployer*: automatically acquires and configures the (IaaS) resources needed to launch a new deployment configuration.

The interactions among the above components are reported in Figure 4. The high-level model of the application, built by the application developer through the modeler interface, is supplied to the deployment configuration generator (1), which generates all the possible deployment configurations that satisfy existing resource demand requirements. These candidate deployment configurations are passed to the SLA composer (2), which computes the actual Security SLA that each given deployment is able to fulfill (i.e., the *assessed Security SLA* of each application configuration) and returns them back to the deployment configuration generator (3). It is worth noting that, during this process, the SLA composer and the modeler interact with each other in order to update the current model of the cloud application. Assessed Security SLAs are submitted to the

security reasoner (4), which is responsible for selecting only those that represent a level of security equal or greater than the level of security represented by the application desired Security SLA. The admissible Security SLAs, corresponding to all the deployment configurations that are able to provide at least the required level of security, are then passed back to the deployment configuration generator (5) and constitute the set of admissible configurations that can be used by the reconfiguration engine (6) to implement the MTD strategy. The actual deployment and execution of each configuration are automatically managed by the deployer, which is able to acquire the needed resources and configure them with the proper application components (7). As shown, reconfigurations are triggered iteratively according to a reconfiguration strategy implemented by the reconfiguration engine. It is worth noting that, during the above operations, the high-level model of the application is dynamically updated and enhanced with deployment and security information.

In the following subsections, we will give some details about the generation of deployment configurations, the assessment of configurations' level of security, the comparison of configurations' SLAs, and the configurations' automated deployment and execution. The components responsible for these operations have been partly developed in the context of previous researches conducted by the authors of this paper, and have a fundamental role in enabling the adoption of MTD techniques in cloud applications while preserving Security SLAs across reconfigurations. Finally, we will discuss the existing design issues related to the reconfiguration engine, and we will provide some directions toward its implementation.

4.1 Deployment configuration generator

The deployment configuration generator is responsible for identifying the possible application deployment configurations, namely the possible component-VM mappings, which satisfy the resource demand requirements of the application components. Such requirements are specified by the developer when modeling the application high-level architecture and are reported in the MACM representation as *properties* of respective nodes.

As pointed out in [4], the mapping of application components to cloud resources cannot be treated as a traditional assignment problem, since the number and type of target resources is not predetermined and fixed. Indeed, current cloud service providers typically offer different types of virtual machine instances, each characterized by different computation and storage capabilities and by a different cost. Moreover, available instances are often equipped with a different software stack and are provided with different protection mechanisms, thus offering a different level of security.

The deployment of a cloud application may involve the acquisition of multiple instances of the same type, or even of a heterogeneous set of instances, possibly leased from multiple providers, therefore the number of resources to acquire for deployment is not known a-priori.

Let $C = \{c_1, \cdots, c_n\}$ be the set of the n software components of the cloud application to deploy and let $O = \{o_1, \cdots, o_m\}$ be the set of the m available *offerings*, each representing a different type of VM instance that can be acquired to host application components. Theoretically, any partition of C into a set of $K \leq n$ disjoint subsets is a potential deployment configuration, with each subset representing one VM instance hosting one or more components. Actually, if considering the real constraints of an application, only some of the possible partitions of C represent a feasible deployment configuration, due to the resource demand constraints of each component and the concrete capabilities offered by each instance. Moreover, additional constraints related to cost should be considered in a real deployment.

Several techniques can be adopted for the generation of candidate deployment solutions, including partial/total enumeration approaches and random selection approaches. In [4], we presented a simple algorithm to generate **all** the distinct deployment configurations in the form $< L, Q >$, where L represents a possible partition of C that simply groups together application components into subsets named *VM-sets*, and Q states how each VM-set is mapped to a real offering. Once a new deployment configuration has been identified, verifying whether resource demand or cost constraints are fulfilled is straightforward, assuming that the resource demand of each component on the one hand, and the capabilities and cost of each offering on the other, are known, as well as the maximum cost that the application provider is willing to pay.

The deployment configurations that satisfy existing resource demand and cost requirements are passed to the SLA composer, in order to determine the Security SLAs that they are actually able to grant.

4.2 SLA composer

The SLA composer takes in input the MACM representation of an application deployment configuration, and builds the Security SLA associated with the application in that configuration. The SLA composition process has been widely discussed in [26]. It is aimed at assessing what are the security controls that are correctly implemented by the application and that can be included in the application Security SLA: a security control is considered as "correctly enforced" by an application if and only if it is declared in the SLA granted by each of the application components for which it is relevant, including logic components, VMs and providers.

In order to clarify the impact of the deployment configuration on the achieved security and to demonstrate the need for composition, let us consider an example. Control `AC-3 - ACCESS ENFORCEMENT` of the NIST framework [22] requires that "*The information system enforces approved authorizations for logical access to information and system resources in accordance with applicable access control policies*". Such control can be implemented in several ways, even in accordance with some of the related *enhancements* proposed by NIST, which provide better specification of the measures and techniques to adopt (e.g., enhancement (7) requires the adoption of role-based access control). Related to the deployment configuration marked as "Deployment 1" in the simple cloud application example shown in Figure 1, if `DB` implements an access control policy but the hosting `VM` does not, we have to state that the deployed `DB` does not correctly implement control `AC-3`. It is worth mentioning that, from the point of view of the `DB` component and of its specific requirements, if both `DB` and the underlying `VM` correctly implement an access control policy, it is not relevant whether it is also implemented by `W`, that uses the `DB`, to state that the access control policy

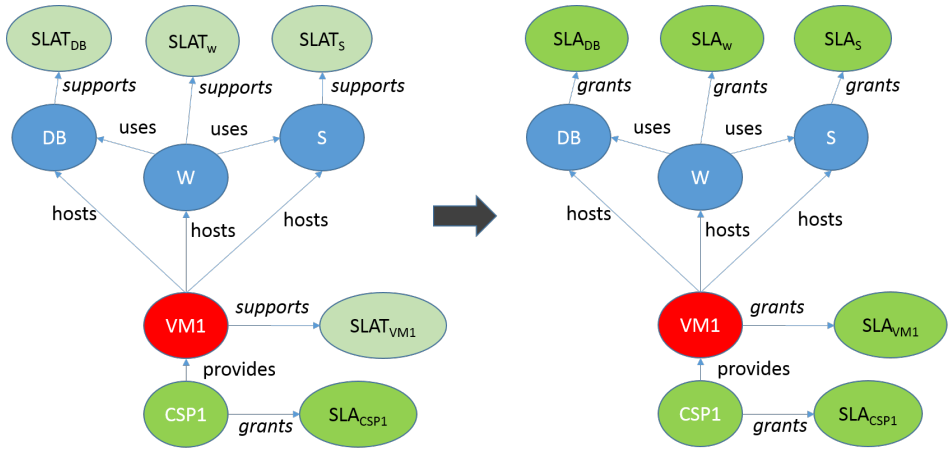

Figure 5: SLA composition

requirement is covered by DB. Nevertheless, when considering node W and the application as a whole, AC-3 is correctly implemented if and only if it is enforced by W, DB and VM, since all the possible access points to the system must be protected. In this regard, going back to the MTD strategy introduced in Section 3.2, which relies upon multiple (diverse) replicas of W, it is worth mentioning that the different web server implementations may adopt different access control enforcement policies, and they may result in configurations that are not admissible with respect to the enforced Security SLA.

Let us consider a different control, namely PE-2 - PHYSICAL ACCESS AUTHORIZATIONS, which involves the management of a list of individuals with authorized access to the facility where the information system resides. Clearly, in the "Deployment 1" situation, if provider CSP1 properly protects its physical facilities (i.e., the locations where the physical hardware server hosting VM1 is installed), control PE-2 can be considered as correctly implemented by the application. In the second deployment configuration depicted in Figure 1, if CSP2 does not declare PE-2 in its SLA, the controls will be considered as not implemented.

In order to perform the assessment at each application component, suitable composition rules are applied for each security control independently of the others: these rules depend on the type of control under analysis, on the way it is implemented in each application component, on how components interact with one another and on the resources used for their deployment.

In particular, during composition, the application MACM model is translated into a set of Prolog facts and rules, which are dynamically generated, based on the application logic and deployment structure, by relying on a set of pre-built tables including the template rules for each control. The rules take into account and suitably combine (i) the security guarantees that are potentially granted by each component and that are summarized in the respective SLA Templates (resulting from the application MACM representation), and (ii) the security guarantees intrinsically offered by adopted cloud resources and by their respective providers.

The components' SLA Templates can be obtained by applying a security assessment process as described in [4], by means of a questionnaire-based procedure aimed at identifying the security policies enforced by each component internally.

The security level offered by the cloud resources used for deployment, instead, can be obtained, as mentioned in [14], by processing the information that is currently available as a result of international assessment initiatives aimed at evaluating the security offered by existing providers (such CSA STAR program [10] and the Consensus Assessments Initiative Questionnaire (CAIQ) [9], which is the result of a providers' security self-assessment process, promoted by CSA and consisting in a set of yes or no assertions related to the implementation of specific security controls). Alternatively, when this information is not available, it can be obtained by means of a security review process. As said, we assume that the information on available services and on related security properties is available in the service catalogue and in the SLA repository, respectively.

As anticipated, the SLA composer builds the Security SLA related to each deployment configuration produced by the deployment configuration generator, and makes these SLAs available to the security reasoner, described in the following.

4.3 Security reasoner

The security reasoner is able to evaluate the global security level corresponding to a given Security SLA and to compare different SLAs belonging to different application deployment configurations. Thanks to this capability, it is possible to identify the deployment configurations that are able to provide the minimum desired level of security, stated in the application desired Security SLA.

In [14], we showed how to compare different SLAs based on the Reference Evaluation Model (REM) methodology [6], previously introduced to quantitatively measure the level of security of an infrastructure. Security SLAs built by the SLA composer are compared with the desired Security SLA by the security reasoner based on the declared security capabilities and on the levels chosen for the related metrics. The deployment configurations whose Security SLA represents a level of security lower than the desired one are simply discarded, while the others are ordered based on their global security level and passed to the reconfiguration engine.

4.4 Reconfiguration engine

The reconfiguration engine is the entity that determines the new configuration to activate based on a specific reconfiguration strategy. It is worth mentioning that the reconfiguration strategy must take into account several factors:

- the natural degradation of the security level due to the exposure of the attack surface for a given period of time during which an attacker can gain knowledge on it and exploit some exposed vulnerability;
- the possible occurrence of a security incident (or the raise of an alert), which may be detected during the application monitoring phase immediately following the application deployment.
- the risk associated with the specific application; indeed, not all the applications are equally likely to be attacked depending on their mission, the type of enforced security policies, the expected damage, the skills required from the attacker, etc..
- the cost of reconfiguration;
- the overhead introduced by reconfiguration and the application resiliency to the reconfiguration's (transient) adverse effects (e.g., the application may experience some outage period during reconfiguration).

As anticipated, in this paper we are not interested in proposing a specific reconfiguration strategy, which will be addressed in our future work. In general, a probabilistic approach similar to that proposed by [25] may be adopted to determine the configuration that will be less likely attacked in the next round; alternatively, the configuration to activate may be chosen in order to optimize some parameter, as done in [18], by applying an analytical optimization model aimed at minimizing, for example, the reconfiguration cost or at maximizing the distance among subsequent configurations.

In any case, the output of the reconfiguration engine is the MACM representation of the configuration to activate, which is supplied to the deployer based on the reconfiguration strategy. As said, this process is iterative in order to continually change the attack surface of the system.

4.5 Deployer

The deployer is responsible for automatically activating the new deployment configuration selected by the reconfiguration engine based on its MACM representation.

In [5], we presented the cloud application deployer tool developed in the context of the MUSA project [20]. Based on the application deployment-specific MACM representation, the deployer is able to build a deployment plan, specified in the JSON format, which includes all the information needed to automate resource acquisition and configuration. In particular, the plan has a dedicated section for each provider involved in the deployment configuration that contains two sections, including respectively (i) some general infrastructure-related information (i.e., the provider name, the resource Zone, the User needed to access the resource in order to install and configure the software components, and - if supported by the provider - the Network in which the resources have to be configured) and (ii) the list of VMs to acquire along with the list of components to install and configure on them. Configuration automation is supported by the cloud automation tool Chef [7]: in particular, the components are specified in the deployment plan by means of the associated Chef cookbook and Chef recipes. Finally, for each VM, the plan specifies the inbound and outbound access rules.

The deployer acquires the resources by means of a Broker and deploys and configures the application's components based on the recipes specified in the deployment plan for each virtual machine. Afterward, it starts the cloud services and launches the application. The acquisition and configuration of the new resources can be carried out in background, in order to limit as much as possible the reconfiguration overhead. Only when the new configuration is ready the old one can be stopped, in order to ensure service availability.

With regard to the MTD strategies introduced in Section 3.2, a Chef cookbook (the WebPool Cookbook) is available at [28], containing the recipes to install Apache, Nginx and HAProxy, respectively, on different VMs acquired by the Broker according to the deployment plan. It is worth mentioning that, at current state, the most part of the MTD framework is available and the integration of the reconfiguration functionalities is quite straightforward.

5 DISCUSSION AND CONCLUSIONS

Moving Target Defense represents a promising proactive defense technique that can be successfully used to protect systems from targeted attacks, exploiting the knowledge on the system vulnerabilities progressively gained by means of reconnaissance efforts.

MTD basically consists in reconfiguring the system by applying diversity, redundancy and shuffle techniques at several layers. In this paper, we discussed the adoption of MTD techniques in the cloud environment, outlining the need for assessing the level of security intrinsically provided by each application configuration involved in the MTD reconfiguration strategy. In particular, we proposed Security Service Level Agreements as a means to state the security guarantees provided by a cloud application, and presented a Security SLA-based MTD framework that enables (i) to automatically assess the level of security actually guaranteed by a cloud application configuration, (ii) to compare different configurations based on their security level, (iii) to automatically switch among different configurations by acquiring and configuring related cloud resources.

The goal of this paper was to highlight the issues related to the provision of security guarantees in the cloud environment in the presence of the high dynamism introduced by MTD techniques, an aspect that seems to have been overlooked in existing work. As a future research direction, we plan to work on the definition of a novel reconfiguration strategy able to ensure the best trade-off between the security gain and the reconfiguration cost. Moreover, since the evaluation of MTD strategies is still an open issue [12], we plan to work on the definition of a quantitative measure of the attack surface of a cloud application configuration, in order to be able to quantify and compare different MTD strategies.

REFERENCES

[1] H. Alavizadeh, D. S. Kim, J. B. Hong, and J. Jang-Jaccard. 2017. Effective Security Analysis for Combinations of MTD Techniques on Cloud Computing (Short Paper). In *Information Security Practice and Experience*, Joseph K. Liu and Pierangela

Samarati (Eds.). Springer International Publishing, Cham, 539–548.
[2] A. G. Bardas, S. C. Sundaramurthy, X. Ou, and S. A. DeLoach. 2017. MTD CBITS: Moving Target Defense for Cloud-Based IT Systems. In *ESORICS*.
[3] V. Casola, A. De Benedictis, M. Erascu, J. Modic, and M. Rak. 2017. Automatically Enforcing Security SLAs in the Cloud. *IEEE Transactions on Services Computing* 10, 5 (2017), 741–755.
[4] V. Casola, A. De Benedictis, M. Rak, and U. Villano. 2018. Security-by-design in multi-cloud applications: An optimization approach. *Information Sciences* 454-455 (2018), 344 – 362. https://doi.org/10.1016/j.ins.2018.04.081
[5] V. Casola, A. De Benedictis, M. Rak, U. Villano, E. Rios, A. Rego, and G. Capone. 2017. MUSA Deployer: Deployment of Multi-cloud Applications. In *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Vol. 00. 107–112. https://doi.org/10.1109/WETICE.2017.46
[6] V. Casola, A. Mazzeo, N. Mazzocca, and V. Vittorini. 2007. A Policy-Based Methodology for Security Evaluation: A Security Metric for Public Key Infrastructures. *Journal of Computer Security* 15, 2 (2007), 197–229.
[7] Chef Software Inc. 2018. Chef web site. https://www.chef.io/chef/.
[8] A. Chowdhary and D. Pisharody, S.and Huang. 2016. SDN Based Scalable MTD Solution in Cloud Network. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense (MTD '16)*. ACM, New York, NY, USA, 27–36. https://doi.org/10.1145/2995272.2995274
[9] Cloud Security Alliance. 2011. Consensus Assessment Initiative Questionnaire. https://cloudsecurityalliance.org/group/consensus-assessments/.
[10] Cloud Security Alliance. 2011. Security, Trust & Assurance Registry (STAR). https://cloudsecurityalliance.org/star/.
[11] Cloud Security Alliance. 2015. Cloud Controls Matrix v3.0. https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3/
[12] W. Connell, M. Albanese, and S. Venkatesan. 2017. A Framework for Moving Target Defense Quantification. In *ICT Systems Security and Privacy Protection*, Sabrina De Capitani di Vimercati and Fabio Martinelli (Eds.). Springer International Publishing, Cham, 124–138.
[13] Danga Interactive. [n. d.]. The Memcached web site. http://memcached.org/.
[14] A. De Benedictis, V. Casola, M. Rak, and U. Villano. 2016. Cloud Security: From Per-Provider to Per-Service Security SLAs. In *2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)*. 469–474. https://doi.org/10.1109/INCoS.2016.61
[15] J. B. Hong and D. S. Kim. 2016. Assessing the Effectiveness of Moving Target Defenses Using Security Models. *IEEE Transactions on Dependable and Secure Computing* 13, 2 (March-April 2016), 163–177. https://doi.org/10.1109/TDSC.2015.2443790
[16] D. Thebeau II, B. Reidy, R. Valerdi, A. Gudagi, H. Kurra, Y. Al-Nashif, S. Hariri, and F. Sheldon. 2014. Improving Cyber Resiliency of Cloud Application Services by Applying Software Behavior Encryption (SBE). *Procedia Computer Science* 28 (2014), 62 – 70. https://doi.org/10.1016/j.procs.2014.03.009 2014 Conference on Systems Engineering Research.
[17] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang. 2011. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats* (1st ed.). Springer Publishing Company, Incorporated.
[18] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell. 2014. Catch Me If You Can: A Cloud-Enabled DDoS Defense. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 264–275. https://doi.org/10.1109/DSN.2014.35
[19] P. K. Manadhata and J. M. Wing. 2011. *A Formal Model for a System's Attack Surface*. Springer New York, New York, NY. 1–28 pages. https://doi.org/10.1007/978-1-4614-0977-9_1
[20] MUSA Consortium. 2015. The MUSA project web site. http://musa-project.eu/.
[21] Nginx Inc. 2018. Nginx web site. http://nginx.org.
[22] NIST. 2013. *SP 800-53 Rev 4: Recommended Security and Privacy Controls for Federal Information Systems and Organizations*. Technical Report. National Institute of Standards and Technology. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf
[23] A. Pannetrat, G. Hogben, S. Katopodis, G. Spanoudakis, and C.S. Cazorla. 2013. D2.1: Security-aware sla specification language and cloud security dependency model. Technical report, Certification infrastrUcture for MUlti-Layer cloUd Services (CUMULUS).
[24] S. Pearson. 2011. Toward Accountability in the Cloud. *Internet Computing, IEEE* 15, 4 (July 2011), 64–69. https://doi.org/10.1109/MIC.2011.98
[25] W. Peng, F. Li, C. T. Huang, and X. Zou. 2014. A moving-target defense strategy for Cloud-based services with heterogeneous and dynamic attack surfaces. In *2014 IEEE International Conference on Communications (ICC)*. 804–809. https://doi.org/10.1109/ICC.2014.6883418
[26] Massimiliano Rak. 2017. Security Assurance of (Multi-)Cloud Application with Security SLA Composition. In *Green, Pervasive, and Cloud Computing*, Man Ho Allen Au, Arcangelo Castiglione, Kim-Kwang Raymond Choo, Francesco Palmieri, and Kuan-Ching Li (Eds.). Springer International Publishing, Cham, 786–799.
[27] SPECS Consortium. 2013. The SPECS project web site. http://specs-project.eu/.
[28] SPECS Consortium. 2015. The WebPool Cookbook - Bitbucket repository. https://bitbucket.org/specs-team/specs-mechanism-enforcement-webpool/overview.
[29] The Apache Software Foundation. 2018. The Apache HTTP Server Project web site. https://httpd.apache.org/.
[30] M. Villarreal-Vasquez, B. Bhargava, P. Angin, N. Ahmed, D. Goodwin, K. Brin, and J. Kobes. 2017. An MTD-Based Self-Adaptive Resilience Approach for Cloud Systems. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. 723–726. https://doi.org/10.1109/CLOUD.2017.101
[31] Willy Tarreau. 2018. The HAProxy web site. http://www.haproxy.org/.