Resource-Aware Detection and Defense System against Multi-Type Attacks in the Cloud: Repeated Bayesian Stackelberg Game

Omar Abdel Wahab, Jamal Bentahar, *Member, IEEE*, Hadi Otrok, *Senior Member, IEEE*, and Azzam Mourad, *Senior Member, IEEE*

Abstract—Cloud-based systems are subject to various attack types launched by Virtual Machines (VMs) manipulated by attackers having different goals and skills. The existing detection and defense mechanisms might be suitable for simple attack environments but become ineffective when the system faces advanced attack scenarios wherein simultaneous attacks of different types are involved. This is because these mechanisms overlook the attackers' strategies in the detection system's design, ignore the system's resource constraints, and lack sufficient knowledge about the attackers' types and abilities. To address these shortcomings, we propose a repeated Bayesian Stackelberg game consisting of the following phases: risk assessment framework that identifies the VMs' risk levels, live-migration-based defense mechanism that protects services from being successful targets for attackers, machine-learning-based technique that collects malicious data from VMs using honeypots and employs one-class Support Vector Machine to learn the attackers' types distributions, and resource-aware Bayesian Stackelberg game that provides the hypervisor with the detection load's optimal distribution over VMs that maximizes the detection of multi-type attacks. Experiments conducted using Amazon's datacenter and Amazon Web Services honeypot data reveal that our solution maximizes the detection, minimizes the number of attacked services, and runs efficiently compared to the state-of-the-art detection and defense strategies, namely *Collabra*, probabilistic migration, Stackelberg, maxmin, and fair allocation.

Index Terms—Adversarial artificial intelligence, intrusion detection, game theory, machine learning, data-driven optimization, Moving Target Defense (MTD), honeypots, security risk assessment

1 Introduction

Security concerns have accompanied the notion of cloud computing since its inception until now. On the one hand, it is crucial to assure security in cloud systems since cloud data centres are supposed to provide a safe and secure environment for users and providers to host and access their data and resources. On the other hand, cloud-based systems are exposed to a wide set of security threats; even more than those that target traditional computing systems because of the cloud's virtual and elastic properties [1]. Practically, each operation/communication occurring at the cloud's virtualization layer (e.g., CPU virtualization, memory management, etc.) can be subject of various malicious attacks. Although the common high-level goal of all attackers is to cause financial/performance damage, the low-level objective of each particular attacker varies according to the attack's target and magnitude of damage intended to cause. For example, some attackers might aim to crash the hypervisor to stop the functioning of the whole cloud system, whilst others might be interested in disrupting some particular VMs pertaining to a specific client. We consider in this work a complex and realistic attack scenario in which the cloud system faces multiple simultaneous types of attacks launched by attackers having distinct skills and objectives. The considered attackers are deemed to be intelligent in the sense that they are continuously observing the detection strategies of the cloud systems and updating their attack strategies accordingly with the aim of maximizing their attack success chances.

Problem Statement. Although plenty of Intrusion Detection Systems (IDSs) for cloud-based applications can be found in the literature, most of these techniques are developed and upgraded from traditional detection techniques used in non-cloud environments, which limits their effectiveness when applied in a cloud setting [1]. In fact, the existing detection systems can be classified into three main branches: network-based, host-based, and hypervisor-based IDSs [1]. Network-based systems put the monitoring agents at the network's level to monitor the circulating traffic and recognize any malicious behavior. The fact that these systems operate at the

- *O. Abdel Wahab is with the Department of Computer Science and Engineering, Université du Québec en Outaouais, Gatineau, QC J8X 3X7, Canada. E-mail: omar.abdulwahab@uqo.ca.*
- *J. Bentahar is with the Concordia Institute for Information Systems Engineering, Concordia University, Montréal, QC H3G 1M8, Canada. E-mail: bentahar@ciise.concordia.ca.*
- *H. Otrok is with the Department of ECE, Khalifa University of Science, Technology and Research, Abu Dhabi, UAE. E-mail: Hadi.Otrok@kustar.ac.ae.*
- *A. Mourad is with the Department of Mathematics and Computer Science, Lebanese American University, Beirut, Lebanon. E-mail: azzam.mourad@lau.edu.lb.*

Manuscript received 8 Mar. 2018; revised 29 Dec. 2018; accepted 24 Mar. 2019. Date of publication 27 Mar. 2019; date of current version 12 Mar. 2021. (Corresponding author: Omar Abdel Wahab.)
Digital Object Identifier no. 10.1109/TDSC.2019.2907946

network's layer only makes them unable to catch insider attacks that sneak into the internal virtualized system. To remedy this shortcoming, host-based IDSs deploy the monitoring agents at the VMs' layer to monitor their activities and report any abnormal behavior. The main limitation of this approach lies in the burdens it puts on the users who are required to spend their own resources and efforts to maintain the health of the monitoring agents. To alleviate these burdens, hypervisor-based systems place the monitoring agents at the cloud system's layer and assign to the host hypervisors the role of observing the VMs' system metrics and identifying malicious activities.

Unfortunately, all of the three branches of IDSs suffer from four essential problems that limit their performance in practical cloud systems. First, they are based on the simplistic idea of monitoring and analyzing events without accounting for the malicious strategies of the attackers that take advantage of the cloud's virtual and elastic features to perplex the detection system and complicate the detection process [1]. This raises the need for elaborating an up-to-date intelligent detection technique that considers the strategies of the attackers in its design to increase the awareness of the cloud system and enable it to cope with complex attack scenarios. Second, the current IDSs do not explain how to deal with the cloud system's limited security resources problem in the detection process; thus assuming (directly or indirectly) that the cloud system is able to provide permanent and full detection coverage on all its nodes. However, it is no secret that the magnitude of resources that can be devoted to detection is bounded by a certain budget that is determined in such a way that does not affect the portion of resources consecrated to serving clients. This necessitates thinking of a resource-aware selective detection strategy that distributes the cloud system's detection load among the different VMs so that it respects the limited security resources' budget and maintains at the same time optimal detection effectiveness. Third, the existing IDSs rely on a simplistic attack scenario which supposes that attackers launch their attacks without having prior knowledge of the intrusion detection arrangements adopted by the cloud system. Albeit such an assumption might hold for some limited time, attackers are becoming smart enough to observe the cloud system's detection strategies over time and adjust their own attack strategies accordingly in order to complicate and confuse the detection process. Fourth, the current detection approaches lack for real-time learning about the types and objectives of the attackers targeting the cloud system; which deprives them from valuable information that can be used to adjust and optimize the cloud system's detection strategies over time.

Contributions. The goal of this work is to develop a comprehensive detection and defense mechanism against multi-type attacks in the cloud. To the best of our knowledge, this work is the first that advances such a comprehensive detection and defense strategy against multiple types of attacks in the domain of cloud computing. The proposed solution is presented in the form of a repeated Bayesian Stackelberg game that consists of four phases executed repeatedly to provide the cloud system with incremental and continuous learning about the attackers' strategies and objectives and the VMs' actual security status. The first phase is concerned with evaluating the risk level of each VM on the basis of its potential vulnerabilities, expected threats, and past attack history. Based on the results obtained from the risk assessment phase, the services deployment phase introduces an intelligent defense mechanism inspired by the Moving Target Defense (MTD) concept [2], which migrates services running inside risky VMs toward other more secure VMs. The risky VMs, running no active services, are exploited in the attackers' types recognition phase through deploying honeypots [3] inside them to collect malicious data from attackers. This data is then analyzed using a one-class Support Vector Machine (SVM) learning classifier [4] to determine the types and objectives of the attackers targeting the cloud system. Using this information, we design a resource-aware Bayesian Stackelberg game whose goal is to provide the cloud system with the optimal detection load distribution strategy over the set of VMs that maximizes the detection of simultaneous attacks of multiple types. In summary, the main contributions of this work are:

- Designing and solving a Bayesian Stackelberg game that guides the cloud system to determine the optimal detection load distribution strategy among VMs that maximizes the detection of multi-type intelligent attacks. To the best of our knowledge, this strategy is the first in the domain of cloud computing that is able to maximize the detection in such a complex (yet realistic) attack environment wherein the cloud system is supposed to face simultaneous attacks of different types launched by intelligent attackers.
- Proposing a risk assessment framework that assists the cloud system with evaluating the risk level of each VM and identifying the risky ones that are likely to be targets for attacks. For this purpose, we formulate a risk level determination model that capitalizes on the VMs' potential vulnerabilities, expected threats, and past attack history to make thoughtful decisions.
- Developing an MTD-based defense mechanism that protects cloud services from being successful targets for attackers. This is done by putting forward an intelligent security-oriented live migration strategy that allows the hypervisor (acting on behalf of the cloud system) to migrate the services running inside risky VMs to other more secure ones.
- Putting forward an attackers' types recognition technique that provides the hypervisor with a detailed view of the types and objectives of the attackers targeting the cloud system. This is achieved by developing a honeypots' deployment strategy inside risky VMs to collect malicious data and proposing a one-class SVM learning classifier which analyzes this data and predicts the actual types of the attackers.

The performance of the proposed solution is evaluated using real data from Amazon's public datacenter's [5] and honeypot data from Amazon Web Services (AWSs) [6]. Experimental results reveal that our solution maximizes the detection performance and minimizes the number of attacked services compared to the existing detection and defense strategies, namely *Collabra*, probabilistic migration, maxmin, one-stage Stackelberg, and fair allocation. Moreover, experimental results show that our solution achieves acceptable execution

time and is scalable to the increase in both the number of co-hosted VMs and percentage of co-resident malicious VMs.

Paper Outline. Section 2 reviews the current IDSs proposed for cloud-based environments. In Section 3, we formulate the problem and illustrate the attack model considered in this work. In Section 4, we design the Bayesian Stackelberg game proposed to derive the optimal detection load distribution strategy among VMs and demonstrate how to solve it using the backward induction reasoning. Section 5 explains the details of the repeated Bayesian Stackelberg game that provides a comprehensive detection and defense system. In Section 6, we describe the experimental environment and present experimental results. In Section 7, we provide a detailed discussion on the originality of this work compared to the state-of-the-art. Finally, Section 8 recapitulates the main insights of the paper.

2 Related Work

The main intrusion detection techniques proposed for cloud-based systems can be classified into three major categories: network-based, host-based, and hypervisor-based systems [1].

2.1 Network-Based IDSs

In [7], the authors discuss an intrusion detection framework that monitors network traffic using a cluster-based architecture to support multiple security domains. The basic idea is to export the intra-VM network traffic to be processed by a physical IDS. Moreover, a traffic deduplication technique is advanced to remove redundant network traffic and minimize the overhead.

In [8], a customer-controllable on-demand IDS is introduced. The network interactions among VMs within a pre-defined virtual network are monitored and suspicious activities are registered and analyzed. The performance of the framework is adaptable based on the volume of traffic load in the network, where, for example, the number of IDS components can be adjusted on the basis of the amount of traffic circulating inside the network.

DoS attacks have been studied in [9], where the authors offered a method to trace the botmaster (i.e., administrator of the botnets) back. According to this method, the local network admin of the victim node has to gather data related to the network traffic between the Command-and-Control (C&C) servers and bots as well as the hostname of the C&C server. This data is then communicated to a traceback service whose role is to embed Prebbleware, a piece of code that uncovers its host machine's information, on the communication packets between the botmaster and victim machine.

2.2 Host-Based IDSs

In [10], a multi-tier detection model for large-scale clouds called *Varanus* is proposed. The basic idea is to split VMs into a set of groups using the k-nearest neighbor algorithm based on the similarity among their configuration settings. Thereafter, the VMs belonging to the same cluster exchange their monitoring information and the under-utilized VMs are selected to analyze the whole collected data.

In [11], the authors discuss a host-based intrusion detection technique that selectively monitors (only) the failed system call traces of the VMs. These traces are then analyzed and classified either normal or malicious using k-nearest neighbor. Finally, users are alerted of any malicious activity in their system.

2.3 Hypervisor-Based IDSs

In [4], the authors propose an online anomaly detection technique that operates at the hypervisor's layer. The system architecture consists of four main components, namely the Cloud Resilience Manager (CRM), System Resilience Engine (SRE), Network Analysis Engine (NAE), and System Analysis Engine (SAE). At the first stage, the CRM deployed on each cloud node collects features from the VMs and their local networks and sends this data to the NAE and SRE components. These two latter components employ the one-class SVM to carry out a local anomaly detection.

The authors of [12] advanced Collabra, a distributed IDS that is integrated into Xen hypevisors to preserve the security of the cloud system. Collabra scans each hyper-call made by every application of the VMs to guarantee the integrity of the cloud infrastructure and ensure fail-safe transaction processes. Collabra performs in a collaborative fashion to enhance the results of the real-time detection.

Lombardi and Di Pietro propose in [13] a virtualization-supported detection technique called Advanced Cloud Protection System (ACPS). In ACPS, the system-call invocations of the VMs are constantly watched by an entity situated in the kernel space of the host called *Interceptor*. The suspected activities are then registered and prioritized into a *Warning Pool*. Finally, the *Evaluator* entity inspects these activities to make the appropriate decision on whether there exists a security threat or not.

2.4 Game-Theoretic Approaches for Security Applications

In [14], the authors aim to answer the challenge of efficiently deploying MTD techniques in large-sized networked systems. To this end, they propose to integrate the Shuffle, Diversity, and Redundancy MTD techniques into the Hierarchical Attack Representation Model (HARM) to assess their security levels. They employ as well several importance measures to choose highly important network components on which MTD techniques should be deployed.

In [15], the authors propose an MTD technique to improve the security of Web applications. They modeled the problem as a Bayesian Stackelberg game whose leader is the Web administrator and followers are the hackers which can be of different types. They formulated an optimization problem whose solution guides the administrator to find the configuration switching strategy that best maximizes the system's security while accounting for the associated switching costs.

In [16], a systematic framework is introduced to derive the optimal defense allocation strategies under interdependencies (e.g., assets owned by the same vendor), where such interdependencies are modeled using an interdependency graph. Attackers are thus assumed to take advantage of those interdependencies to attack valuable assets in the network. Specifically, a game is modeled among multiple defenders, each of which is responsible for protecting a set of assets. The objective of each defender is to minimize its own expected loss knowing that the attack probabilities on its assets

depend on its own defense actions, the actions of the other defenders, and the interdependency graph.

In [17], the authors formulate a game theoretical model between an external adversary and a network of decoy nodes and propose a framework of two phases. In the first phase, the interactions between the adversary and one single decoy node are studied, especially the cases where the adversary (1) tries to recognize the decoy node through observing the timing of node responses, and (2) studies the differences in protocol implementations between decoy and real nodes in order to identify the decoy ones. The outcome of this phase is the time for an adversary to learn whether a certain node is real or decoy. In the second phase, games between an adversary trying to discover real nodes in a network of real and decoy nodes are formulated. The outcome of this phase is the optimal policy of the system to randomize the IP address space to complicate the recognition of real nodes.

While MTD-based techniques and Bayesian Stackelberg games have already been used in the security domain, our work is the first that employs them in a cloud computing environment to model the hypervisor-VM relationship, while taking into account that the attacks can be distributed across several VMs to complicate the detection process. Moreover, we propose in this work a risk assessment strategy to evaluate the risk level of each VM, followed by a honeypot deployment technique to collect data from attackers and a machine learning technique to analyze this data and extract relevant knowledge regarding the attacker types' probability distributions. This information is then integrated into a Bayesian Stackelberg game to optimize the intrusion detection decisions. Such a data-driven optimization methodology for improving the security decisions in terms of detecting multi-type attacks is novel and has not been employed yet in the literature.

2.5 Discussion and Unique Features of Our Solution

The existing IDSs suffer from two principal limitations that make them insufficient to deal with practical cloud-based systems. In the first place, they totally ignore the attackers' strategies in the design of the detection system, which minimizes their chances of capturing sophisticated attacks. In the second place, they do not explain how the proposed detection techniques can work under a limited budget of security resources, which restricts their effectiveness in realistic resource-constrained applications. In a recent work [1], a maxmin-based detection load distribution strategy has been developed by our research group. Specifically, a maxmin game is modeled between the attackers trying to minimize the cloud system's detection probability by distributing their attacks over a set of VMs and the hypervisor trying to maximize this minimization by optimally distributing the detection load among VMs. Similar to our current work, this work takes into consideration the strategies of the attackers in the design of the detection system and is able to work using a limited budget of security resources. Beyond this work, our solution is able to deal with a more complex attack scenario wherein attackers are able to observe the cloud system's detection strategies and adapt their attack plans accordingly.

In a preliminary version of this work, we proposed a one-stage Stackelberg game [18] that provides the hypervisor with the optimal detection load distribution strategies; while considering the strategies and abilities of the attacker but abstracting on the types of attackers. This paper builds on and extends our previous work by offering (1) a risk assessment framework that helps the hypervisor determine the risk level on each VM; (2) an MTD-based defense mechanism that intelligently migrates services running inside risky VMs into other more secure VMs; (3) an attackers' types recognition technique that collects malicious data from VMs using honeypots and analyzes them using one-class SVM; and (4) a Bayesian Stackelberg game that accounts for the distributions of the attackers' types in the design of the problem to increase the awareness of the hypervisor and help it optimize its detection load distribution strategies. Finally, our solution runs in a repeated fashion to provide the hypervisor with incremental and continuous learning about the attackers strategies and skills and the cloud system's current security status. The two versions are compared experimentally as well to verify the improvements brought to the work by our new amendments.

3 Problem Formulation

We illustrate in this section the problem formulation and explain the attack model considered in the rest of the paper.

3.1 System Model

Our system model consists of a set of virtual machines $V = \{v_1, v_2, \ldots, v_k\}$ hosted on a shared hypervisor. Note that when $i \in \{1, \ldots, k\}$ can be understood from the context, we simply use v instead of v_i. These VMs might be either well-behaving or attacking. Well-behaving VMs are those that aim at doing their jobs smoothly without having the intention to harm neither the cloud system nor other VMs. On the other hand, attacking VMs seek to harm the cloud system and/or other co-hosted VMs by continuously and collaboratively sending malicious code fragments to form distributed malicious attacks. Such VMs might be either (1) malicious in case their real owners create the attacks or (2) compromised in case the source of attacks is a third party that manipulates VMs and injects its malicious code through them.[1] Each attacking VM is of type $y \in Y$, where Y denotes the set of all attackers' types (e.g., privilege escalation, DoS attackers, etc). Knowing this fact, the cloud system has to find the optimal detection strategy that maximizes the detection of such attacks. To do so, the hypervisor, acting on behalf of the cloud system, has a specific amount of resources R that comprises both the amount R_c of resources to be dedicated to serving clients and the amount R_d of resources to be dedicated for intrusion detection such that $R = R_c + R_d$. Thus, the objective of the hypervisor becomes finding the optimal detection load distribution strategy that maximizes the detection of distributed attacks, while respecting the budget R_d of resources. We model this situation as a repeated Bayesian Stackelberg security game of two players, i.e., hypervisor and attackers. In game theory, a game is said to be Bayesian if some players are unsure about the types, preferences, or payoffs of other players [19]. Our game is Bayesian since the hypervisor is uncertain about the types of attackers that might be targeting the cloud system. Specifically, we consider an attack model

1. In the rest of the paper, the term *attacker* is used to refer to both cases.

in which multiple attackers of different types (e.g., Denial of Service, privilege escalation, etc.) are expected to target the cloud system. The game is played sequentially in the sense that the hypervisor representing the *leader* of the game commits first to a certain detection load distribution strategy and then attackers (*followers* of the game) choose their attack distribution strategy after having observed the hypervisor's strategy implementation. Note that the proposed game is dynamic. Specifically, a game is said to be dynamic if it satisfies one of the two following conditions [20]. The first condition is achieved when the interaction among players is itself inherently dynamic in the sense that the players are able to observe each other's actions prior to making decisions regarding their optimal responses. The second condition is achieved when a one-off game is repeated a number of times, thus allowing players to examine the outcomes of previous games before playing later ones. Our proposed game satisfies both conditions and can hence be considered a dynamic game. In fact, we assume that attackers have the ability to observe the security arrangements adopted by the hypervisor over time to learn which VMs are better protected than others and update their attack strategies accordingly. Moreover, our game is played repeatedly, where at each stage of the game, the probability distributions over the attacker types and the utility of the hypervisor for each of its VMs are (potentially) subject to change (see Section 5 for more details).

In the following, we define the individual utility functions for both hypervisor and attacker. In fact, based on the strategies adopted by both the hypervisor and attacker, a reward is assigned to each of these parties. Particularly, when the hypervisor, facing an attack of type y, selects the pure strategy i (i.e., monitoring v_i) and the attacker selects the pure strategy j (i.e., attacking through v_j), the hypervisor receives a reward of $R^y_{ij}(t)$ and the attacker receives a payoff of $Q^y_{ij}(t)$. The reward function of the hypervisor is defined as follows:

$$R^y_{ij}(t) = \begin{cases} \lambda^h_{v_i}(t) \times w^h_{v_i} - mon_{v_i}, & \text{if } i = j \\ -w^h_{v_j} \times \kappa^y_{v_j} - mon_{v_i}, & \text{if } i \neq j \end{cases}. \quad (1)$$

The first part of this reward function represents the success of the hypervisor in protecting the virtual machine $v_i \in V_i$ of worth $w^h_{v_i}$ (the worth of a VM depends mainly on its price as well as on its hardware, network, and storage configuration) minus the cost mon_{v_i} of monitoring the virtual machine $v_i \in V$. Since the success of detection depends heavily on the IDS's detection probability as mentioned earlier, the reward of the hypervisor at time t is weighed based on its accumulated average detection success rate $\lambda^h_{v_i}(t)$ for all times prior to t (i.e., based on the historical data of the previous detection processes). The second part of the reward function represents the loss of the hypervisor incurred by not monitoring the VM v_j that is being attacked, which is function of the worth of that VM for the hypervisor times the degree of damage $\kappa^y_{v_j}$ caused by the attack of type y on v_j, minus the monitoring cost of the VM v_i that was not targeted by the attack.

Then, the utility of the hypervisor for each VM v_i can be modeled as the discounted sum of the reward function $R^y_{ij}(t)$ across time periods and is depicted as follows:

$$U^y_{ij} = \sum_{t=t_1}^{t_2} \beta^{1/t} \times R^y_{ij}(t). \quad (2)$$

The reason we multiply the reward function $R^y_{ij}(t)$ by $\beta^{1/t}$ is to give more weight to the recent rewards compared to the older ones in the time interval $[t_1, t_2]$.

On the other hand, the attacker's payoff S^y_{ij} would be:

$$S^y_{ij}(t) = \begin{cases} w^a_{v_j} \times \kappa^y_{v_j} - att_{v_j}, & \text{if } i \neq j \\ -w^a_{v_i} \times \lambda^a_{v_i}(t) - att_{v_i}, & \text{if } i = j \end{cases}. \quad (3)$$

This first part of the attacker's payoff function represents the success of the attacker in assaulting through VM v_j that has not been monitored by the hypervisor, which is function of the worth of v_j multiplied by the degree $\kappa^y_{v_j}$ of damage caused by attacking through v_j minus the cost att_{v_j} of preparing the attack on v_j. The second part of the reward function represents the attacker's failure in launching its attack through VM v_j, which is function of the worth of that VM times the probability $\lambda^a_{v_i}(t)$ that the attack through this monitored VM would be actually captured by the hypervisor (based on the attacker's historical observations), minus the cost att_{v_i} of launching the attack through this VM.

Then, the utility of the attacker for each VM v_i can be modeled as the discounted sum of the reward function $Q^y_{ij}(t)$ across time periods and is depicted as follows:

$$Q^y_{ij} = \sum_{t=t_1}^{t_2} \beta^{1/t} \times S^y_{ij}(t). \quad (4)$$

3.2 Overview of Existing Vulnerabilities and Attacks on Virtual Machines

We consider in this work the attacks that occur at the cloud system's virtualization surface which offers attackers with a new appealing security attack vector. Roughly speaking, each functionality provided by the hypervisor (e.g., CPU virtualization, VM management, etc.) can include some vulnerabilities that attackers might exploit to carry out their malicious activities. We discuss in the following some of the attacks that might be exerted against the cloud system w.r.t the corresponding vulnerabilities that might be exploited to carry out such attacks. These attacks have been utilized when performing our experiments as will be explained in Section 6. The list of attacks along with their corresponding vulnerabilities are summarized in Table 1. These attacks have been inspired by the list of cloud-specific vulnerabilities identified in [21] as a recapitulation of some real vulnerabilities collected from the National Institute of Standards and Technology (NIST)'s National Vulnerability Database (NVD) [22], SecurityFocus [23], Red Hat's Bugzilla [24] and Code Vulnerabilities and Exposures (CVEs) [25]. The attacks include co-hosted VMs' memory modification, DoS, virtual machine destruction, virtual machine crash, and privilege escalation. The details about these attacks and how they could be technically carried out can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TDSC.2019.2907946.

3.3 Assumptions

The following assumptions are considered in the rest of the paper:

1) Our solution is designed to model situations in which the hypervisor has a fixed, limited, and known budget

TABLE 1
List of Attacks w.r.t the Associated Vulnerabilities

Attacks	Vulnerabilities
Co-hosted VMs' Memory Modification	Soft Memory Management Unit (MMU)
Denial of Service	I/O and networking Interrupt and timer Paravirtualized I/O VM exits
Virtual Machine Destruction	VM Management
Virtual Machine Crash	Hypercalls
Privileges Escalation	Symmetric Multiprocessing (SMP) Remote Management Software Hypervisor Add-ons

of security resources R_d that it cannot exceed when distributing the detection load over VMs. We thus assume that the hypervisor cannot provide full and permanent detection coverage on all VMs at all times since this will have negative effects on the share of resources dedicated to serving clients and would lead hence to decrease the overall revenue of the cloud providers.

2) We assume that VMs worth differently for the hypervisor and attackers. This assumption is realistic since each of these parties uses different criteria to evaluate the importance of VMs. For example, the hypervisor might be more interested in the prices of the VMs, while the attacker might care more about the sensitivity of the data contained in the VM and/or the potential vulnerabilities that are present on the VM.
3) The utility functions are designed in such a way that makes the hypervisor not allocate any detection load to those VMs, $\mathcal{V} \subseteq V$, whose monitoring cost $mon(v')$ exceeds their worth $w_h(v')$ for the hypervisor, $\forall v' \in \mathcal{V}$. This vision is mathematically implemented in Eq. (1) by designing the hypervisor's utility for a certain VM in such a way to be negative when the monitoring cost of that VM is lower than its worth.
4) We assume that attackers can, over time, get an idea of how the hypervisor is choosing the VMs to put more detection load on. For example, after a certain number of attack attempts on some VMs that have particular configurations/characteristics, the attacker might infer that all VMs having such configurations/characteristics are highly monitored and hard to attack and hence it will move to attacking other VMs. This might be considered as a basic and incremental learning process. Specifically, we do not assume that the attacker is able to immediately learn the hypervisor's strategy. Instead, the attacker observes the previous moves of the hypervisor (as is the case in traditional Stackelberg games). However, the attacker does not solely rely on its immediate previous observation; but instead it capitalizes on the cumulative learning over time (obtained through multiple previous moves) to play its best response to the hypervisor's strategies.

TABLE 3
Attackers' Payoff (DoS)

VM	Worth	Damage	Attack cost
v_1	9	4	3
v_2	11	6	5
v_3	6	1	0.5

3.4 Illustrative Example

The core challenge in designing a Bayesian Stackelberg game is to populate the payoff matrices of both the hypervisor and attackers in a meaningful fashion. To do so, let's first consider a cloud system consisting of three VMs v_1, v_2, and v_3 hosted on top of hypervisor h at time t. In Table 2, we highlight the worth of each of those VMs for the hypervisor, the cost entailed by monitoring each single VM, and the potential damage that might arise from attacking each VM. Similarly, we show in Table 3 the worth of each considered VM for the attacker, evaluate the cost entailed by launching an attack through each of these VMs, and highlight the degree of damage that the attacker might cause through launching attacks through each particular VM. Assume now that the average detection success rate of the hypervisor over the three VMs is 0.7 (i.e., $\lambda^h_{v_1}(t) = \lambda^h_{v_2}(t) = \lambda^h_{v_3}(t) = 0.7$) and that the average attack sucess rate of the attacker over the three VMs is 0.5 (i.e., $\lambda^a_{v_1}(t) = \lambda^a_{v_2}(t) = \lambda^a_{v_3}(t) = 0.5$). Suppose that the cloud system is expected to face DoS attackers. Based on the inputs from Tables 2 and 3 and the possible actions of both players (i.e., hypervisor and attacker), Table 4 shows the payoff values that each of these players would obtain.

In the payoff matrix (Table 4), the row represents the hypervisor and the column represents the attacker. Thus, $R_{ij}(t)$ represents the hypervisor's utility when this hypervisor is monitoring v_i while the attacker is launching its attack through v_j. Similarly, $Q_{ij}(t)$ represents the attacker's utility when this attacker is attacking through v_j while the hypervisor is monitoring v_i. For example, when the hypervisor monitors v_1 and the DoS attacker chooses that same VM v_1 to launch attack through, the hypervisor's payoff would be $R^{DoS}_{11}(t) = 0.7 \times 10 - 3 = 4$ (Eq. (1)) for having successfully protected v_1 and the attackers' payoff would be $Q^{DoS}_{11}(t) = -9 \times 0.5 - 3 = -7.5$ (Eq. (3)). On the other hand, if the hypervisor chooses to monitor v_1 while attackers choose v_2 to attack through, then the hypervisor would undergo a

TABLE 2
Hypervisor's Payoff (DoS)

VM	Worth	Damage	Monitoring cost
v_1	10	-3	3
v_2	14	-1	6
v_3	9	-1	2

TABLE 4
Payoff Matrix (DoS)

VM	v_1	v_2	v_3
v_1	$4, -7.5$	$-33, 61$	$-33, 5.5$
v_2	$-20, 33$	$3.8, -10.5$	$-20, 5.5$
v_3	$-11, 33$	$-11, 61$	$4.3, -3.5$

negative loss of $R_{12}^{DoS}(t) = -9 \times 0.5 - 3 = -7.5$ and the attacker will have a positive payoff of $Q_{12}^{DoS}(t) = 11 \times 6 - 5 = 61$. Having defined the utility matrices, the problem of computing the optimal detection load probability distribution can now be solved using the simplex technique. For space constraints, we are not able to show the details of this technique. However, a detailed stepwise methodology for solving optimization problems using simplex can be found in our previous work published in [1].

4 Adaptive Detection Load Distribution Strategy: Bayesian Stackelberg Game

We formulate in this section the intrusion detection problem as a Stackelberg security game between the cloud system and attackers. Practically, the hypervisor (acting on behalf of the cloud system) plays the role of the game leader and makes the first move by choosing its detection load distribution strategy over VMs, whereas attackers are the followers that observe the leader's strategy (Assumption 4 - Section 3.3) and choose their best responses to it in terms of attack distribution strategies. The backward induction reasoning [26] is employed to determine the optimal strategies of both the cloud system and attackers. This is done by first deriving the best response of the attackers to a (fixed) observed strategy of the cloud system and then integrating this best response to the cloud system's optimization problem to help it select the optimal detection load distribution strategies. Intuitively, this means that the cloud system anticipates that attackers will play their best responses to its (observed) detection load distribution strategy and embeds this knowledge into its optimization problem to select the optimal detection load distribution strategy using this information. Let L and F denote the index sets of the hypervisor (leader) and attacker's (follower) pure strategies, respectively. Let l represent a vector of the hypervisor's pure strategies (a.k.a hypevisor's policy) and f represent a vector of the attacker's pure strategies (a.k.a attacker's policy). Thus, the value l_i would represent the proportion of times in which the hypervisor plays the pure strategy i from its policy set, which means monitoring the VM v_i. Similarly, the value f_j represents the proportion of times in which the attacker plays the pure strategy j from its policy set, which means attacking through VM v_j.

Let us fix first the hypervisor's policy to a certain policy l. After observing l (i.e., the hypervisor's vector of pure strategies over time), the attacker needs to solve the following linear programming optimization problem in order to determine its optimal response to l:

$$\begin{aligned} &\text{maximize} \sum_{j\in F}\sum_{i\in L} Q_{ij} \times f_j \times l_i \\ &\text{subject to} \sum_{j\in F} f_j = 1, \\ &\qquad f_j \in [0,1], \qquad \forall j \in F. \end{aligned} \tag{5}$$

Knowing the fixed strategy l of the leader, the best response $f_j(l)$ of the attacker should yield a non-negative utility to the attacker, which means that Problem (5) has to satisfy the following constraint:

$$f_j \times \sum_{i\in L} Q_{ij} \times l_i \geq 0, \quad \forall j \in F. \tag{6}$$

Moreover, given that $f_j(l)$ is the attacker's best response strategy, any deviation from this strategy (i.e., $1 - f_j$) would lead the attacker to undergo a loss in terms of utility. Thus, Problem (5) has to satisfy the following constraint as well:

$$(1 - f_j) \times \sum_{i\in L} Q_{ij} \times l_i \leq 0, \quad \forall j \in F. \tag{7}$$

In Eq. (7), f_j represents the best pure strategy of the attacker in response to the observed strategy l_i of the hypervisor. Thus, f_j can be either 0 or 1. In case the best response f_j is set to 1 meaning that the attacker chooses to attack, then the deviation from f_j would be $1 - f_j = 1 - 1 = 0$ (do not attack), which means that the utility of the attacker in this case would be always 0 (i.e., $0 \times \sum_{i\in L} Q_{ij} \times l_i$). Intuitively, this means that the attacker would not gain (nor lose) anything since it didn't launch any attack. On the other hand, if the best response of the attacker f_j is set to 0 meaning that the attacker chooses not to attack in response to the hypervisor's observed strategy l_i, then the deviation from f_j would be $1 - f_j = 1 - 0 = 1$, which means that the attacker would chose to attack. Then, the utility of the attacker will be always negative, which represents the fact that the attack is unsuccessful since the attacker will be caught by the hypervisor (as a result of not playing the best response to the observed hypervisor's strategy), along with the cost spent to launch this attack.

Let's move now to the cloud system's side. The hypervisor, knowing that the attacker will play its best response $f_j(l)$ to every hypervisor's strategy l, incorporates this knowledge into its optimization problem to determine the solution l that maximizes its own payoff. Thus, the hypervisor has to solve the following problem:

$$\begin{aligned} &\text{maximize} \sum_{i\in L}\sum_{j\in F} U_{ij} \times f_j(l) \times l_i \\ &\text{subject to} \sum_{i\in L} l_i = 1, \\ &\qquad l_i \in [0,1], \qquad \forall i \in L. \end{aligned} \tag{8}$$

Problem (8) can be completed by incorporating the characterization of $f_j(l)$ depicted in Problem 5 and Eqs. (6) and (7). Taking into account the fact that given any optimal mixed strategy $f_j(l)$, then all the pure strategies in its support are also optimal [27], we can consider only the optimal pure strategies of the attacker (which always exist) and symbolize the optimal pure strategies using binary variables. Moreover, to enhance the decisions of the hypervisor, we incorporate the probability distribution p^y of facing each type $y \in Y$ of attackers into the hypervisor's optimization problem. For example, if the hypervisor learns that the majority of attackers targeting the cloud system in a certain period are DoS attackers, then it would adjust its detection load strategy towards assigning more load to the VMs that are suspected to be vulnerable to such attacks. In Section 5.3, we explain how the hypervisor would be able to practically compute p^y. Knowing all this information, the hypervisor's problem becomes:

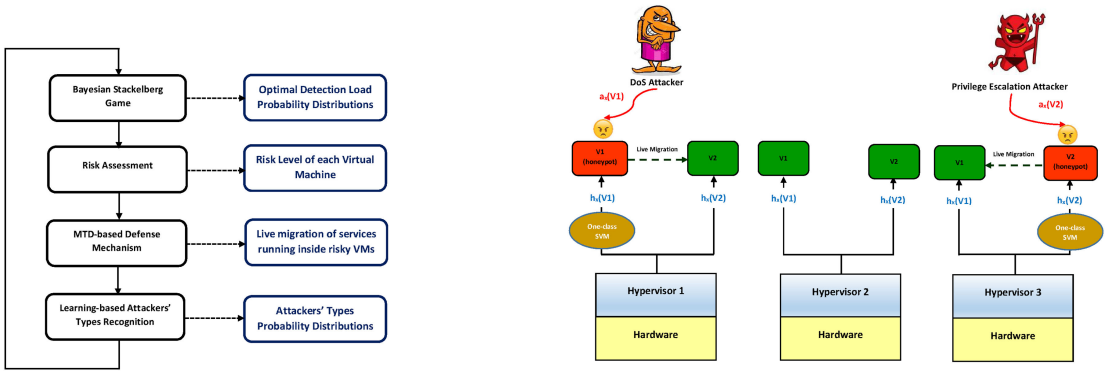


Fig. 1. Repeated Bayesian Stackelberg game phases: Bayesian stackelberg game, risk assessment, services deployment, and Honeypots deployment.

$$
\begin{aligned}
&\underset{l,f}{\text{maximize}} \sum_{y\in Y}\sum_{i\in L}\sum_{j\in F} p^y \times U_{ij}^y \times l_i \times f_j^y \\
&\text{subject to } \sum_{i\in L} l_i = 1, \\
&\quad \sum_{j\in F} f_j^y = 1, && \forall y \in Y \\
&\quad (1 - f_j^y) \times \sum_{i\in L} Q_{ij} \times l_i \leq 0, && \forall j \in F,\ y \in Y \\
&\quad f_j^y \times \sum_{i\in L} Q_{ij} \times l_i \geq 0, && \forall j \in F,\ y \in Y \\
&\quad l_i \in [0,1], && \forall i \in L \\
&\quad f_j^y \in \{0,1\}, && \forall j \in F,\ y \in Y.
\end{aligned} \tag{9}
$$

In Problem (9), the first and fifth constraints compel a feasible mixed policy for the hypervisor, whereas the second and sixth constraints compel a feasible pure strategy for the attacker. The sixth constraint restricts as well the actions' vector of the attacker to be a pure distribution over F. The third and fourth constraints force the best response $f_j(l)$ to be optimal for the attacker in terms of gained utility. Problem (9) is an integer program with a non-convex quadratic objective [27]. Thus, the final step would be converting the Mixed-Integer Quadratic Programming (MIQP) at hand into a Mixed-Integer Linear Programming (MILP) by removing the non-linearity of the objective function. This can be achieved by assigning the value of $l_i \times f_j^y$ to a new variable z_{ij}^y. Thus, the problem becomes:

$$
\begin{aligned}
&\underset{l,f}{\text{maximize}} \sum_{y\in Y}\sum_{i\in L}\sum_{j\in F} p^y \times U_{ij}^y \times z_{ij}^y \\
&\text{subject to } \sum_{i\in L}\sum_{j\in F} z_{ij}^y = 1, && \forall y \in Y \\
&\quad f_j^y \leq \sum_{i\in L} z_{ij}^y \leq 1, && \forall j \in F,\ y \in Y \\
&\quad \sum_{j\in F} f_j^y = 1, && \forall y \in Y \\
&\quad (1 - f_j^y) \times \sum_{i\in L} Q_{ij} \times z_{ij}^y \leq 0 && \forall j \in F,\ y \in Y \\
&\quad f_j^y \times \sum_{i\in L} Q_{ij} \times z_{ij}^y \geq 0 && \forall j \in F,\ y \in Y \\
&\quad z_{ij}^y \in [0,1], && \forall i \in L,\ j \in F,\ y \in Y \\
&\quad f_j^y \in \{0,1\}, && \forall j \in F,\ y \in Y.
\end{aligned} \tag{10}
$$

Having linearized the problem, the MILP in Problem (10) can be now solved using a linear programming solver tool to derive the optimal mixed strategies of the cloud system and attackers [28]. Note also that the probability distributions obtained form the Bayesian Stackelberg game are changed from time to time (according the results of the risk assessment phase), which adds an extra complication layer to the attackers and makes it quite difficult for them to breach the detection strategy of the hypervisor.

5 Learning-Based Detection and Defense System: Repeated Bayesian Stackelberg Game

As depicted in Fig. 1, the repeated Stackelberg game consists of four main phases: Bayesian Stackelberg game, virtual machines' risk assessment, services deployments and defense mechanism, and attackers' types recognition technique. These phases run repeatedly at each time unit t of the discrete time window $[t_1, t_2]$. The Bayesian Stackelberg game (described in Section 4) computes the optimal probability distributions of the hypervisor's detection load over the guest VMs. To evaluate the effectiveness of the detection strategy, the risk assessment phase enables the hypervisor to conduct an in-depth study on the vulnerabilities and threats that might be present on VMs and to analyze their past attack history to derive the appropriate risk level of each VM. Having identified the risky VMs, the goal of the services deployment phase is to advance a defense mechanism that protects services from being successful target for attackers. This is done by offering a live-migration-based decision making framework that allows the hypervisor to migrate services hosted on VMs classified as risky to other safer VMs. Finally, the honeypots deployment phase exploits the idle VMs (running no active services) by deploying honeypots inside them to collect malicious data with the aim of studying and learning the behavior and objectives of the attackers. The collected data is analyzed using a one-class SVM classifier to predict the types of attackers and learn about their probability distributions. This information is used finally to feed the Bayesian Stackelberg game of the next time moment $x + 1$ with the probability distributions over the attackers' types to adjust and optimize the hypervisor's detection load distribution strategies. Note that only the first two phases (Bayesian Stackelberg game and risk assessment) have to be continuously

TABLE 5
Virtual Machine Worth Scale and Description

Worth Level	Value	Description
Important	6	The VM has sophisticated hardware, networking, and storage capabilities.
Medium	3	The VM has intermediate hardware, networking, and storage capabilities.
Moderate	1	The VM has simple hardware, networking, and storage capabilities.

repeated (i.e., every time unit of the discrete time window). Specifically, the execution of the rest of the phases is dependent on the output of the risk assessment phase. In other words, if no VMs are suspected to be risky at a certain time unit, there will be no need to proceed with the other subsequent steps at that time moment. In what follows, we explain each phase of the repeated Stackelberg game in detail and provide numerical examples.

5.1 Virtual Machines Risk Assessment

Having computed the optimal detection load distribution strategy using the one-stage Bayesian Stackelberg game described in Section 4, the hypervisor assesses in this phase the risk level of each VM. The methodology used for risk assessment is inspired mainly by that of the NIST [29], which provides a comprehensive guide on how to evaluate the security risk levels of Information Technology (IT) resources. Specifically, the risk level of a VM v is estimated in terms of the *likelihood* of exploiting a specific *vulnerability* that is present on v to exert some *attack* along with the consequent *impact* of that malicious act on v. Formally, the risk level assessment function of VM v at the time moment $t \in [t_1, t_2]$ is calculated as follows:

$$Risk_v(t) = w_v(t) \times \upsilon_v(t) \times \vartheta_v(t), \tag{11}$$

where $w_v(t)$ is the worth of v at time moment t, $\upsilon_v(t)$ is the magnitude of impact resulting from the exploit of the vulnerabilities present on v at time moment t, and $\vartheta_v(t)$ is the threat likelihood on v at time moment t. Thus, the first step in assessing the risk levels would be estimating the worth of each virtual machine. The worth is an indicator of the degree of damage that could be entailed by the exercise of a certain attack on the VM. Obviously, the worth of a certain VM is decided on the basis of its current hardware, storage, and networking capabilities (e.g., memory, CPU, bandwidth, etc.). Table 5 shows a list of possible worth levels, values, and descriptions that can be used to assess the worths of the VMs.

TABLE 6
Vulnerability Scale and Description

Vulnerability Level	Value	Description
High	6	The exploit of the vulnerability results in extremely painful losses for the VM and cloud system as a whole. Such vulnerabilities can include vCPUs, VM management, SMP, paravirtualized I/O and remote management software.
Medium	3	The exploit of the vulnerability results in painful losses for the VM and cloud system. Such vulnerabilities can include soft MMU.
Low	1	The exploit of the vulnerability results in manageable losses for the VM. Such vulnerabilities can include hypercalls.

TABLE 7
Threat Scale and Description

Threat Level	Value	Description
High	6	The threat is extremely strong and performed by an expert attacker. Such threats can include DoS, and privilege escalation.
Medium	3	The threat is strong and performed by a motivated attacker. Such threats can include co-hosted VMs' memory modification and VM destruction.
Low	1	The threat is weak and performed by a non-professional attacker. Such threats can include virtual machines crash.

The second step in the risk assessment process involves identifying and listing the VM's potential vulnerabilities that attackers might take advantage of to carry out their malicious attacks. In our case, we use the list of vulnerabilities identified in Table 1. Table 6 shows a list of possible vulnerability levels, values, and descriptions that can be used to assess the impacts of vulnerability exploitations on the VMs.

Having characterized the potential vulnerability exploitation impacts, the third step is to determine the corresponding threats that exploit the identified vulnerabilities to launch attacks against VMs. For our risk assessment process, we restrict the analysis to the list of attacks identified in Table 1. Table 7 shows a list of some possible threats levels, values, and descriptions that can be used to assess the threat likelihood on the VMs.

Now that we have defined the worth, vulnerability, and threat levels, we need to proceed with identifying the risk levels scale to be used as a reference when deciding about the VMs' risk levels. The risk levels scales and descriptions are presented in Table 8.

We are now well-equipped to move forward with the risk levels determination step, where the risk level of each VM is computed using Eq. (11) after normalization. We give in Table 9 a numerical example that clarifies how to compute and determine the risk levels of three VMs based on the worth, vulnerability, threat, and risk scales defined in Tables 5, 6, 7, and 8 respectively.

In Table 9, v_1 has to be classified as being low-risk (according to Table 8), v_2 as moderately risky, and v_3 as highly risky. Note that we multiply by 6 and divide by 216 (i.e., $6 \times 6 \times 6$) in Table 9 to normalize the computed risk level values [29].

Nonetheless, our risk assessment process is not yet complete. In fact, despite its importance and effectiveness, the

TABLE 8
Risk Scale and Description

Risk Level	Risk Scale	Description
High	5-6	There is an urgent need to implement corrective measures (e.g., live migration) to resume the normal operation of the cloud system.
Medium	3-4	There is a need to implement corrective measures within a reasonable period of time to resume the normal operation of the cloud system.
Low	1-2	The risk does not constitute an obstacle to the normal operation of the cloud system.

TABLE 9
Risk Levels Determination Example

VM	Worth	Vulnerability Impact	Threat Likelihood	Risk
v_1	6	1	1	$\frac{6\times1\times1}{216}\times 6 = 0.17$
v_2	6	3	6	$\frac{6\times3\times6}{216}\times 6 = 3$
v_3	6	6	6	$\frac{6\times6\times6}{216}\times 6 = 6$

above presented risk assessment is generic for all VMs and does not take into account the past history of each VM. Practically, to make the risk analysis more realistic and thorough, we have to consider the past attack history of the VMs in our analysis. Therefore, we propose to integrate the attack growth/decay (growth in case the number of attacks is increasing and decay otherwise) factor $e^{t.k_v(t)}$ of each VM v at time moment t into our risk level determination formula. Thus, the risk assessment formula presented initially in Eq. (11) becomes:

$$Risk_v(t) = w_v(t) \times \upsilon_v(t) \times \vartheta_v(t) \times e^{[(t-1)-(t-2)].k_v(t)}, \tag{12}$$

$$k_v(t) = \frac{\ln(\frac{N_v(t-2)}{N_v(t-1)})}{[(t-2)-(t-1)]}, \tag{13}$$

where $k_v(t)$ is the attack growth/decay rate on v and $N_v(t)$ is the number of times v has been attacked at time moment t. As depicted in Eq. (13), the attack growth/decay rate $k_v(t)$ is computed based on the difference between the number of attacks that existed at the two past consecutive time moments $t-1$ and $t-2$. Assume that the risk calculations presented in Table 9 were derived at time moment $t=3$ and that v_1 got attacked three times at time moment $t=1$ and five times at time moment $t=2$. We explain in the following how Eq. (13) has been derived and how to practically compute the attack growth factor $k_{v_1}(t)$ of v_1 at time moment $t=3$. Specifically, we have: $5 = 3 \times e^{(2-1).k_{v_1}(3)} \Rightarrow 5/3 = e^{k_{v_1}(3)} \Rightarrow \ln(5/3) = \ln(e^{k_{v_1}(3)}) \Rightarrow \ln(5/3) = k_{v_1}(3) \Rightarrow k_{v_1}(3) = \ln(5/3) = 0.511$. Thus, the risk level of v_1 would be updated to become $R_{v_1}(3) = 0.17 \times e^{0.511\times1} = 0.283$, where v_1 remains a low-risk VM.

5.2 MTD-Based Defense Mechanism

In the light of the results obtained from the risk assessment phase, we discuss in this section an MTD-based services' deployment strategy whose goal is to provide a defense mechanism to protect the services hosted in the cloud system from being successful targets for attackers. Practically, we propose a security-oriented live migration strategy [30] that allows the hypervisor to migrate the services running inside VMs classified as risky to be hosted in other more secure VMs. To do so, the hypervisor has to identify first the set of VMs that might serve as replacements for the risky ones. Apart from security considerations, determining such a set of VMs involves some technical constraints, where the migration process should maintain some technical compatibilities between the migration source and destination VMs. For example, the Operating systems (OSs) of the source and destination VMs have to be consistent since migration between distinct OSs (e.g., Windows and Linux) might entail some technical complications and unanticipated technological roadblocks. Moving to the security perspective, the set of VMs that are eligible to serve as replacements should evidently be selected to be non-risky based on the risk assessment's results.

Formally, let $E_v(t)$ denote the set of VMs that are eligible to replace a VM v at time moment t. These VMs satisfy thus the aforementioned technical constraints and are classified as low-risk in the risk assessment phase. Also, let $p_{v\longrightarrow v'}(t)$ denote the percentage of worth increase between v and v' at time moment t, which is calculated as per

$$p_{v\longrightarrow v'}(t) = \begin{cases} \frac{w'_v(t)-w_v(t)}{w_v(t)}, & \text{if } \frac{w_v(t)-w_v(t)}{w_v(t)} \geq 0 \\ +\infty, & \text{otherwise} \end{cases}. \tag{14}$$

Let v^* be the VM that gives the minimum worth increase percentage w.r.t v, i.e., $p_{v\longrightarrow v^*}(t) = \min(p_{v\longrightarrow v_m}(t)),\ \forall v_m \in E_v(t)$. The decision of the hypervisor to migrate a service running inside v to another VM $v' \in E_v(t)$ is taken as follows:

- if $E_v(t) \neq \emptyset$ and $p_{v\longrightarrow v^*}(t) \neq +\infty$, the hypervisor selects the VM v^* that gives the least percentage increase in the worth value $p_{v\longrightarrow v^*}(t)$ compared to v.
- if $E_v(t) = \emptyset$ or $p_{v\longrightarrow v^*}(t) = +\infty$, then the hypervisor creates a new VM v'' to be the migration destination for the services running in v.

The idea behind selecting the VM giving the least worth increase percentage to serve as a replacement is to guarantee that the migrated service will be running in a new environment that is very similar to that it was running inside (before migration) in terms of VM's actual storage, CPU, and memory states. This is because the worth is an indicator of the current hardware, storage, and networking capabilities of the VM. Such a migration decision would help maintain the performance of the service after the migration process. Along with the same line, we exclude the VMs that give a negative worth percentage increase (by assigning them $+\infty$ in Eq. (14) so that they will never be selected as minimum) because we do not want the migrated service to run in an environment that does not satisfy its actual performance needs. On the other hand, if no VMs satisfying the technical and security constraints of the migration source v_i are available (i.e., $E_{v_i}(t) = \emptyset$) or no VMs having non-negative worth increase percentage compared to v_i exist (i.e., $p_{v_i\longrightarrow v_j^*}(t) = +\infty$), then the hypervisor creates a new VM and migrates the services running inside the risky VM to it.

Note that it would be predictable for attackers to guess that the services running in a certain VM would be migrated to another VM having similar hardware configuration as designed in Eq. (14). However, this is true only if the attackers are made aware that a migration decision is to be taken. Therefore, the idea of the proposed MTD technique is to make attackers unaware that the services running inside (risky) VMs are being migrated. This is achieved through the honeypot deployment technique discussed in the next subsection which keeps a copy of the migrated services running in the honeypot VMs, while using fake worthless data to populate these services. This would give attackers the impression that honeypot VMs are still running real services and that no migration is being carried out.

5.3 Honeypot Deployment and Machine Learning

In order to learn the probability distributions over the attackers' types and inspect their objectives, the hypervisor

can exploit the *idle* VMs by deploying honeypots inside them to serve as traps for attackers. A honeypot in our case is a deception VM that is configured by the hypervisor to serve as a purposed target for attacks. The objective is to give attackers the impression that they are interacting with a real system and hence encourage them to freely launch their attacks in order to gather massive and valuable information. In this way, any connection with the honeypot would be deemed to be an attack and all the traffic circulating to the honeypot is roughly entirely unauthorized. Honeypot systems can be either of low-interaction or high-interaction [3]. Low-interaction honeypots (e.g., Honeyd) function by emulating services designed to catch some specific malicious activities (e.g., FTP login), which makes them limited to a confined level of interaction with attackers. The main advantages of low-interaction honeypots lie in their simplicity to deploy and maintain and in the minimal risk that they entail to the system. Practically, low-interaction honeypots do not allow attackers to have access to the OS, which protects the cloud system and co-hosted VMs from potential attacks. Nonetheless, the main disadvantages of low-interaction honeypots are the limited amount of information that they can capture and their simple configuration that increases the capability of skillful attackers to detect their presence.

On the other hand, high-interaction honeypots (e.g., Honeynets) consist of real applications and OSs that are designed for advanced research purposes. Simply speaking, high-interaction honeypots involve providing a real execution environment (e.g., a real Windows honeypot system running a real FTP server) in which nothing is being emulated. The main advantage of such honeypots is the ability to gather large amounts of information that enable analyzing and understanding the complete extent of the attackers' malicious behavior. Moreover, the fact that high-interaction honeypots rely on real systems makes them appealing to attackers and hard to be recognized as being traps. However, the main self-evident disadvantage of such a type of honeypots lies in the risks that they might impose on the real system. Therefore, a thoughtful implementation and configuration of high-interaction honeypots is required to block attackers from exploiting these honeypots to hurt other non-honeypot systems. Such a thoughtful implementation might include, for example, isolating the CPU assigned to honeypots from that assigned to non-honeypot VMs to prevent scheduling tasks coming from honeypots on the same physical CPU as other non-honeypot VMs.

Because the aim of our honeypots deployment process is to study the behavior of the attackers to be able to determine the probability distributions over their types, we choose to employ high-interaction honeypots for our problem. The fact that high-level interaction honeypots make no predefined assumptions on how attackers shall misbehave makes them able to capture all types of malicious activities including unexpected misbehavior. Thus, they are suitable to study and analyze different types of attacks including unknown ones. Furthermore, in order to make honeypots even more appealing for attackers, our honeypot deployment approach consists of keeping a copy of the (migrated) services running inside honeypot VMs, while using fake data to populate them. For example, a banking system that migrates to another safer VM will keep running inside the honeypot VM, while using dummy accounts numbers, clients' names, etc. Along with the same line, the services running inside honeypot VMs are changed and updated on a regular basis to minimize the chances of being discovered by attackers as traps.

Having collected the necessary data from honeypots, we need a classification technique to analyze this data and learn the probability distributions over the attackers' types. To this end, we choose to employ the one-class Support Vector Machine (SVM) [4] which has been proposed as an extension of the traditional SVM binary classifier. One-class SVMs try to find the decision boundary (i.e., hyperplane) which separates the majority of the data points from the origin. In this way, the data points that lie on the other side of the decision boundary will be deemed to be *outliers* or *abnormal activity*. This enables the decision function to classify any new data as being analogous or different from a certain pattern of data fed in the training phase (i.e., novelty detection). The selection of one-class SVM to be used in our problem stems from three main observations. First, one-class SVM is an unsupervised classification technique which requires no extensive prior information nor predefined class labels for the analyzed data. Second, one-class SVM supports multi-class data classification, which makes it appropriate for our problem in which we deal with attackers of multiple types. Third, the fact that one-class SVM is dedicated to novelty detection makes it well-suited to identify new types of (yet) undetected attacks. This might be achieved by considering each type of already identified attacks as a *normal activity* and determining the degree of similarity/dissimilarity of each set of new data w.r.t that normal activity data. Suppose, for example, that the classification system has already identified DoS and privilege escalation attacks. If new features that do not match neither DoS nor privileges escalation attacks' features are found on the honeypot system, then this would be considered as a new attack type targeting the cloud system.

Formally, let $x = (x_1, x_2, \ldots, x_n)$ denote the feature vector which contains all the attack features (e.g., source and destination IP addresses, host names, protocol used, geographical information of the attack sources, etc.) collected by the honeypot system. The one-class SVM classification problem is mapped into solving the following objective function's minimization problem:

$$\begin{cases} \min\limits_{\omega,\xi_i,\rho} \ \dfrac{1}{2}\|w\|^2 + \dfrac{1}{\upsilon n}\sum\limits_{i=1}^{n}\xi_n - \rho \\ \text{subject to:} \\ (w.\phi(x_i)) \geq \rho - \xi_i \qquad \forall i = 1, \ldots, n \\ \xi_i \geq 0 \qquad \forall i = 1, \ldots, n \end{cases}, \tag{15}$$

where n is the size of the training set, w represents the normal vector to the hyperplane, and ρ is the bias term. Moreover, $\phi(.)$ denotes a transformation function concretized by the kernel function to project the data into a higher dimensional space and $\xi_i \in \xi_n$ are slack variables used to allow some data points to lie within the margin so as to prevent the SVM classifier from over-fitting with noisy data. Yet more importantly, υ is the regularisation parameter that determines the shape of the solution by specifying (1) an upper bound on the fraction of outliers; and (2) a lower bound on the number of training tuples employed as support vectors. Thus, an increased value

of υ widens the soft margin and augments the probability that the training data will fall outside the normal borders. Problem (15) can be solved using the Lagrange multipliers method so that the decision function $f(x)$ becomes:

$$f(x) = sgn((w.\phi(x_i)) - \rho) = sgn\left(\sum_{i=1}^{N} \alpha_i k(x, x_i) - \rho\right), \tag{16}$$

where $k(x, x_i)$ is the kernel function, which might be either linear, polynomial, gaussian, or sigmoid, i.e.,

$$K(x_j, x_i) = \begin{cases} x_i.x_j, linear \\ (\gamma.x_i.x_j + c)^d, polynomial \\ \exp(-\gamma.\mid x_i - x_j \mid^2), gaussian\ radial\ basis \\ \tanh(\gamma.x_i.x_j + c), sigmoid \end{cases}. \tag{17}$$

Based on the results obtained from the classification process, the hypervisor computes the probability p^y for each attacker's type $y \in Y$ using

$$p^y = \frac{\text{Number of observations classified as ``}y\text{''}}{\text{Total number of observations}}. \tag{18}$$

Finally, this information is used back to feed the Bayesian Stackelberg game (Section 4) with the attackers' types probability distributions to help it continuously adjust and optimize the detection load probability distributions over the set of guest VMs.

6 Experimental Results and Analysis

We explain in this section the environment employed to perform our experiments and present and analyze the experimental results.

6.1 Experimental Setup

To carry out the experiments, we build our own cloud datacenter using CloudSim [31], a cloud simulator that provides realistic cloud features such as co-hosted VMs, network connections among cloud components, and services migration support. The decision to create our own cloud rather than using rented resources from existing providers stems from two observations [1]. In the first place, most of the cloud providers (e.g., Amazon EC2) have strict restrictions concerning any security testing on their resources and infrastructure. In the second place, cloud providers forbid any direct access of the users to the VMs' host system; thus making the acquisition of performance data and the implementation of new algorithms at the host's level far difficult to achieve. The characteristics of the created cloud are populated from the Amazon EC2 X-large instances [5] in terms of VMs configurations and pricing scheme. Specifically, the cloud datacenter is equipped with 100 physical machines; each hosting a number of VMs varying from 10 to 50. The image size of the VMs is of 10000 MB, the memory RAM capacity is of 16 GB, and the hard drive storage is of 976.5625 GB. Each VM is supplied with a 5-core CPU of 1000 Millions of Instructions Per Second (MIPS) each. The network bandwidth share of each VM is 50000 Kbit/s. Moreover, Linux has been adopted as an OS in the datacenter, x86 as a system architecture, and Xen as a Virtual Machine Monitor (VMM). The prices of the VMs, used to compute the utility functions, have been selected according to Amazon EC2 pricing scheme.[2]

TABLE 10
Attacks Occurrence Distributions on Xen Hypervisors

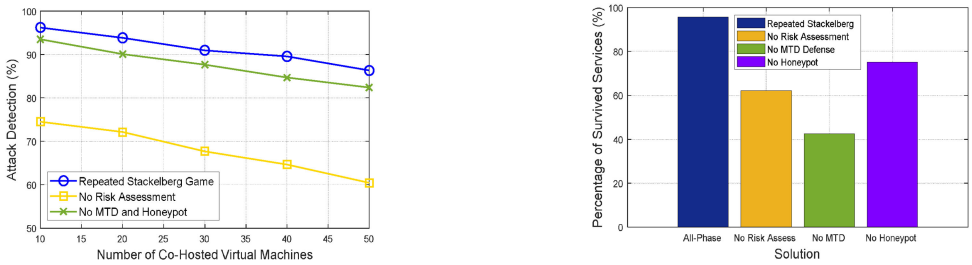
Attack	Occurrence on Xen Hypervisors (%)	Attack Detection (%)
Co-hosted VMs' Memory Modification	8.5%	92.4468%
Denial of Service	45.8%	91.5871%
Virtual Machine Destruction	13.6%	88.0113%
Virtual Machine Crash	5.1%	86.6557%
Privileges Escalation	27%	89.6290%

To analyze the performance of the attackers' types recognition phase, we use a dataset [6] from the Data Driven Security (DDS) datasets collection. The dataset is collected from AWS honeypots deployed on several instances across the world for a period covering March to September 2013 [32]. The dataset records attack data including source and destination IP addresses, host names, protocol used (e.g., TCP), source and destination ports, and geographical information of the attack sources (i.e., country, postal code, longitude, and latitude). To create the training and test sets, we use the k-fold cross-validation technique (with $k = 10$) whereby the dataset is split into k subsets, each used every time as test set and the remaining $k - 1$ subsets are combined together to form the training set. The principal advantage of the k-fold cross-validation lies in its ability to diminish the bias of the classification results on the way based on which data is being divided since each data tuple will be part of the test set exactly once and part of the training set $k - 1$ times.

Finally, to populate the probability distributions over the attackers' types (used to achieve the Bayesian property of the game), we capitalize on the findings presented in [21], which surveys the attacks/vulnerabilities distributions on Xen hypervisors (used in our simulations) based on real data collected from NVD [22], SecurityFocus [23], Red Hat's Bugzilla [24] and CVEs [25]. These probability distributions are summarized in Table 10. Note that all the experiments have been conducted in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-4790 CPU 3.60 GHz Processor and 16 GB RAM.

To show the improvements brought by our solution compared to the state-of-the-art, we compare our work experimentally with five other detection and defense strategies, namely Collabra [12], probabilistic migration [2], one-stage Stackelberg [18], maxmin [1], and fair allocation [33]. The core idea of Collabra [12] is to analyze every hyper-call initiated by each guest application to recognize distributed attacks that aim to compromise the host hypervisor. In the fair allocation model, the detection load is distributed in an equal manner among VMs so as to guarantee the fairness of the detection process. On the other hand, the maxmin-based detection load distribution strategy leverages a maxmin game whose utility functions are mainly fed by the trust scores computed by the hypervisor toward its guest VMs. Although the maxmin-based strategy accounts for the attackers' strategies and resources constraints in the design of the problem (as is the case in our solution); it does not

2. http://aws.amazon.com/ec2/pricing/



(a) Attack Detection (b) Survived Services

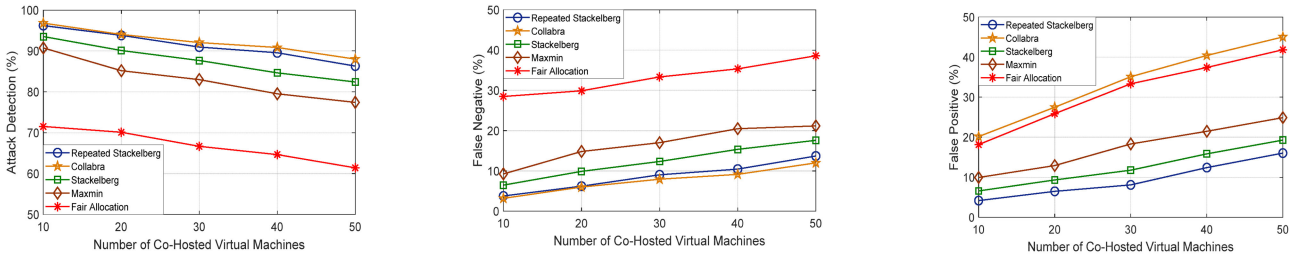
Fig. 2. The objective of this Fig. is to asses the impact of each phase of the repeated Bayesian Stackelberg game on the overall solution's performance.

account however for the fact that attackers have the ability to monitor the cloud system's strategies and adjust their own strategies. Similar to our solution, the one-stage Stackelberg accounts for this challenge by computing the best responses of the attackers to the hypervisor's detection load distribution strategies and incorporating this knowledge into the hypervisor's optimization problem. Different from our solution, the one-stage Stackelberg model abstracts on the types of attackers and is not able hence to provide the hypervisor with real-time learning about the actual types and objectives of the attackers. Our work overcomes this limitation by collecting and analyzing malicious data to learn the probability distributions over the types of attackers targeting the cloud system and incorporating this knowledge into the hypervisor's optimization problem to optimize its decisions. Moreover, our solution offers a proactive defense mechanism that protects services from being successful targets for attackers and works in a repeated fashion to provide incremental and continuous learning for the cloud system. Finally, the probabilistic migration defense strategy [2] relies on the idea of migrating VMs at a certain moment of time based on the probability according to which those VMs are expected to be compromised by attackers during the next time moment, where such a probability is mainly dependent on the attacks' growth success probability over time.

6.2 Experimental Results

In the first set of experiments (Fig. 2), we test different combinations of the proposed repeated Bayesian Stackelberg game experimentally to verify the importance of each phase when used alone and when combined with the other phases. In Fig. 2a, we measure the attack detection performance while comparing the cases where (1) all the phases are integrated into the solution, (2) the risk assessment phase is removed from the solution, and (3) the MTD defense and honeypot and machine learning phases are removed from the solution. By looking at Fig. 2a, we can observe that removing the risk assessment component results in a considerable degradation in the detection performance by 20 percent. This decrease can be justified by the fact that without the risk assessment strategy, the migration of the services according to the MTD mechanism would be done in an arbitrary fashion lack of any knowledge of the risk levels of the VMs. This, in turn, leads to mostly uninformative data collected by the honeypots and analyzed by the machine learning technique. Consequently, the quality of the decisions generated by the Bayesian Stackelberg game fed by such data would be decreased. On the other hand, removing the MTD and honeypot phases from the solution results in a less significant decrease in the detection performance (compared to removing the risk assessment phase) as shown in Fig. 2a. The reason is that by removing the MTD and honeypot phases, no data can be collected and analyzed at all (which is better than having misleading data). Thus, the performance of the solution without these two phases converges to that of a one-shot Stackelberg game which includes no learning component with regards to the attackers types distributions. Overall, we can conclude that the risk assessment phase is the most important phase to optimize the detection performance. On the other hand, the MTD and honeypot phases have the less impact on the detection performance.

In Fig. 2b, we assess the performance in terms of percentage of survived services, while considering the cases where (1) all the phases are integrated into the solution, (2) the risk assessment phase is removed from the solution, (3) the MTD defense phase is removed from the solution, and (4) the honeypot and machine learning phase is removed from the solution. For this experiment, we were able to separate the MTD phase from the honeypot and machine learning phase since the MTD-based migration strategy would influence the percentage of survived services even when used separately from the honeypot and machine learning phase, as opposed to the case of attack detection (Fig. 2a) in which when no migration occurs then no data at all can be analyzed by the machine learning technique. By examining Fig. 2b, we notice that removing the MTD phase (case 3) leads to the poorest performance compared to the other cases. This result is expected since removing the MTD phase leads to taking out the proactive (migration) step from the solution and giving attackers the chance to launch their attacks. On the other hand, keeping the MTD phase (along with the risk assessment phase) and removing the honeypot and machine learning phase (case 4) results in a less painful decrease in the percentage of survived services, where the performance of this combination converges to the performance of a one-shot Stackelberg game. Finally, removing the risk assessment phase and keeping the MTD and honeypot phases (case 2) would lead to arbitrary migration decisions and hence decrease in the effectiveness of the MTD technique. Thus, we can conclude that the MTD phase is the most important phase to increase the percentage of survived services. On the other hand, the honeypot and machine learning phase has the least significant impact on this metric. Overall, we can conclude that the honeypot and machine learning phase, whose removal has the least impact on both the detection performance and number of



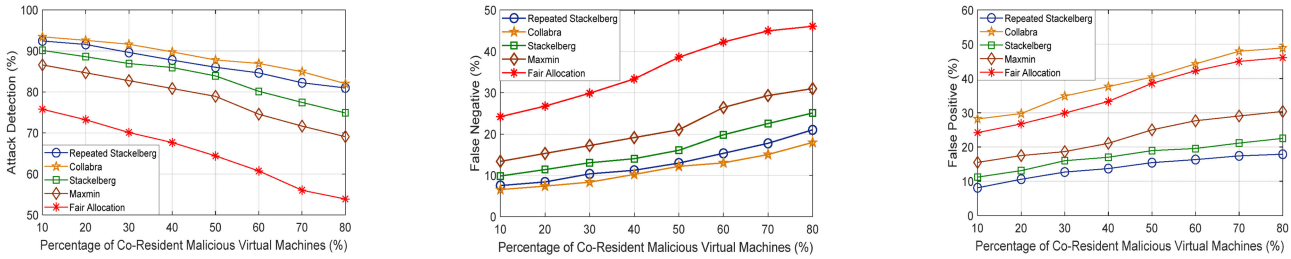
(a) Attack Detection (b) False Negative (c) False Positive

Fig. 3. Our solution improves the detection performance and is scalable to the increase in the number of co-hosted VMs compared to the one-stage Stackelberg, maxmin, and fair allocation strategies.

survived services, is rather an optimization phase (that can be helpful to obtain better decisions) than a mandatory phase.

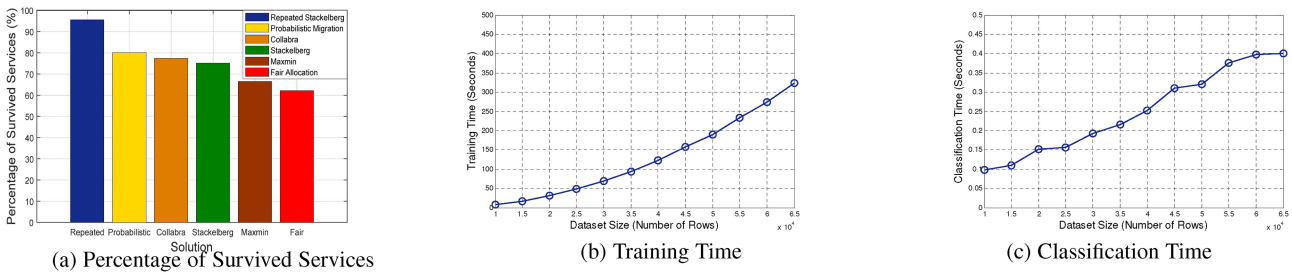
Next, we investigate in Fig. 3 the detection performance metrics (attack detection, false negative, and false positive percentages) of the four studied solutions. Attack detection represents the percentage of attacks that the IDS was able to identify as such. Different from the literature's definition of false negative (i.e., the case where the IDS classifies some activity as benignant when the activity is an actual attack), we consider in our case that the false negative situation occurs when there exists an actual attack targeting the VM but there is no monitoring effort put on that VM to protect it. Different from the literature's usual definition of false positive (i.e., the case where an alarm is raised when there is no actual attack on a particular resource), we consider in our case that the false positive situation occurs when there is some monitoring effort put on a certain VM while this VM is not being attacked. This metric is of special importance since it gives a hint on the amount of security resources wasted during the detection process. By examining Fig. 3, we can notice that the performance of all the studied solutions begins to decrease with the increase in the number of co-hosted VMs. The reason is that increasing the number of VMs on a single physical machine increases the attack space for attackers by giving them an increased number of VMs to distribute their attacks over. Moreover, the increase in the number of co-hosted VMs would lead to reduce the effectiveness of the security budget since the same budget would need to be distributed across a larger number of VMs. Thus, the share of security resources for each single VM is naturally reduced as the number of VMs grows up. However, we can notice from Fig. 3 that our solution and Collabra remain far more resilient to an increased number of co-hosted VMs than the other solutions. The second observation that can be made from Fig. 3 is that our repeated Bayesian Stackelberg, Collabra, maxmin, and one-stage Stackelberg models achieve better detection performance (in terms of attack detection, false negative, and false positive) compared to the fair allocation strategy. The reason is that the repeated Bayesian Stackelberg, one-stage Stackelberg, and maxmin models consider the attackers' strategies in the formulation of the game, which enables them to compute the optimal detection load distributions that best synchronize with the attackers' strategies. On the other hand, the fair allocation model seeks to achieve the fairness in the detection process by distributing the detection load in an equal manner among VMs; thus overlooking how attackers' are distributing their attacks. For example, a fair allocation model which distributes the detection load amongst three VMs v_1, v_2, and v_3 so that each one receives 33.33 percent might end up assigning a big part of the security resources (i.e., 33 percent) monitoring a VM that will not be selected by attackers to contribute in the attacks. Moreover, the Stackelberg-based solutions (i.e., our solution and the one-stage Stackelberg) outperform the maxmin-based solution since the former models account for the fact that attackers have the ability to monitor the hypervisor's detection load distribution strategies and they integrate this knowledge into the hypervisor's optimization problem to optimize its detection strategies. Our repeated Bayesian Stackelberg solution, in its turn, performs better than the one-stage Stackelberg because it includes a learning component that learns the types and objectives of the attackers and incorporates this knowledge into the hypervisor's optimization problem. This increases the awareness of the hypervisor about the nature and gravity of the attacks that are expected to be launched on every VM and aids it hence to adjust the detection load distributions accordingly. Finally, our repeated Bayesian Stackelberg game and Collabra achieve very close detection performance results in terms of attack detection and false negative percentages since Collabra monitors every activity of the VMs, which allows it to achieve high detection performance that is similar to that of our solution. However, unlike our solution in which the false positive percentage is negligible, Collabra entails high percentages of false positives up to 45 percent (Figs. 3c and 4c), thus causing a significant wastage of resources. The reason is that Collabra monitors all of the VMs' activities whether or not these VMs are launching attacks, which leads to large and unnecessary squandering of resources since usually most of the times the VMs are not supposed to launch attacks. It is worth mentioning that we characterize the attack detection metric in a more detailed way in Table 10 by showing the detection rate specific to each of the six considered attacks, where the results depicted in Fig. 3a represent the average detection over all attack types.

In Fig. 4, we study the scalability of our solution with respect to the variation in the percentage of co-resident malicious VMs. To do so, we vary the percentage of attacking VMs co-residing on a single cloud system from 10 percent up to 80 percent to explore the effects of this variation on the performance of the studied solutions. As shown in Fig. 4, the performance of all the solutions begins to decrease with the increase in the percentage of attacking VMs. This unsurprising result is due to the fact that the bigger the number of VMs attacking the system is, the less is the ability of the cloud system to capture attacks under the limited budget of security resources.



(a) Attack Detection (b) False Negative (c) False Positive

Fig. 4. Our solution improves the detection performance and is scalable to the increase in the percentage of co-resident malicious VMs compared to the one-stage Stackelberg, maxmin, and fair allocation strategies.



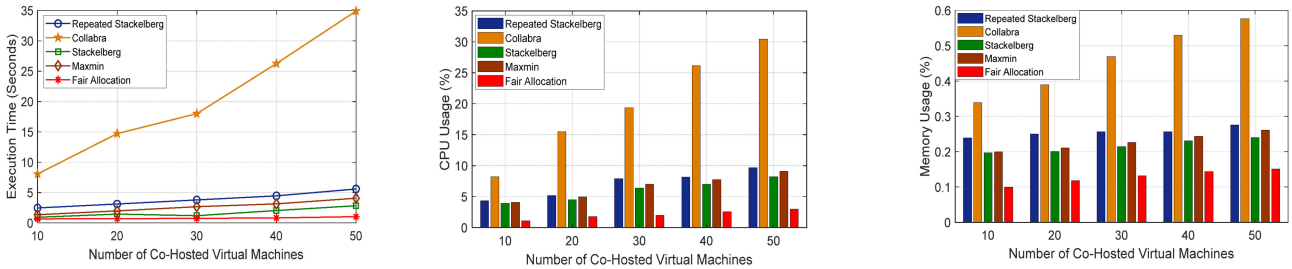
(a) Percentage of Survived Services (b) Training Time (c) Classification Time

Fig. 5. Our solution maximizes the percentage of survived services and entails acceptable training and classification times.

Thus, a possible choice for the cloud system's administrators would be to increase the security budget to face an increased number of attacks. Specifically, our attackers' types recognition phase can aid the cloud's administrators in deciding whether there is a need to increase the security resources budget or not by giving them detailed information about the volume and nature of attacks targeting the cloud system. Fortunately, our solution (along with Collabra) shows a better scalability to an increased percentage of attacking VMs compared to the other models even in extreme cases (i.e., 80 percent of co-resident malicious VMs) thanks to the previously discussed advantages brought by our solution.

In Fig. 5, we study the effectiveness and efficiency of our MTD-based defense mechanism and machine learning technique by measuring the percentage of survived services, and the training and classification times. In Fig. 5a, we measure the percentage of survived services which represents the percentage of services that remained unattacked during their whole lifetime. In this Fig., the probabilistic migration [2] (described in Section 6.1) is added to the comparisons. We notice from Fig. 5a that our solution is able to increase the number of survived services compared to the other solutions. This is thanks to the proactive defense mechanism that our solution advances and that migrates the services running inside risky VMs to other more secure VMs to protect them from being successful targets for attacks. The absence of such a mechanism in the Collabra, one-stage Stackelberg, maxmin, and fair allocation solutions limits their effectiveness to some reactive measures (i.e., detection) and hence leads to an increased number of attacked services. Besides, our work outperforms the probabilistic migration defense strategy [2] since we provide a comprehensive risk assessment framework which takes into consideration not only the attack growth success probability (considered in [2]), but also the potential vulnerabilities of the VMs, their expected threats, as well as their past attack history when deciding on whether to migrate services or not. Fig. 5b shows the time required to train the one-class SVM on various training datasets sizes. To do so, we employ the DDS honeypot data collected from AWSs and whose original size amounts to 650,000 rows. To study the impact of the training dataset's size on the training time, we vary the size of the data from 10,000 to 650,000. Unsurprisingly, Fig. 5b reveals that the training time increases with the increase in the size of the training dataset and reaches at the extreme case (i.e., 650,000 rows) $330s$. The main time complexity lies in the process of constructing an SVM model for each class label. Practically, since we have 6 types of attackers (Table 1) serving as class labels for the training dataset, we have to build one SVM model for each single class and train it to differentiate the samples of that class from the samples of all remaining classes (i.e., novelty detection). We argue that the obtained time is insignificant, especially since this phase is executed offline and not required to be repeated at each time moment as discussed in Section 5. Having completed the training process, the next step is to execute the actual classification part, which consists of assigning an attack type for each particular sample. Since the classification time is also dependent on the dataset's size as is the case for the training time, we test the classification's time on different dataset sizes. It can be noticed from Fig. 5c that the classification time is negligible in all the considered cases, where it does take $0.4s$ to classify samples in a dataset consisting of 650,000 rows.

Finally, we study in Fig. 6 the execution time, CPU utilization, and memory utilization of the considered solutions. By examining Fig. 6a, we notice that the fair allocation approach yields the fastest performance. This is because the detection load is to be distributed equally across VMs, which removes the time complexity of finding the optimal detection load probability distributions. On the other hand, Collabra gives the (largely) slowest performance and the poorest scalability to the increase in the number of VMs



(a) Execution Time (b) CPU Utilization (c) Memory Utilization

Fig. 6. Our solution is efficient in terms of execution time, CPU utilization, and memory utilization compared to Collabra, one-stage Stackelberg, maxmin, and fair allocation strategies

owing to the fact that it requires analyzing all of the VMs' activities. The one-Stackelberg performs faster than the maxmin-based and our repeated Bayesian Stackelberg game. The time difference between the one-stage Stackelberg and maxmin models may be thought of as the time needed by the latter to gather objective and subjective sources of trust and compute the final trust values prior to executing the maxmin game. On the other hand, the time difference between our repeated Stackelberg model and the one-stage Stackelberg and maxmin-based models lies in the time taken by our solution to perform the VMs' risk assessment as well as the computational time entailed by the integration of the attackers' type into the optimization problem. Though, our solution still performs in an efficient manner, where it takes $\approx 5.6s$ to run in a cloud system consisting of 50 co-hosted VMs. We can also notice that the time complexity of our solution grows polynomially with the increase in the number of VMs, which boosts its scalability in large-scale datacenters. We compare in Figs. 6b and 6c the resource consumption entailed by the different considered solutions. We can notice from Figs. 6b and 6c that the fair allocation model records the least CPU and memory utilization since it involves no heavy computation and storage duties. Moreover, our solution, the one-stage Stackelberg, and maxmin models can considerably decrease the CPU consumption by $\approx 15\%$ (Fig. 6b) and memory consumption by $\approx 25\%$ (Fig. 6c) compared to Collabra. The reason is that in Collabra, the hypervisor has to check every incoming call, store it in the memory, and analyze it in order to detect intrusions, which entails high computational and storage overhead. In contrary, in the other three solutions, the detection is done in a selective manner so that the hypervisor doesn't have to monitor all the activities of the VMs. Another important remark to be drawn from Figs. 6b and 6c is that our solution, the one-stage Stackelberg, maxmin, and fair allocation are quite more scalable than Collabra to the increase in the number of co-hosted VMs. The reason is that in Collabra, as the number of VMs grows up, the number of calls to be stored and analyzed by the hypervisor becomes quite greater, which would entail considerable overhead in large-scale cloud systems.

7 Discussion

In this Section, we provide an in-depth discussion on the originality of our solution compared to the state-of-the-art and shed light on the technical challenges of the different steps of our approach. Starting with the game model, it is true that Bayesian Stackelberg games have already been used in the literature to solve security-oriented problems. For example, a Bayesian Stackelberg game has been proposed in [34] to protect Los Angeles airport against multi-type attacks (e.g., thieves, terrorists). In [15], a Bayesian Stackelberg game has been designed to help Web application administrators chose thoughtful defense strategies to deal with attackers of different types. However, the main limitations of these approaches is that they assume that the attacker types' probability distributions are known a priori, which is not realistic in practice. In this work, we propose a practical and effective data-driven optimization methodology which employs techniques from several disciplines (i.e., risk assessment, MTD, and machine learning) to learn these probabilities and then integrate them into the Bayesian Stackelberg game. Such a data-driven learning methodology provides an effective means for determining the distribution of the attacker types that target a certain cloud system. Moreover, this approach is novel in this field and opens the door for further data-driven optimization solutions in the domain of cybersecurity. Besides, the design of the utility functions in our case is a serious challenge. In fact, the utility functions should be designed while taking into account several factors such as the values of VMs being targeted/protected, type and impact of attacks being launched, effectiveness of monitoring/attack processes, and attack/monitoring costs.

Moving to the MTD technique, it is true that many MTD-based approaches have been proposed to protect assets from being successful targets for attackers. However, adopting it in a cloud environment involves many technical challenges. Specifically, unlike the case of physical security systems where moving checkpoints and patrols across many locations only involves security concerns (i.e., making sure that the new location is safe), migrating services from one VM to another should maintain some technical compatibilities between the migration source and destination. For example, the Operating Systems (OSs) of the source and destination VMs have to be consistent since migration between distinct OSs (e.g., Windows and Linux) might entail some technical complications and unanticipated technological roadblocks. To tackle this challenge, we devise in this work a migration strategy which takes into account both the security and technical aspects in the migration process through intelligently choosing the migration destinations to be the ones that maximize the security and minimize differences in terms of hardware, network, and storage characteristics w.r.t the migration source.

Concerning honeypots, apart from the traditional honeypot deployment techniques, deploying honeypots in the cloud entails many advantages and challenges. On the

one hand, the cloud computing architecture facilitates the deployment of honeypots through offering businesses complete isolation from their production network. Moreover, the use of the cloud eliminates the need for purchasing specific hardware or dedicated Internet connections. That is, once a honeypot machine has been compromised and the data is gathered, a snaposhot might be employed to revert the system back to its captured state prior to the happening of the attack. On the other hand, the use of honeypots in the cloud imposes legal and policy implications for cloud providers. In particular, some providers feel reluctant to direct hackers to their networks and/or to collect malware data within their infrastructure. This in fact could lead to harm their reputation (hosting the compromised system) and even to block their IP ranges and domains, thus impacting their market shares. Therefore, designing a honeypot solution in a cloud environment should be done in careful manner.

8 Conclusion

This paper proposes a comprehensive detection and defense mechanism for cloud-based systems that consists of the following phases: (1) risk assessment framework that evaluates the risk level of each guest VM; (2) MTD-based defense mechanism that intelligently migrates services running inside risky VMs to other more secure VMs; (3) machine learning technique that recognizes the types of attackers using honeypot data; and (4) resource-aware Bayesian Stackelberg game that aids the hypervisor in determining the optimal detection load distribution strategy among VMs. Experiments conducted using Amazon's datacenter and AWSs honeypot data reveal that our solution improves the detection performance up to $\approx 7\%$ and minimizes the percentage of attacked services by $\approx 15\%$ compared to the state-of-the-art detection and defense strategies, namely Collabra, probabilistic migration, maxmin, one-stage Stackelberg, and fair allocation. As for the efficiency, the experimental results show that As for the efficiency, the experimental results show that our machine learning technique needs $\approx 330s$ to train in a dataset comprising 65,000 rows and consisting of six types of attackers. Finally, our detection load distribution strategy takes $\approx 5.6s$ to run in a cloud system of 50 co-hosted VMs and grows polynomially with the increase in the number of co-hosted VMs.

Acknowledgments

This work has been supported by the Fonds de Recherche du Québec—Nature et Technologie (FRQNT), Natural Sciences and Engineering Research Council of Canada (NSERC), Khalifa University of Science, Technology & Research (KUSTAR), Associated Research Unit of the National Council for Scientific Research (CNRS-Lebanon), and Lebanese American University.

References

[1] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Optimal load distribution for the detection of VM-based DDoS attacks in the cloud," *IEEE Trans. Serv. Comput.*, 2017, doi: 10.1109/TSC.2017.2694426.

[2] W. Peng, F. Li, C.-T. Huang, and X. Zou, "A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces," in *Proc. IEEE Int. Conf. Commun.*, 2014, pp. 804–809.

[3] N. Provos, "A virtual honeypot framework," in *Proc. USENIX Security Symp.*, 2004, vol. 173, pp. 1–14.

[4] M. R. Watson, A. K. Marnerides, A. Mauthe, D. Hutchison, et al.," Malware detection in cloud computing infrastructures," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 2, pp. 192–205, Mar.-Apr. 2016.

[5] "Amazon EC2 instances," [Online]. Available: https://aws.amazon.com/ec2/details/, Accessed on: May 16, 2017

[6] "Amazon Web Services honeypot data," [Online]. Available: http://datadrivensecurity.info/blog/pages/dds-dataset-collection.html, Accessed on: May 16, 2017.

[7] B. Li, P. Liu, and L. Lin, "A cluster-based intrusion detection framework for monitoring the traffic of cloud environments," in *Proc. IEEE 3rd Int. Conf. Cyber Security Cloud Comput.*, 2016, pp. 42–45.

[8] T. Alharkan and P. Martin, "IDSaaS: Intrusion detection system as a service in public clouds," in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 686–687.

[9] W. Lin and D. Lee, "Traceback attacks in cloud–pebbletrace botnet," in *Proc. 32nd Int. Conf. Distrib. Comput. Syst. Workshops*, 2012, pp. 417–426.

[10] J. S. Ward and A. Barker, "Varanus: In situ monitoring for large scale cloud systems," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, 2013, vol. 2, pp. 341–344.

[11] P. Deshpande, S. C. Sharma, S. K. Peddoju, and S. Junaid, "HIDS: A host based intrusion detection system for cloud computing environment," *Int. J. Syst. Assurance Eng. Manage.*, vol. 9, no. 3, pp. 567–576, 2018.

[12] S. Bharadwaja, W. Sun, M. Niamat, and F. Shen, "Collabra: A Xen hypervisor based collaborative intrusion detection system," in *Proc. 8th Int. Conf. Inf. Technol.: New Generations*, 2011, pp. 695–700.

[13] F. Lombardi and R. Di Pietro , "Secure virtualization for cloud computing," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1113–1122, 2011.

[14] J. B. Hong and D. S. Kim, "Assessing the effectiveness of moving target defenses using security models," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 2, pp. 163–177, Mar.-Apr. 2016.

[15] S. G. Vadlamudi, S. Sengupta, M. Taguinod, Z. Zhao, A. Doupé, G.-J. Ahn, and S. Kambhampati, "Moving target defense for web applications using bayesian stackelberg games," in *Proc. Int. Conf. Autonomous Agents Multiagent Syst.*, 2016, pp. 1377–1378.

[16] A. R. Hota, A. A. Clements, S. Sundaram, and S. Bagchi, "Optimal and game-theoretic deployment of security investments in interdependent assets," in *Proc. Int. Conf. Decision Game Theory Security*, 2016, pp. 101–113.

[17] A. Clark, K. Sun, L. Bushnell, and R. Poovendran, "A game-theoretic approach to ip address randomization in decoy-based cyber defense," in *Proc. Int. Conf. Decision Game Theory Security*, 2015, pp. 3–21.

[18] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "I know you are watching me: Stackelberg-based adaptive intrusion detection strategy for insider attacks in the cloud," in *Proc. IEEE Int. Conf. Web Services*, 2017, pp. 728–735.

[19] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, MA, USA: MIT Press, 1994.

[20] T. Basar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, vol. 23, Philadelphia, PA, USA: SIAM, 1999.

[21] D. Perez-Botero , J. Szefer, and R. B. Lee, "Characterizing hypervisor vulnerabilities in cloud computing servers," in *Proc. Int. Workshop Security Cloud Comput.*, 2013, pp. 3–10.

[22] "National Vulnerability Database," [Online]. Available: http://web.nvd.nist.gov/view/vuln/search, Accessed on: May 16, 2017

[23] "Securityfocus," [Online]. Available: http://www.securityfocus.com/, Accessed on: May 16, 2017

[24] "Red Hat Bugzilla," [Online]. Available: https://bugzilla.redhat.com/, Accessed on: May 16, 2017

[25] "CVE Security Vulnerability Database," [Online]. Available: http://www.cvedetails.com/, Accessed on: May 16, 2017

[26] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "A stackelberg game for distributed formation of business-driven services communities," *Expert Syst. Appl.*, vol. 45, pp. 359–372, 2016.

[27] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games," in *Proc. 7th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2008, pp. 895–902.

[28] T. S. Ferguson, "Game theory," Los Angeles, CA, USA: Mathematics Department, UCLA, 2008.

[29] B. Guttman and E. A. Roback, *An Introduction to Computer Security: The NIST Handbook*. Darby, PA, USA: DIANE Publishing, 1995.

[30] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 1, pp. 95–108, Jan.-Feb. 2017.
[31] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose , and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw.: Practice Exp.*, vol. 41, no. 1, pp. 23–50, 2011.
[32] J. Jacobs and B. Rudis, *Data-Driven Security: Analysis, Visualization and Dashboards*. Hoboken, NJ, USA: Wiley, 2014.
[33] W. Wang, B. Liang, and B. Li, "Multi-resource fair allocation in heterogeneous cloud computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2822–2835, Oct. 2015.
[34] J. Pita, M. Jain, F. Ordónez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus, "Using game theory for los angeles airport security," *AI Magazine*, vol. 30, no. 1, 2009, Art. no. 43.

Omar Abdel Wahab received the MSc degree in computer science from the Lebanese American University (LAU), Lebanon, in 2013, and the PhD degree in information and systems engineering from Concordia University, Montreal, Canada. He is an assistant professor with the Department of Computer Science and Engineering, Université du Québec en Outaouais, Canada. From 2017 to 2018, he was a postdoctoral fellow with the École de technologie supérieure (ETS), Montreal, Canada. The main topics of his current research activities are in the areas of artificial intelligence, cybersecurity, cloud computing, and big data analytics. He is recipient of many prestigious awards including Quebec Merit Scholarship (FRQNT Québec). Moreover, he is a TPC member of several prestigious conferences and reviewer of several highly ranked journals.

Jamal Bentahar received the bachelor's degree in software engineering from the National Institute of Statistics and Applied Economics, Morocco, in 1998, the MSc degree in software engineering from Mohamed V University, Morocco, in 2001, and the PhD degree in computer science and software engineering from Laval University, Canada, in 2005. He is a full professor with the Concordia Institute for Information Systems Engineering, Faculty of Engineering and Computer Science, Concordia University, Canada. From 2005 to 2006, he was a postdoctoral fellow at Laval University, and then Simon Fraser University, Canada. His research interests include the areas of computational logics, model checking, multi-agent systems, service computing, game theory, and software engineering. He is a member of the IEEE.

Hadi Otrok received the PhD degree in ECE from Concordia University. He holds an associate professor position with the Department of ECE at Khalifa University, an affiliate associate professor with the Concordia Institute for Information Systems Engineering at Concordia University, Montreal, Canada, and an affiliate associate professor with the electrical department at École de technologie supérieure (ETS), Montreal, Canada. He is a senior member of the IEEE, associate editor at: IEEE communications letters and Ad hoc Networks (Elsevier), and a co-chair of several committees at various IEEE conferences. His research interests include Computer and Network Security, Web Services and Cloud Computing, Ad hoc Networks,Application of Game Theory and Mechanism Design.

Azzam Mourad received the PhD degree in electrical and computer engineering from Concordia University, Montreal, Canada. He is an associate professor of computer science at the Lebanese American University and adjunct associate professor with the Sotware Engineering and IT Department at École de technologie supérieure (ETS), Montreal, Canada. His research interests include Information Security, Web Services, Mobile Cloud Computing, Big Data Analysis, Vehicular Networks, and Formal Semantics. He is coordinator of the Associated Research Unit on Intelligent Transport & Vehicular Technologies. He is serving as associate editor for IEEE Communications Letters, General Co-Chair of WiMob2016, and Track Chair, TPC member and reviewer of several prestigious conferences and journals. He is a senior member of the IEEE.