How to distribute the detection load among virtual machines to maximize the detection of distributed attacks in the cloud?

Omar Abdel Wahab*, Jamal Bentahar*, Hadi Otrok†*, Azzam Mourad‡

*Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada
†Department of ECE, Khalifa University of Science, Technology & Research, Abu Dhabi, UAE
‡Department of Computer science and Mathematics, Lebanese American University, Beirut, Lebanon
Email addresses: {o_abul, bentahar}@ciise.concordia.ca, Hadi.Otrok@kustar.ac.ae, azzam.mourad@lau.edu.lb

***Abstract*—Security has been identified to be the principal stumbling-block preventing users and enterprises from moving their businesses to the cloud. The reason is that cloud systems, besides inheriting all the vulnerabilities of the traditional computing systems, appeal to new types of threats engendered mainly by the virtualization concept that allows multiple users' virtual machines (VMs) to share a common computing platform. This broadens the attack space of the malicious users and increases their ability to attack both the cloud system and other co-resident VMs. Motivated by the absence of any approach that addresses the problem of optimal detection load distribution in the domain of cloud computing, we develop a resource-aware maxmin game theoretical model that guides the hypervisor on how the detection load should be optimally distributed among its guest VMs in the real-time. The objective is to maximize the hypervisor's probability of detection, knowing that the attacker is dividing the attack over several VMs to minimize this probability. Experimental results on Amazon EC2 pricing dataset reveal that our model increases the probability of detecting distributed attacks, reduces the false positives, and minimizes the resources wasted during the detection process.**

***Keywords*-Detection load distribution; distributed attack; cloud computing; security; virtualization.**

I. Introduction

Security, along with trust, is identified as one of the major challenges threatening the (apparently) bright future of cloud computing [1], [2]. Particularly, the distributed, virtual, and multi-tenant nature of cloud systems exposes them to a wider set of vulnerabilities, in addition to those present in the traditional computing systems. These vulnerabilities originate from the interactions that take place among the different parties involved in the cloud computing scenario, namely users, services, and cloud systems. Each interaction between each pair of participants can be exploited by attackers to launch their malicious attacks [3]. Practically, the interactions between users and services exhibit the same types of attacks that are applicable apart from cloud such as flooding and SQL injection attacks. The interactions between users and cloud systems entail, inter alia, phishing-like attacks resulting in faked bills. Finally, the interactions between services and cloud systems make room for attacks like resources exhaustion, Denial of Services (DoS), availability reduction, and privacy breach [4]. In this work, we focus on the latter attack surface and consider the scenario in which malicious users owning a set of services in the form of VMs in a cloud system, create distributed attacks by splitting the attack fragments over several VMs to be sent by these VMs at different time intervals. Such attacks may be launched either by one attacker splitting the attack over several VMs he owns or by a coalition of attackers each of which is launching a part of the attack on one (or more) of his VMs [1]. The objective is to compromise the hypervisor (i.e., software agent responsible for regulating the access of the VMs to the hardware resources) and potentially attack other users' co-resident VMs. Interestingly, an experimental study on Amazon EC2 services has shown that attackers, by investing a few more dollars in creating and launching VMs, might be able to increase their chances up to 40% in placing their malicious VMs on the same physical infrastructure as some target users' VMs [5].

Problem Statement. Numerous Intrusion Detection Systems (IDSs) [6], [7], [8], [9], [10] have been proposed for detecting malicious attacks in cloud computing. These systems can be classified into three main categories: network-based, host-based, and hypervisor-based systems. The basic idea of network-based IDSs is to monitor the incoming and outgoing network traffic as well as the content of the packets in order to identify intrusions. These approaches stop at the borders of detecting outside attackers but are ineffective when attackers penetrate into the internal (virtualized) cloud system [8]. Host-based IDSs consist of placing an agent on the VMs to monitor their behavior and states including logs and system calls to identify the malicious behavior. This makes host-based systems more effective than network-based systems in capturing the insider attacks. Nonetheless, such approaches pose additional responsibilities for cloud users who are required to monitor and manage such agents, which plays against their adoption especially when users

[1]For simplification, the term *attacker* is used in the rest of the paper to represent both the case of one attacker and a coalition of attackers.

own multiple service instances in a cloud environment. Moreover, masterful attackers might be able to compromise the monitoring agents by violating the VM instances.

On the other hand, hypervisor-based IDSs move the intrusion detection responsibilities from the user's site to the cloud system's site by placing the IDS at the hypervisor's level. The hypervisor inspects the service instances' system metrics (e.g., CPU utilization, read/write operations) directly from the hosting infrastructure to identify any potential abnormal behavior. However, in real systems, such a mechanism requires monitoring and analyzing a huge number of events [11]. Moreover, the detection of distributed attacks requires the collection and correlation of events coming from different VMs; each of which contains a part of the attack. Nevertheless, the amount of resources dedicated for intrusion detection cannot be infinite since increasing such an amount means decreasing the amount that can be dedicated to serving cloud users. Therefore, a resource-aware detection mechanism that is able to divide the detection load on the set of VMs in such a way that maximizes the probability of detecting distributed attacks is required.

Contributions. Given a limited amount of resources that the hypervisor is allowed to dedicate for intrusion detection, we develop a maxmin game model between the attacker trying to minimize the hypervisor's probability of detection by distributing the attack over a set of VMs and the hypervisor trying to maximize this minimization, while respecting the resources limitation. To the best of our knowledge, this is the first work that considers this challenging problem in the domain of cloud computing. The strategy of the attacker is to select a probability distribution for the attacks over the set of VMs in such a way to minimize the hypervisor's probability of detecting such attacks (e.g., the VMs that the attacker thinks will be the least monitored by the hypervisor). On the other hand, the strategy of the hypervisor is to select a probability distribution for the detection load over the set of VMs it hosts so as to maximize the attacker's minimization. The game is converted then into a Linear Programming problem and solved using the simplex method [12]. The outcome of the game is a probability distribution over the set of VMs informing the hypervisor about the optimal percentage of detection load that should be placed on each of its guest VMs in the real-time.

The performance of proposed model is evaluated with help of the Amazon EC2 pricing dataset [13]. Experimental results show that our model maximizes the probability of detecting attacks and minimizes both the percentage of false positives and percentage of resources wasted during the detection process.

Outline of the paper. Section II presents a literature review on the IDSs in cloud computing and highlights the unique features of our model. Section III formulates the problem. Section IV describes the maxmin game model between the hypervisor and attacker and computes the solution of the game. Section V provides a numerical example to demonstrate how our proposed model can be effectively applied in practical scenarios. Section VI explains the experimental setup and presents experimental results. Finally, Section VII concludes the paper and points out future work.

II. Related Work

As mentioned earlier, the current state-of-the-art IDSs proposed for cloud-based systems can be classified into three main branches: host-based, network-based, and hypervisor-based systems. In the following, we explain the main contribution in each of these branches and highlight the unique features of our proposed model.

A. Network-based Detection Systems

In [14], the authors propose a cooperative IDS for DoS attacks. They assume that an IDS is deployed in each cloud computing region. Each IDS collects network packets and analyzes them. If the type of the packet matches any type defined in the block table (that maintains the bad packet that should be blocked), then this packet is immediately dropped by the IDS. If no match exists but the packet is categorized as anomalous, then the degree of severity of that suspicious packet is checked. If the packet is classified as serious, then the IDS drops it and notifies the other IDSs accordingly. If the packet is classified as moderate, the IDS performs data clustering and threshold check to find outliers and updates the alert level accordingly. Finally, if the packet is identified as slight, then the system simply ignores the alert.

In [15], the authors address the botnet attack in cloud environment by proposing a scheme for tracing back the botmaster (i.e., the malicious user that administrates the botnet). In the proposed scheme, the local network administrator of the victim machine collects information (i.e., memory images, network traffic between bots and Command-and-Control (C&C) servers, and hostname of the C&C server), files them to a traceback server, and asks the latter for a traceback service. The traceback server embeds then Pebbleware, a piece of code that reveals its host machine's information, on the communication packets from the victim node to the botmaster. Once the Prebbleware reaches the botmaster, the latter's machine is obliged to send its IP address to the traceback server.

To conclude, the basic idea of network-based IDSs is to monitor the incoming and outgoing network traffic to detect intrusions. The main limitation of these approaches is their ineffectiveness in capturing the internal attacks wherein attackers penetrate into the internal cloud system.

B. Host-based Detection Systems

In [16], the authors propose a distributed detection mechanism for detecting Distributed Denial of Service (DDoS) attacks in cloud environments. The basic idea is to deploy and configure an IDS within each VM whose responsibility

is to collect alerts. The alerts from the different VMs are then sent and stored into a Mysql database hosted in the cloud fusion unit of the front-end server. Thereafter, the alerts are converted into basic probabilities assignments and analyzed using Dempster-Shafer's combination rule [17].

In [9], the authors propose Varanus, a multi-tier detection model for large-scale Infrastructure as a Service (IaaS) cloud systems. In Varanus, VMs are partitioned into a set of groups based on the similarity between their software configuration features (e.g., web servers, database servers and compute nodes) using the k-nearest neighbor clustering algorithm. Each VM participates in a gossip-based monitoring scheme by propagating and receiving information (e.g., CPU usage, network traffic) to/from the other VM agents in the same group. In each group, the under-utilized VMs are then nominated to perform the data analysis. Finally, the aggregate value for each group of VMs is communicated between the different groups located in the same cloud domain.

Summarizing, the basic idea of host-based IDSs is to deploy a monitoring agent on each VM to monitor its states and behavior and identify hence any malicious behavior. Although these systems are more effective than network-based IDSs in identifying both internal and external attacks, such systems entail additional monitoring and management responsibilities for the users and can be impeded by masterful attackers who violate the VM instances.

C. Hypervisor-based Detection Systems

In [6], the authors propose a virtualization-supported security architecture for cloud resources called Advanced Cloud Protection System (ACPS). The basic idea is to monitor the integrity of guest VMs and hosting infrastructure components, while being invisible to the end users. To detect attacks, system-call invocations performed by VMs are being continuously monitored by an *Interceptor* entity that is located into the kernel space of the hosting platform. Suspicious activities are then logged by a *Warning Recorder* entity into the *Warning Pool* that prioritizes the order of evaluation of these activities. The *Warning Recorder* asynchronously computes checksums for critical host infrastructure and guest kernel code, data, and files. This information is passed then to the *Evaluator* entity that examines the events and decides whether the system's security has been breached or not.

In [8], the authors propose a hypervisor-based IDS for malicious activities on VMs. Every second, endpoint agents installed on hypervisors retrieve the performance metrics of the host VMs such as CPU utilization, block device read/write data, and network data transmitted/received. This data is transmitted to the controller node, a service residing within the cloud environment, which is responsible for analyzing the received data against stored signatures and confirming the existence of an attack or not.

To summarize, the existing IDSs stop at the borders of monitoring and analyzing events to identify intrusions. Thus, these systems consider the detection problem from the perspective of the IDS only without accounting for the strategies of the attacker who seeks to minimize the probability of detecting his attacks. Particularly, the problem of splitting the detection load over a set of VMs in order to maximize the detection of distributed attacks, which we address in this work, has not been tackled yet in the domain of cloud computing.

III. Problem Formulation

As depicted in Fig. 1, a virtualized cloud system consists of a set of hypervisors $H = \{h_1, h_2, ..., h_n\}$; each of which is hosting a set of virtual machines $V_h = \{v_{1h}, v_{2h}, ..., v_{lh}\}$ owned by clients from the set $C = \{c_1, c_2, ..., c_k\}$, where each client $c_i \in C$ owns one or more virtual machines. The clients may be either *well-behaving* or *malicious*. Well-behaving clients are simply those who seek to accomplish their work smoothly without having the intention to harm neither the cloud system nor the other clients. Malicious clients, on the other hand, seek to launch attacks leading to harm the cloud system and/or some other clients' co-resident VMs. A hypervisor $h_i \in H$ is a software agent residing between the cloud system's hardware and the guest VMs and whose goal is to allow the concurrent running of multiple operating systems abstracted as VMs on a shared hardware.

*Definition 1 (**Virtualized Cloud System**):* A virtualized cloud system consists of a set of hardware resources $I = \{I_1, ..., I_k\}$ managed by a set hypervisors $H = \{h_1, ..., h_k\}$; where each hypervisor h is hosting a set of VMs $V_h = \{v_{1h}, \ldots, v_{lh}\}$ owned by a set of clients $C = \{c_1, ..., c_n\}$ to provide each $v_{ih} \in V_h$ with a view that its operating system and applications are operating directly on some physical hardware.

Thus, the role of the hypervisor consists of emulating the hardware system and scheduling the access of the VMs to it. Achieving this demands recurrent interactions between the guest VMs and the underlying hypervisor. These interactions are the paramount source of the security threats that the attacker exploits to inject his malicious attacks to harm hypervisor and to take advantage of the software bugs present in the hypervisor's source code to attack other co-resident VMs (a.k.a co-resident attack) [18]. For example, attackers may exploit the VM exit operations (or hypercalls in case of paravirtualization) whose frequency is quite high (e.g., an idle VM running on top of Xen 4.0 performs ~ 600 VM exit operations per second) to inject malicious code leading to breach the integrity or confidentiality of other co-resident VMs, crash or slow down the hypervisor, and/or cause denial-of-service to violate the availability of the system.

In order to complicate the detection process and rip off the hypervisor, the attacker can use a mixed strategy by distributing the attack over multiple VMs running on top of the same hypervisor. To this end, the attacker splits its attack

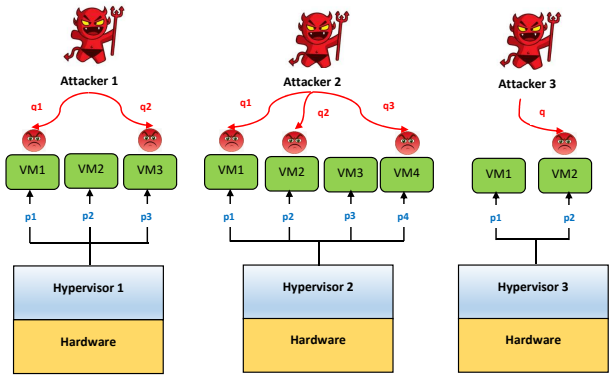


Figure 1: Attack Scenario: Attackers distribute their attacks over a set of malicious VMs to minimize the detection probability, while hypervisors distribute the detection load over the set of guest VMs to maximize this minimization.

code into several malicious fragments and assigns a set of fragments to each VM. Each malicious VM aims at sending k malicious fragments to the hypervisor at different time intervals. Let $Q_{V_h} = (q(v_{1h}), ..., q(v_{lh}))$ denote the probability distribution vector over the set of VMs V_h that the attacker owns on a certain hypervisor h such that $\sum_{v_{ih} \in V_h} q(v_{ih}) = 1$. The attack succeeds if one or many malicious fragments attain the hypervisor without being detected (the number of fragments depends on the type of the launched attack).

Definition 2 ***(Distributed Attack):*** A distributed attack is a set of k malicious code fragments $\{f_1, ..., f_k\}$ distributed over V_h with a probability of $q(v_{ih})$ for each $v_{ih} \in V_h$ such that $\sum_{v_{ih} \in V_h} q(v_{ih}) = 1$.

Knowing this fact, the hypervisor, having a limited amount of resources to be dedicated for detection, has to choose a mixed strategy consisting of the optimal detection load probability distribution vector $P_{V_h} = (p(v_{1h}), ..., p(v_{lh}))$ over the set of VMs V_h hosted on top of it such that $\sum_{v_{ih} \in V_h} p(v_{ih}) = 1$. Note that the detection process entails a cost for the hypervisor that is function of the power used to analyze events, cost of collecting events, and computational (CPU and memory usage) cost.

IV. Determining the Optimal Detection Load Distribution: A Maxmin Game

Based on the strategies chosen by both the attacker (probability distribution Q_{V_h} of the attack over the set of VMs) and hypervisor (probability distribution P_{V_h} of the detection load over the set of host VMs), the payoff of the hypervisor is quantified in terms of the probability of detecting intrusions at a certain time window $[t_1, t_2]$ in which time is assumed to be discrete, proportionally to the price of each protected VM. The utility function of hypervisor h at time $t_2 + 1$ is computed as follows:

$$U_{t_2+1}(h) = \sum_{v_{ih} \in V_h} W(v_{ih}) \times \beta_{[t_1,t_2]}, \tag{1}$$

where $W(v_{ih})$ represents the worth of each virtual machine v_{ih} (e.g., in terms of price, criticality of the applications running on it, etc.), and $\beta_{[t_1,t_2]}$ is the average detection rate of the IDS agent running on h at the time window $[t_1, t_2]$ and is computed as per Eq. (2).

$$\beta_{[t_1,t_2]} = 1 - \sum_{x=t_1}^{t_2} \sum_{v_{ih} \in V_h} \frac{(q_x(v_{ih}) - p_x(v_{ih}))}{t_2 - t_1} \text{ for each } q_x(v_{ih}) > p_x(v_{ih}), \tag{2}$$

where $p_x(v_{ih})$ is the value of $p(v_{ih})$ at time x and $q_x(v_{ih})$ is the value of $q(v_{ih})$ at time x. It is worth mentioning that all the calculations in the rest of the paper are done at time $t_2 + 1$ (i.e., the current time for the hypervisor). Thus, we simplify the notation and use $U(h)$ instead of $U_{t_2+1}(h)$ when referring to hypervisor's h utility at time $t_2 + 1$. The payoff of the attacker a represents the loss incurred to the hypervisor as a result of a successful attack. Therefore, the payoff of the attacker is the negation of the hypervisor's payoff, i.e.,

$$U(a) = -U(h) \tag{3}$$

This forms a hypervisor-attacker (i.e., two-player) zero-sum game in which one player's gain is equivalent to the other player's loss.

Definition 3 ***(Hypervisor-attacker Zero-sum Game):*** A hypervisor-attacker zero-sum game is a tuple $G = \langle h, a, P_{V_h}, Q_{V_h}, U(h), U(a) \rangle$, where:

- h: denotes the hypervisor (i.e., the first player).
- a: denotes the attacker (i.e., the second player).
- P_{V_h}: denotes the probability distribution vector of the detection load over the set of VMs V_h hosted on top of h (i.e., the mixed strategy of h).
- Q_{V_h}: denotes the probability distribution vector of the attack over the set of VMs V_h hosted on top of h (i.e., the mixed strategy of a).
- $U(h)$: the utility function of the hypervisor h.
- $U(a)$: the utility function of the attacker a such that $U(a) = -U(h)$.

The objective of the attacker is to choose its probability distribution Q_{V_h} for distributing the attack over the VMs' set with the aim of minimizing the hypervisor's detection probability and hence minimizing the latter's payoff, i.e.,

$$\arg\min_{Q_{V_h}} U(h) \tag{4}$$

Knowing this fact, the hypervisor would choose a probability distribution P_{V_h} over the set of VMs in such a way to maximize the attacker's minimization, i.e.,

$$\arg\max_{P_{V_h}} \min_{Q_{V_h}} U(h) \tag{5}$$

This forms a maxmin game wherein the attacker tries to minimize the hypervisor's probability of detecting attacks by distributing the attack over multiple VMs, whereas the hypervisor tries to maximize this minimization by choosing the optimal distribution of detection load over the VMs.

*Definition 4 (**Hypervisor's Maxmin Strategy**):* The maxmin strategy for the hypervisor h is $\arg\max_{P_{V_h}} \min_{Q_{V_h}} U(h)$ and the maximin value for h is $\max_{P_{V_h}} \min_{Q_{V_h}} U(h)$.

The solution of the game can be devised using Linear Programming, referred to as the problem of determining the values of some real variables for the purpose of minimizing or maximizing a linear function (known as objective function) subject to linear constraints on these variables. To this end, let us consider the problem first from the point of view of the hypervisor trying to maximize the minimum of the attacker and let us rewrite Eq. (5) as follows:

$$\begin{aligned} \text{maximize} \quad & \min_{Q_{V_h}} \sum_{v_{ih} \in V_h} p(v_{ih}) \times U(h) \\ \text{subject to} \quad & \sum_{v_{ih} \in V_h} p(v_{ih}) = 1, \\ & p(v_{ih}) \ \geq \ 0, \ for\ all\ v_{ih} \in V_h. \end{aligned} \tag{6}$$

By inspecting Eq. (6), we can notice that the objective function is not linear in the p's owing to the presence of the *min* operator. Therefore, the problem in its current form cannot be solved using linear programming. To linearize it, we define a variable f such that $f \leq \min_{Q_{V_h}} \sum_{v_{ih} \in V_h} p(v_{ih}) \times U(h)$ and try to make f as large as possible subject to this new constraint. Thus, the problem is turned into choosing f and $\sum_{v_{ih} \in V_h} p(v_{ih})$ to:

$$\begin{aligned} \text{maximize} \quad & f \\ \text{subject to} \quad & f \leq \sum_{v_{ih} \in V_h} p(v_{ih}) \times U(h), \\ & p(v_{1h}) + \cdots + p(v_{lh}) = 1, \\ & p(v_{ih}) \ \geq \ 0, \ for\ all\ v_{ih} \in V_h. \end{aligned} \tag{7}$$

Intuitively, this means that the hypervisor, by choosing its mixed strategy $p(v_{ih}) \in P_{V_h}$, is trying to make as large as possible the minimum that the attacker is attempting to inflict on it by playing his mixed strategy $q(v_{ih}) \in Q_{V_h}$. To ease the computations, we transform the linear program presented in Eq. (7) into a simpler form. Assume that $f > 0$ and let $x(v_{ih}) = \frac{p(v_{ih})}{f}$. The constraint $p(v_{1h}) + \cdots + p(v_{lh}) = 1$ becomes then $x(v_{1h}) + \cdots + x(v_{lh}) = 1/f$. Since maximizing f is equivalent to minimizing f's reciprocal $1/f$, we can get rid of f in our problem by rather minimizing $x(v_{1h}) + ... + x(v_{lh})$. Thus, the problem becomes: choose $x(v_{1h}) + \cdots + x(v_{lh})$ to:

$$\begin{aligned} \text{minimize} \quad & x(v_{1h}) + \cdots + x(v_{lh}) \\ \text{subject to} \quad & 1 \leq \sum_{v_{ih} \in V_h} x(v_{ih}) \times U(h), \\ & x(v_{ih}) \ \geq \ 0, \ for\ all\ v_{ih} \in V_h. \end{aligned} \tag{8}$$

The above problem may be solved in polynomial time using the simplex method for solving Linear Programming, which is known to perform very fast [12]. Having solved the problem, the hypervisor's optimal strategy would be $p(v_{ih}) = f \times x(v_{ih})$ for each $v_{ih} \in V_h$.

If we consider the problem from the point of view of the attacker, the latter's objective is to minimize the hypervisor's maximal probability of detection.

*Definition 5 (**Attacker's Minmax Strategy**):* The minmax strategy for the attacker a is $\arg\min_{Q_{V_h}} \max_{P_{V_h}} U(h)$ and the minmax value for a is $\min_{Q_{V_h}} \max_{P_{V_h}} U(h)$.

The problem can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \max_{P_{V_h}} \sum_{v_{ih} \in V_h} q(v_{ih}) \times U(h) \\ \text{subject to} \quad & \sum_{v_{ih} \in V_h} q(v_{ih}) = 1, \\ & q(v_{ih}) \ \geq \ 0, \ for\ all\ v_{ih} \in V_h. \end{aligned} \tag{9}$$

Using the similar logic of transformation followed for the hypervisor's maximization problem, the problem in Eq. (9) can be rewritten as:

$$\begin{aligned} \text{minimize} \quad & g \\ \text{subject to} \quad & g \geq \sum_{v_{ih} \in V_h} q(v_{ih}) \times U(h), \\ & q(v_{1h}) + ... + q(v_{lh}) = 1, \\ & q(v_{ih}) \ \geq \ 0, \ for\ all\ v_{ih} \in V_h. \end{aligned} \tag{10}$$

By carefully examining Eq. (7) and Eq. (10), we can notice that the two programs are dual. Following the duality theorem [19], the maximum that the hypervisor can realize in Eq. (10) is equivalent to the minimum that the attacker can achieve in Eq. (7).

V. Numerical Example

Consider a hypervisor hosting, at time t, three VMs V_1, V_2, and V_3 having prices of 9.412\$, 5.88\$ and 8.23\$ per hour respectively. Assume as well that the detection probability of

that hypervisor at time t is 0.85. By using Eq. (1) and Eq. (3) for computing the utility values of the hypervisor and attacker respectively, we obtain the following game matrix:

$$U = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \end{array} \begin{array}{c} \begin{array}{ccc} V_1 & V_2 & V_3 \end{array} \\ \begin{pmatrix} 8 & -5 & -7 \\ -8 & 5 & -7 \\ -8 & -5 & 7 \end{pmatrix} \end{array}$$

In this matrix, the row represents the hypervisor, while the column represents the attacker. Since the studied game is a zero-sum game in which the hypervisor's gain is equal to the attacker's loss and vice versa, we simplify notations and show only in the matrix the payoff of the hypervisor. Thus, $U(i,j)$ represents the utility of the hypervisor when this latter is monitoring V_i and the attacker is attacking on V_j, while $-U(i,j)$ would represent the utility of the attacker in that case. For example, If the hypervisor is monitoring V_1 while the attacker is attacking V_1, then the hypervisor would gain $U(1,1) = 9.412 \times 0.85 = 8$ for having successfully protected V_1 and the attacker would lose 8 for having his attack failed. On the other hand, If the hypervisor is monitoring V_1 while the attacker is attacking V_2, then the hypervisor would lose $U(1,2) = 5.88 \times 0.85 = 5$ for failing to protect V_2 and the attacker would gain 5. The problem of determining the optimal detection load distribution can be solved using the simplex method by following the subsequent steps:

Step 1**:** *Add a constant to all the matrix's entries, if necessary, to make sure that all the entries are non-negative.*

In order to make all U's elements non-negative, we need to add 8 to all of its entries. Thus, the game matrix becomes:

$$U' = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \end{array} \begin{array}{c} \begin{array}{ccc} V_1 & V_2 & V_3 \end{array} \\ \begin{pmatrix} 16 & 3 & 1 \\ 0 & 13 & 1 \\ 0 & 3 & 15 \end{pmatrix} \end{array}$$

Step 2**:** *Create a tableau T by (1) extending the matrix with a border of $+1'$s along the right edge, $-1'$s along the lower edge, and zero in the lower right corners and (2) labelling the hypervisor's strategies on the left from x_1 to x_m and those of the attacker on the top from y_1 to y_n.*

By applying step 2, we get:

	y_1	y_2	y_3	
x_1	16	3	1	1
x_2	0	13	1	1
x_3	0	3	15	1
	-1	-1	-1	0

Step 3**:** *Select the pivot belonging to row a and column b subject to the following properties:*

1) *The border number in the lower edge of the pivot's column b must be negative.*
2) *The pivot $T(a,b)$ must be positive.*
3) *The pivot should belong to the row giving the smallest ratio (of the border number in right edge to the pivot) among all the positive entries in the pivot column.*

Since all the three columns in T contain a negative value in their lower edge, it's possible to select any of them to be the pivot column. Let's choose column 1. Since the pivot must be positive, then the selection space is restricted to the first row. The pivot would be then $T(1,1) = 16$.

Step 4**:** *Perform the pivoting steps as follows:*

1) *Substitute the pivot value with its reciprocal.*
2) *Substitute each element in the pivot row, except for the pivot, with its value divided by the value of the pivot.*
3) *Substitute each element in the pivot column, except for the pivot, with the negative of its value divided by the value of the pivot.*
4) *Substitute each element $T(i,j)$ not belonging neither to the pivot row nor to the pivot column with $T(i,j) - T(a,j) \times T(i,b)/T(a,b)$.*

Step 5**:** *Substitute the label of the pivot row with that of the pivot column and vice versa.*

By applying steps 4 and 5, we get:

	x_1	y_2	y_3	
y_1	$1/16$	$3/16$	$1/16$	$1/16$
x_2	0	13	1	1
x_3	0	3	15	1
	$1/16$	$-13/16$	$-15/16$	$1/16$

Step 6**:** *Check whether there is any negative number remaining in the lower border row. If so, return to step* 3*; otherwise, jump to step* 7*.*

Since there still exists two negative entries in the lower border row, we go back to step 2 and apply the pivoting process anew. This process is repeated until getting all the entries in the lower border row non-negative. Once this condition is achieved, the tableau would become:

	x_1	x_2	x_3	
y_1	$1/16$	$-1/72$	$-1/303$	$1/22$
y_2	0	$5/64$	$-1/192$	$3/41$
y_3	0	$-1/64$	$4/59$	$1/19$
	$1/16$	$2/41$	$1/17$	$7/41$

Now that all the values in the lower border row are non-negative, we can proceed with step 7.

Step 7**:** *The solution is determined as follows:*

1) *The optimal strategy of the hypervisor is (1) zero for the hypervisor's variables that end up on the left side, and (2) the value of the bottom edge in the same column divided by the lower right corner for those that end up on the top.*
2) *The attacker's optimal strategy is (1) zero for the attacker's variables that end up on the top, and (2)*

the value of the right edge in the same row divided by the lower right corner for those that end up on the left.

In our example, the optimal detection load probability distribution of the hypervisor over its guest VMs would be $P(V_1) = \frac{1/16}{7/41} = 0.3664$, $P(V_2) = \frac{2/41}{7/41} = 0.2863$, and $P(V_3) = \frac{1/17}{7/41} = 0.3473$, whereas the optimal attack probability distribution of the attacker over the VMs would be $q(V_1) = \frac{1/22}{7/41} = 0.2672$, $q(V_2) = \frac{3/41}{7/41} = 0.4275$, and $q(V_3) = \frac{1/19}{7/41} = 0.3053$.

VI. Experimental Results

In this section, we describe the experimental setup and present experimental results.

A. Experimental Setup

In this section, we conduct experimentations to validate the performance of our model. To this end, we create 75 hypervisors and assign to each of them a number of VMs varying from 5 to 25, where this number is realistic to study the scalability of our model on today's existing cloud systems (e.g., VMware ESXi 5.X [2]). The Amazon EC2 pricing dataset [13] has been used to populate the prices of the VMs that are used to compute the utility functions of both the hypervisor and attacker. The implementation has been performed using MATLAB 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. Lack of any existing model that deals with the problem of distributing the detection load among VMs in cloud systems, we compare our model against a common resources allocation strategy, namely the fair allocation strategy in which the hypervisor splits the detection load over the VMs in an equal manner. We believe that comparing with models that simply monitor and analyze events without accounting for the sophisticated strategies of the attackers would be unfair for both parties.

B. Experimental Results

Fig. 2a measures the false positive percentage that represents the number of attacks that the system was not able to capture during the detection process. This percentage is computed by subtracting the probability distributions of the attacker from those of the hypervisor when the values of the former are greater. The average false positive rate $\alpha_{[t_1,t_2]}$ at time window $[t_1,t_2]$ is computed as follows:

$$\alpha_{[t_1,t_2]} = \sum_{x=t_1}^{t_2} \sum_{v_{ih} \in V_h} \frac{(q_x(v_{ih}) - p_x(v_{ih}))}{t_2 - t_1} \text{ for each } q_x(v_{ih}) > p_x(v_{ih}). \tag{11}$$

[2]http://searchdatacenter.techtarget.com/feature/How-many-VMs-per-host-is-too-many

The false positive rate in the example given in Section V would be: $\alpha = [(0.4275 - 0.2863)] = 0.1412$. Thus, the percentage of false positives is: $0.1412 \times 100 = 14.12\%$.

Fig. 2a shows that our model decreases the false positives considerably compared to the fair allocation model. The reason is that our model considers the problem from the perspectives of both hypervisor trying to maximize the detection probability and attacker trying to minimize this maximization in contrary to the fair allocation model that considers solely the hypervisor's perspective without accounting for the optimal distribution of the attacker. Thus, the distribution in the latter model is done in an arbitrary manner without studying the attacker's strategy space.

Fig. 2b measures the percentage of detected attacks. The average percentage of detected attacks $\beta_{[t_1,t_2]}$ at time window $[t_1,t_2]$ is computed as per Eq. (2). For the same arguments explained in the context of false positives, our model is able to significantly increase the percentage of detected attacks compared to the fair allocation strategy as shown in Fig. 2b.

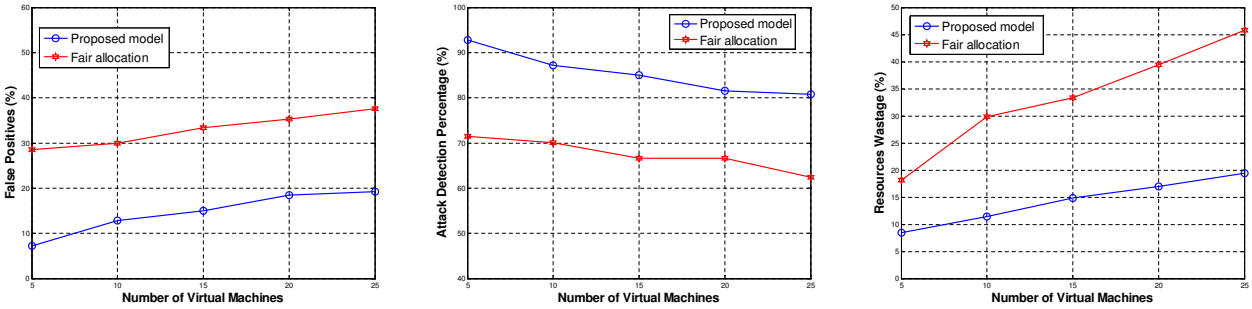
Fig. 2c measures the percentage of resources wasted during the detection process. This metric is computed by subtracting the probability distributions of the hypervisor from those of the attacker when the values of the former are greater. The average rate of resources wasted $\gamma_{[t_1,t_2]}$ at time window $[t_1,t_2]$ is computed as follows:

$$\gamma_{[t_1,t_2]} = \sum_{x=t_1}^{t_2} \sum_{v_{ih} \in V_h} \frac{(p_x(v_{ih}) - q_x(v_{ih}))}{t_2 - t_1} \text{ for each } p_x(v_{ih}) > q_x(v_{ih}), \tag{12}$$

The rate of resources wasted in the example given in Section V would be: $\gamma = [(0.3664 - 0.2672) + (0.3473 - 0.3053)] = 0.1412$. Thus, the percentage of resources wasted is: $0.1412 \times 100 = 14.12\%$. Intuitively, this metric measures the percentage of resources spent by the hypervisor in monitoring the VMs while these VMs are not sending any attack fragment. Fig. 2c reveals that our model is able to considerably reduce the wasted resources. This is justified by the fact that our model guides the hypervisor on the optimal distributions of the detection load that best synchronises with the probability distributions of the attacker.

VII. Conclusion

In this paper, we addressed the problem of determining the optimal distribution of detection load over the set of VMs coresiding on a shared hypervisor in such a way to maximize the probability of detecting distributed attacks. To the best of our knowledge, this is the first work that addresses this challenging problem in the domain cloud computing. We modeled the problem as a hypervisor-attacker maxmin game in which the hypervisor seeks to maximize the probability of detection under a limited amount of resources that can be used for this purpose, knowing that the attacker is trying to minimize this maximization by distributing the attack over a set of several VMs. Simulation results reveal that our model



(a) False Positives (b) Attack Detection (c) Resources Wastage

Figure 2: Our model increases the percentage of detected attacks and decreases the percentages of false positives and resources wastage compared to the fair allocation strategy.

is able to increase the percentage of attack detection up to 20%, while reducing both the percentage of false positive up to 20% and percentage of resources wastage up to 18%.

ACKNOWLEDGMENT

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 (LAU).

REFERENCES

[1] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game," *IEEE Transactions on Services Computing*, 2016.

[2] Z. Han, X. Li, and E. Stroulia, "A hierarchical security-auditing methodology for cloud computing," in *SCC*, 2015, pp. 202–209.

[3] N. Gruschka and M. Jensen, "Attack surfaces: A taxonomy for attacks on cloud services," in *3rd international conference on Cloud Computing*. IEEE, 2010, pp. 276–279.

[4] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "A survey on trust and reputation models for Web services: Single, composite, and communities," *Decision Support Systems*, vol. 74, pp. 121–134, 2015.

[5] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and Communications Security*. ACM, 2009, pp. 199–212.

[6] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1113–1122, 2011.

[7] U. Tupakula, V. Varadharajan, and N. Akku, "Intrusion detection techniques for infrastructure as a service cloud," in *DASC*. IEEE, 2011, pp. 744–751.

[8] J. Nikolai and Y. Wang, "Hypervisor-based cloud intrusion detection system," in *International Conference on Computing, Networking and Communications (ICNC)*, 2014, pp. 989–993.

[9] J. S. Ward and A. Barker, "Varanus: In situ monitoring for large scale cloud systems," in *CloudCom*, vol. 2. IEEE, 2013, pp. 341–344.

[10] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Misbehavior detection framework for community-based cloud computing," in *FiCloud*. IEEE, 2015, pp. 181–188.

[11] N. Ezzati-Jivan and M. R. Dagenais, "A framework to compute statistics of system parameters from very large trace files," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 1, pp. 43–54, 2013.

[12] T. S. Ferguson, "Game theory text," *Mathematics Department, UCLA*, 2008.

[13] "Amazon EC2 Pricing," http://aws.amazon.com/ec2/pricing/, 2016, accessed: 2016-03-21.

[14] C.-C. Lo, C.-C. Huang, and J. Ku, "A cooperative intrusion detection system framework for cloud computing networks," in *ICPPW*. IEEE, 2010, pp. 280–284.

[15] W. Lin and D. Lee, "Traceback Attacks in Cloud–Pebbletrace Botnet," in *ICDCSW*, 2012, pp. 417–426.

[16] A. M. Lonea, D. E. Popescu, and H. Tianfield, "Detecting DDoS attacks in cloud computing environment," *International Journal of Computers Communications & Control*, vol. 8, no. 1, pp. 70–78, 2013.

[17] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game."

[18] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM conference on Computer and Communications Security*. ACM, 2011, pp. 401–412.

[19] D. G. Luenberger, *Introduction to linear and nonlinear programming*. Addison-Wesley Reading, MA, 1973, vol. 28.