Towards Self Adaptable Security Monitoring in IaaS Clouds

Anna Giannakou, Louis Rilling, Jean-Louis Pazat, Frédéric Majorczyk, Christine Morin

Towards Self Adaptable Security Monitoring in IaaS Clouds

Anna Giannakou*, Louis Rilling†, Jean-Louis Pazat‡, Frédéric Majorczyk† and Christine Morin*
*Inria, IRISA
†DGA
‡INSA, IRISA

***Abstract*—Traditional intrusion detection systems are not adaptive enough to cope with the dynamic characteristics of cloud-hosted virtual infrastructures. This makes them unable to address new cloud-oriented security issues. In this paper we introduce SAIDS, a self-adaptable intrusion detection system tailored for cloud environments. SAIDS is designed to reconfigure its components based on environmental changes. A prototype of SAIDS is described.**

***Keywords*-security; cloud; self-adaptation; IDS;**

I. Introduction

Nowadays, Intrusion Detection Systems (IDSs) are one of the most vital tools for the security monitoring of information systems. IDSs have been widely used for identifying malicious patterns in network traffic and hosts execution. Recent trends in the virtualization area have made clouds attractive for organizations that choose to outsource their information systems. Cloud environments present unique features like scalability, on-demand availability, multi tenancy and shared resources. Tenants are given the flexibility to create, destroy, migrate and modify virtual machines (VMs) with little to no effort. However such features often affect the ability of the security monitoring system to detect attacks [1]. Indeed a monitoring system specific to a virtual infrastructure hosted in the cloud may include components that are located in the cloud but outside the virtual infrastructure [2], [3], [4], [5]. Moreover those components may only monitor subsets of VMs [2] or parts of the network. Consequently, changes in the number of VMs (*e.g.* addition of new instances) or in their placement (*e.g.* live migration) require the security monitoring system to be adapted. Due to the high frequency of those changes the adaptation process allows little to no input from the cloud administrator or tenants administrators and has to be automated. To this end we propose SAIDS, a self-adaptable intrusion detection system for IaaS clouds. To maintain an effective level of intrusion detection, SAIDS monitors changes in the virtual infrastructure and reconfigures its components accordingly. Our contributions include the analysis of dynamic events and their impact on the security monitoring system, the design of SAIDS, and finally a preliminary evaluation of SAIDS showing its ability to adapt the security monitoring system. The rest of the paper is organized as follows: Section II presents related work. Section III outlines the main objectives of SAIDS and describes the architecture. Section IV provides implementation details while Section V describes a test scenario along with a preliminary evaluation of SAIDS. Section VI concludes the paper focusing on future work.

II. Related Work

With respect to the origin of the analyzed data, IDSs can be classified into two main categories: Host-based IDSs (HIDSs) monitor local activity on a host like application logs, system calls and system services logs, and Network-based IDSs (NIDSs) monitor network traffic. Intrusion detection in the cloud is the focus of numerous research projects. In this section we discuss the different approaches based on their service model (centralized or distributed) and the goal pursued.

Mazzariello et al. [2] present the deployment of an existing lightweight NIDS on the Eucalyptus Infrastructure as a Service (IaaS) cloud. The strategy is to deploy one NIDS next to every physical server, monitoring a portion of the network traffic for a specific number of VMs. However in this approach the number of NIDSs deployed cannot be adapted based on incoming traffic load in order to avoid impacting the quality of detection. Also, there is no customization based on individual tenant needs.

In Roschke et al. [3], the IDS is deployed in the three-layer cloud service model. The proposed IDS consists of several sensors and a central management unit for the whole system. The sensors can either be HIDSs or NIDSs. An HIDS is included in each VM initialized by the tenant, whereas the NIDS sensors are placed on each layer by the cloud administrator. A tenant can interact with and configure the IDS through an IDS remote controller. However this approach introduces a strong dependency between a tenant and the infrastructure provided by the cloud administrator. The cloud administrator has to implement and deploy the key components of the monitoring architecture such as the VM-integrated HIDS, the communication between the sensors and the central management unit. Furthermore, the integration of HIDS in each of the tenants VMs by the cloud administrator raises serious privacy concerns.

Lo et al. [4] focus on reducing the impact of denial-of-service (DoS) attacks by presenting a distributed cooperative NIDS framework for cloud environments. The proposed solution consists of several NIDSs placed on different cloud

regions, that exchange alerts with each other. NIDSs are plugged in different sections of the cloud infrastructure, along with services that are responsible for implementing the inter-NIDS communication. Each NIDS can decide whether to accept or reject an incoming alert by implementing a majority vote. However the system can not be reconfigured based on individual tenant needs.

Doelitzscher et al. [5] propose a solution based on autonomous agents, that are placed inside running VMs or at other key points of the infrastructure (firewalls, virtual switches or data storage) to collect and analyze information. The tenant can access all gathered events through a dedicated security dashboard that provides detailed information about the status of the services deployed in the cloud. The configuration and deployment of the autonomous agents is performed at the beginning of the VM lifecycle with no further alteration ability in case a new service is added or tenants request protection against a specific class of attacks.

The described projects present different global solutions for cloud-specific IDSs that include many local instances of IDSs (lIDSs). For avoiding confusion between the individual instances and the global system from now on we will refer to a local instance as lIDS.

Our main goal is to provide a generic monitoring architecture that can be adapted depending on the frequent environment changes of a cloud-hosted virtual infrastructure. We aim to automate the adaptation process, in order to require minimal tenant input but also provide flexible monitoring options depending on the tenant's deployed services.

III. Overview of SAIDS

We aim at designing a security monitoring system that self-adapts its components based on changes in the virtual infrastructure. We identify these changes and categorize them based on their source. In section III-A we outline the main objectives of SAIDS, while in section III-B we describe both the cloud and attack models that we consider. Finally section III-C presents a categorization of adaptation-related events and a detailed description of SAIDS architecture.

A. SAIDS Objectives

The self-adaptability goal of SAIDS can be refined with the following properties:

- Self adaptation: system components should react to dynamic changes that occur in a cloud environment. These changes include modifications both on the hardware infrastructure (server addition) as well as the virtual one (VM creation, deletion, migration). Phenomena that present high occurrence such as service addition and load fluctuation are also a source of adaptation. To achieve this, SAIDS features probes to detect changes and reconfigure its components accordingly.
- Scalability: the number of deployed monitoring components should adjust to varying conditions: load of the network traffic monitored, number of physical servers in the datacenter, number of VMs in each virtual infrastructure. Scalability is achieved by instantiating new lIDSs when their capacity is exceeded.
- Customization: based on the type of hosted services the tenant should be able to customize the IDS in order to monitor specific vulnerabilities. SAIDS allows tenants to write and include customized IDS rules that target their deployed services.
- Cost minimization: the overall cost in terms of resources should be kept at a low level both for the provider and tenants while enforcing the targeted detection quality. We address this by enabling component sharing between tenants.

B. Models

We briefly describe the system model of the cloud infrastructure and the attack model considered for SAIDS.

1) System model: We consider an IaaS cloud with a cloud controller that has a global view of the system as demonstrated in Figure 1. Customers pay for resources that are part of a multi-tenant environment based on a Service Level Agreement (SLA). Each customer is in control of an interconnected group of VMs that hosts various services. Two types of networks are constructed: one internal between VMs that belong to the same tenant and one external that is accessible also from outside the infrastructure.

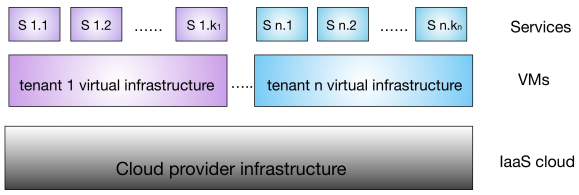

Figure 1. Multi-tenant cloud infrastructure

2) Attack model: We consider software attacks, that originate both from within the cloud infrastructure and from outside sources. We classify attacks in two categories, based on their target. Attacks towards individual services, known as service-level threats (e.g. SQL injection) and attacks towards the infrastructure (virtual or physical), known as system and network level threats (e.g. memory corruption attempts, buffer overflows, spoofing attempts).

C. Architecture

SAIDS is a self-adaptable monitoring system that reacts to dynamic events that occur in a cloud environment. We classify these events into three major categories: virtual infrastructure topology related, performance related and finally events that impact the size of the datacenter. The classification of these events is shown in part A of Table I.

SAIDS consists of three components depicted in Figure 2: local Intrusion Detection Sensors (lIDSs), the Infrastructure

Table I
EVENTS THAT TRIGGER ADAPTATION

Part A			*Part B*
Change category	Event	Origin	Adaptation action
Virtual infrastructure topology	VM creation	Tenant, Provider	{rule update, new lIDS}
	VM destruction	Tenant, Provider	{rule update}
	VM migration	Tenant, Provider	{rule update, new lIDS}
Performance	% Packet drop	Traffic load	{new lIDS}
	Latency		{new lIDS}
Data center size	Server addition	Provider	{rule update, new lIDS}

Monitoring Probe (IMP) and the Adaptation Manager (AM). The flow of the process is as follows: first, the IMP notifies the AM about the events and relates the necessary information. Second, the AM decides on adaptation strategy and third it adapts the lIDSs. The components are run by the cloud provider inside the cloud engine. The AM takes as parameters the topologies of both the virtual and the physical infrastructure and the customization requirements of the tenants as specified in the SLA.

Local Intrusion Detection Sensors collect and analyze network packets that are flowing through subsets of virtual switches. A signature-based detection technique is used because of its high positive rate in detecting known attacks. Furthermore, it requires zero training time making it a suitable choice for immediate efficiency. The packets are decoded and preprocessed in order to check their payload for suspicious patterns by comparing it with a preloaded set of rules. If a match is found the packet is logged and an alert is generated. The rules can match either service or system level threats. Since signatures only match known attacks, the rule database needs to be regularly updated to take new attacks into account. A dedicated update manager is responsible for checking if the existing rule set is up-to-date and downloading the latest available rules.

The Infrastructure Monitoring Probe notifies the AM when a topology change occurs (see part A of Table I). The IMP sends to the AM the ID of the VM that participates in the topology change, the hostname of the compute node that the VM is deployed on (or will be moved to in case of migration) and the IP address of the VM. The IMP is part of the cloud engine.

The Adaptation Manager adapts the lIDSs on the occurrence of the events described in Table I. On a topology change (*e.g.* a VM is migrated) the IMP notifies the AM. The AM then relates the information to the list of services running in that VM (VM info in Figure 2) and selects the set of additional rules to activate in the lIDS responsible for the virtual switch at the VM's new location. This set of additional rules can include specific rules configured by the tenant (SLA info in Figure 2). The AM is also responsible for handling performance degradation of the lIDSs. For example, if the percentage of packets dropped exceeds a threshold (given in the SLA), then a new sensor is deployed and the monitoring of the network traffic is rebalanced.

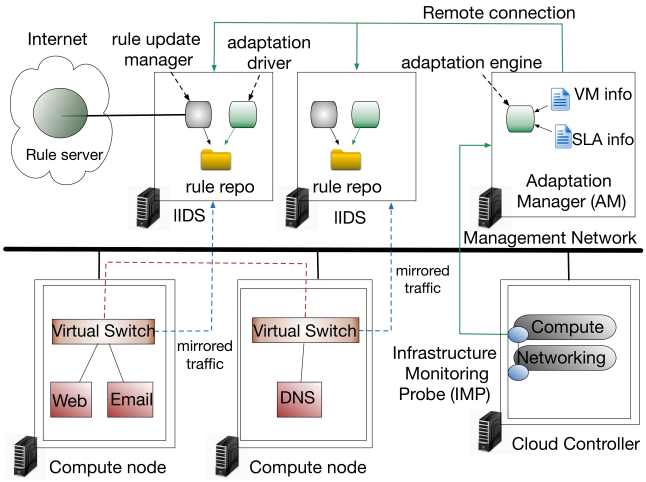

Figure 2. The SAIDS architecture

IV. IMPLEMENTATION OF SAIDS

A prototype of SAIDS was implemented using the Grid5000 experimental platform [6]. We deployed a private cloud using one of the Grid5000 clusters as hardware infrastructure. We used OpenStack [7] as the cloud management system and Open vSwitch [8] as a multilayer virtual switch. The OpenStack cloud computing controller is known as Nova. Network and IP management is done by Neutron.

A. Networking setup

Neutron API allows to create per tenant private networks inside the cloud. To segregate VMs belonging to different tenant networks we used GRE tunnels. A span port was created for mirroring traffic in the virtual switches to the lIDSs. Both features were provided by Open vSwitch.

B. Component Implementation

In this subsection we describe the implementation details for each separate component of SAIDS.

- For lIDSs we deployed Snort [9], an open source NIDS, in VMs on top of a KVM hypervisor. Each Snort sensor is responsible for inspecting all the incoming traffic to a subset of virtual switches. Snort is a signature-based NIDS that features a default set of rules for frequently used services. The activation of additional rules when necessary is done by a dedicated python script (*adaptation driver*) that is executed remotely through ssh from the AM. This driver also restarts Snort so that the new rule set is applied. Oinkmaster [10], a perl script that downloads the latest Snort rules from various locations, was used for regularly updating the rule database. It is instantiated inside each Snort server.
- The IMP is composed of notifiers hooked inside API functions of Nova and Neutron: *allocate_for_instance*

(VM creation), *deallocate_for_instance* (VM destruction), *live_migrate* (VM migration), and *create_port* (to get network information). The connection with the AM was implemented through a ssh tunnel. All the hooks were implemented in python.

- The AM was deployed as a VM in a dedicated server. All relative information regarding tenants' VMs (list of services, compute node hosting the VM, IP of the VM and IP of the Snort server monitoring its virtual switch) is stored in a separate file. The file is updated on each adaptation event (see Table I). Additional sets of rules are also stored per service in the file.

V. Evaluation scenario

We are evaluating experimentally the effectiveness of our approach. We first present a setup manifesting a need for adaptation, followed by an attack scenario and early results.

A. Setup

To do our experiments we deploy a data center with 3 physical servers: one cloud controller and two compute nodes. In the first scenario one tenant has deployed a virtual infrastructure with three services: DNS, web and mail. We place the web server and the email server in compute node 1 (but in separate VMs) and the DNS server in compute node 2. We also deploy two Snort [9] lIDSs VMs (1 and 2), that is one per virtual switch to monitor on the compute nodes. This scenario is representative of a production setup that balances the load of monitored traffic across several lIDS instances [2]. Thus in lIDS 1 only the rules that are related with the monitored services (web and email server) are enabled, whereas in lIDS 2 only the rules that are related with DNS traffic. We use a python script that launches random requests to the services as traffic generator.

B. Attacking scenario

An attacker located outside the cloud tries to mask a php shell injection attempt in various legitimate HTTP requests. The attacker uploads a malicious file through the picture upload service of the web server. Since customized rules against such attempts are activated in the corresponding lIDS VM, an alert is generated. We then decide to migrate the web server to compute node 2. Consequently, in a setup without self-adaptation, the detection process fails because lIDS 2 has no web-server specific rule activated. SAIDS prevents this failure as follows: once the cloud engine starts migrating the VM, the IMP notifies the AM. The AM decides which extra rules need to be enforced in lIDS 2, and modifies the Snort configuration file of lIDS 2 through remote execution of the adaptation driver. Finally Snort is restarted.

We have tested our approach with a 469MB Centos image (1vCPU 2GB of virtual memory) over an 1Gb/s network connection. The VM was migrated between compute nodes having 2 12-core 1.7 Ghz AMD Opteron 246 CPUs each. At the time of the migration there were no outside HTTP requests to the VM and there where no changes being written in memory. The average migration time without the adaptation process was 2.1 sec. With the adaptation the average time was 5.2 sec. We have observed that the adaptation process time is dominated by restarting Snort (average restart time was 2.4 seconds). Indeed each time Snort is restarted the whole ruleset has to be reloaded. Whereas in the current implementation of SAIDS the adaptation process preempts the migration, in the future they will be executed in parallel, which should reduce the overhead.

VI. Conclusion

In this paper, we have identified three major categories of events that require adaptation: topology changes, changes in the size of the datacenter and performance degradation of the intrusion detection sensors. We have described SAIDS, a self adaptable security monitoring framework that takes those events into account. Finally, we have presented preliminary results that show the effectiveness of SAIDS.

In future work we will study more complex security monitoring setups. Furthermore we will combine the security monitoring of both the provider infrastructure and the virtual infrastructures. We will also investigate how to give to tenants partial control over the monitoring framework.

References

[1] N.-U.-H. Shirazi, S. Simpson, A. Marnerides, M. Watson, A. Mauthe, and D. Hutchison, "Assessing the impact of intra-cloud live migration on anomaly detection," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 52–57.

[2] C. Mazzariello, R. Bifulco, and R. Canonico, "Integrating a network ids into an open source cloud computing environment," in *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, Aug 2010, pp. 265–270.

[3] S. Roschke, F. Cheng, and C. Meinel, "Intrusion detection in the cloud," in *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, Dec 2009, pp. 729–734.

[4] C.-C. Lo, C.-C. Huang, and J. Ku, "A cooperative intrusion detection system framework for cloud computing networks," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, Sept 2010, pp. 280–284.

[5] F. Doelitzscher, C. Reich, M. Knahl, A. Passfall, and N. Clarke, "An agent based business aware incident detection system for cloud environments," *Journal of Cloud Computing*, vol. 1, no. 1, 2012.

[6] "Grid5000," http://www.grid5000.fr/.

[7] "OpenStack," version Juno http://www.openstack.org/.

[8] "Open vSwitch," version 2.9 http://openvswitch.org/.

[9] "Snort," http://www.snort.org/.

[10] "Oinkmaster," version 2.0 http://oinkmaster.sourceforge.net/.