Abusitta *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2018) 7:9
https://doi.org/10.1186/s13677-018-0109-4

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

An SVM-based framework for detecting DoS attacks in virtualized clouds under changing environment

Adel Abusitta, Martine Bellaiche* and Michel Dagenais

Abstract

Cloud Computing enables providers to rent out space on their virtual and physical infrastructures. Denial of Service (DoS) attacks threaten the ability of the cloud to respond to clients requests, which results in considerable economic losses. The existing detection approaches are still not mature enough to satisfy a cloud-based detection systems requirements since they overlook the changing/dynamic environment, that characterises the cloud as a result of its inherent characteristics. Indeed, the patterns extracted and used by the existing detection models to identify attacks, are limited to the current VMs infrastructure but do not necessarily hold after performing new adjustments according to the pay-as-you-go business model. Therefore, the accuracy of detection will be negatively affected. Motivated by this fact, we present a new approach for detecting DoS attacks in a virtualized cloud under changing environment. The proposed model enables monitoring and quantifying the effect of resources adjustments on the collected data. This helps filter out the effect of adjustments from the collected data and thus enhance the detection accuracy in dynamic environments. Our solution correlates as well VMs application metrics with the actual resources load, which enables the hypervisor to distinguish between benignant high load and DoS attacks. It helps also the hypervisor identify the compromised VMs that try to needlessly consume more resources. Experimental results show that our model is able to enhance the detection accuracy under changing environments.

Keywords: Cloud computing, DoS attacks detection, Support vector machine, Changing environment, Virtual machines

Introduction

Several major Information and Communications Technology (ICT) companies are competing for creating advanced cloud computing services that are able to deal with small, medium-sized and large-scale enterprise demands. Many companies, organizations and governments are expected to transfer, if not already done, all or parts of their IT solutions to the cloud [1, 2]. This transfer is profitable from an economic point of view since it allows them to streamline the spending on technology infrastructure and capital cost. However, the security threat in terms of Denial of Service (DoS) attacks constitutes a major obstacle against the achievement of this transfer. A DoS attack can be of many types and may be seen in different contexts (e.g., application, web services, network) [3]. However, in this paper, we consider Virtual Machine (VM)-based DoS attacks in a virtualized cloud and define a DoS attack as follows. A DoS attack occurs when one or more VMs drain all the available physical resources such that the hypervisor would not be able to support more VMs [4]. This attack is mainly caused by virtualization [4, 5], which is the backbone of the recent cloud computing architecture, where virtualization allows emulating a particular computer system and sharing physical resources (e.g., CPU and network bandwidth). In this paper, we shed light on the problem of detecting cloud-based DoS attacks under a changing environment. Although several advanced approaches have been proposed to detect DoS attacks in virtualized cloud (e.g., [6–9]), these approaches still causes a significant decrease in the detection accuracy when used in a cloud environment. The reason is that the current approaches do not consider the changing environment,

*Correspondence: martine.bellaiche@polymtl.ca
Department of Computer and Software Engineering, Polytechnique Montreal, 2900 Boulevard Edouard-Montpetit, Montreal, QC H3T 1J4 Canada

that characterises the cloud as a result of its inherent characteristics (resources restriction and scaling). Such characteristics are essential for the VM to meet the requirements of the pay-as-you-go business model [1].

Motivating example

Assume that a cloud provider trained an Support Vector Machine (SVM) classifier on some of the features of the VMs under a certain infrastructure. These features include CPU, network, memory and I/O load. Assume now that the cloud provider, due to some business factors, decides to adjust some of the resources of the VMs. This adjustment includes revoking 45% from some of the resources of the VMs. Such an adjustment will result in a significant decrease in the DoS detection accuracy rate. The reason is that the features used to train the SVM classifier were extracted under the original infrastructure (before revoking 45% from VMs resources). However, these features become unsuitable in the light of the new adjustment in the VMs resources. In other words, the collected data will be affected by the new adjustment, which will lead to an inaccurate classification of the collected data. Tables 1 and 2 show our results of testing the impact of applying resources adjustments on the basic resources of the VMs (CPU, Memory, I/O and Network). We used the API of libvirt that employs cgroups [10] to adjust and limit the resources of the VMs. Using cgroups allows us to exploit Linux Kernel features which limit and allocate resources to VMs—such as CPU time, system memory, network bandwidth, or combinations of these resources [11]. The results show that the detection rate has been decreased as a result of revoking/granting resources from/to the VMs. The details of this experiment are described in "Experimental results and analysis" section.

Indeed, the continuous requests to make adjustments on the infrastructure are necessary as long as the cloud client (e.g., VM) wants to meet the Quality of Service (QoS) requirements. The reason is that the ability of performing new adjustments, to cope with the real-time economic factors, affects the decision of the industries, organizations and governments on whether to adopt or not cloud computing. In other words, the continuous adjustments are necessary for the continuous use of the cloud to meet the variations in the demands and the cost-efficiency, which are considered as the main cloud features.

Table 1 Attack detection rates when revoking resources (CPU, Memory, I/O and Network) from VMs

Resources revoked from VMs	Attack detection rate
0% (baseline)	95.02%
10%	95.79%
20%	90.28%
40%	89.08%
60%	85.18%
80%	83.67%

Table 2 Attack detection rates when granting resources (CPU, Memory, I/O and Network) to VMs

Resources granted to VMs	Attack detection rate
0% (baseline)	95.02%
10%	95.79%
20%	85.28%
40%	84.08%
60%	75.18%
80%	74.67%

Our proposed solution

To address the aforementioned problems, we propose a flexible detection framework based on the SVM learning technique. SVM is a classification technique that employs a nonlinear mapping to convert the original data into higher-dimensional of data, in order to find a hyperplane that optimally separates the training tuples based on their classes [12]. Our framework can be summarized as follows. The hypervisor collects some features to train the SVM classifier to be able to distinguish between the normal activity and DoS attack on the VM. The hypervisor then monitors and quantifies the effect of performing resources adjustments (i.e., granting/revoking resources to/from the VMs) on the collected VMs performance data. This information (i.e, effect of performing resources adjustments) is used thereafter to maintain a filter of resources adjustments effect. The filter is used as a preprocessing step, prior to classification, to get rid of the "noise" that may show up on the collected data (due to the new adjustments) and that may considerably decrease the accuracy of the detection.

Moreover, the proposed framework enables VMs to regularly declare their current application metrics, such as number of clients, requests and sales. This is then used by the hypervisor to correlate these metrics with the actual resources load. This correlation enables the hypervisor to distinguish between benignant high load and DoS attacks. In addition, it enables the hypervisor to identify the compromised VMs that may try to claim and consume more resources. We propose a correlation technique that the hypervisor uses to calculate the expected resources load of the current compromised VMs based on the declared metrics. The calculated resources load is then compared with the actual resources load. If the

calculated resources load is not within a certain range of the actual resources load, the belief that the VM has been compromised increases. In summary, we propose a comprehensive framework that consists of the following contributions:

- Proposing a detection approach to identify DoS attacks in a virtualized cloud under changing environment. To the best of our knowledge, our work is unique in considering the detection problem under changing environment in virtualized clouds.
- Proposing the monitoring and quantification of the effect of performing resources adjustments, which enhances the accuracy of identifying DoS attacks under changing environments.
- Proposing a model to correlate VMs metrics with the actual resources load by the host, which enables the hypervisor to identify compromised VMs.
- Modeling an incentive technique that enables the hypervisor to give incentives in the form of resources to the VMs that have truthfully declared their metrics and punish these VMs that lied about their actual metrics.

Paper outline

The rest of the paper is organized as follows. In "Related work" section, we discuss the related work. In "The proposed framework" section, we present the proposed framework. "Security analysis of the proposed framework" section presents security analysis of the proposed framework. In "Experimental results and analysis" section, we present our empirical results. Finally, "Conclusion" section concludes the paper.

Related work

Machine learning for detecting DoS attacks in the cloud was used by several researchers. This work benefits from many advanced machine learning and and artificial intelligence techniques to predict the status of VMs (i.e, malicious or normal). Lonea et al. [13] uses the normal traffic pattern received from the Virtual Bridge (VB) of the VM to validate for consistency against behavioral patterns of attacks. They use a network intrusion detection system, that analyzes the normal traffic flow obtained from the VB, to check and test for consistency against the attack behavioral patterns. Thus, if abnormal traffic has been detected, the anomaly information will be reported and an alarm generated.

Similarly, Gupta et al. [6] propose a profile based network intrusion detection system. They combine both fine grained data analysis and Bayesian techniques in order to detect TCP SYN flooding. The main advantage of these approaches is the ability to identify DoS symptoms at an early stage, because their approaches are able to collect information at the networking level, before the DoS causes a significant performance degradation. However, the lack of application performance information can result in wrongly identifying high traffic during the peak time as a DoS attack.

Ficco et al. [14] propose a strategy for generating stealthy DoS attacks in the cloud, which uses low overhead attacks to inflict the maximum financial cost to the cloud clients [14]. Masood et al. [7] propose a web-behavior-based detection, where they identify two client's profiles. The first one is for good clients while the second one is for bad clients. A good client will follow a pattern that reflects normal activity on the web, while a bad client will show some abnormal activities. Similarly, Anusha et al. [8] study the behavior of normal users of Web applications. They assume that an attacker spends a very short time (almost zero) over a Web page. They use for that a metric called Time Spent on a Page (TSP). They assume that the attackers TSP is very close to zero. In contrast, the TSP of a normal client should be high enough to interact with the Web page. The work of Kwon et al. [9] also uses a behavioral approach for detection. They start from a tested assumption saying that the behavioral patterns of normal traffic are similar, while the behavior patterns of malicious traffic are not. The cosine similarity is used to check the similarity of the traffic. If such a similarity does not exist, an alarm is generated. A main advantage of this approach that it is able to determine the similarity during run-time. However, there is no guarantee that the normal traffic will always be similar in a dynamic environment (i.e. cloud). In fact, in some applications, we could have many forms of normal traffic that are fairly dissimilar.

Palmieri et al. [15] use a two-phase ML-based detection technique. The first phase is called Blind Source Separation (BSS), while the second phase is called Rule-based Classifier to detect zero-day attacks that change or alter the traffic volume rate. BSS extracts the features of the cloud nodes traffic in order to be used by a decision tree classifier to create a normal traffic profile (baseline). Most recently, Choi et al. [16] propose a data-mining-based approach to detect application layer HTTP GET DoS attacks. They use a normal behavior pattern to detect DoS attacks on VMs. The parameters used for analyzing and creating attack patterns are: CPU usage, packet size and packet header information. They evaluate their approach by comparing it with a signature-based approach. The result showed that their proposed method performs better than SNORT in terms of identifying new attack profiles. Similar to this work, Jeyanthi and Mogankumar [17] and Jeyanthi et al. [18] propose a mechanism to detect DDoS attacks based on clients request rate. Clients requests will be put in a black list or white list based on a certain threshold rate. The threshold is determined by calculating the maximum number of legitimate client requests. The

authors have shown experimentally that the legitimate clients could continue being served during an attack using their method. However, the main disadvantage of this method is that it is threshold based, where setting the optimal threshold is always difficult and infeasible in a production cloud environment.

Chonka and Abawajy [19] and Chonka et al. [20] propose a decision tree classification technique. The method operates in two phases: training phase (first phase) and testing phase (second phase). In the training phase, a rule set that has been generated over time by the decision tree classifier is used to define both known and unknown attributes. In the testing phase, a decision making module is used to decide the likelihood of a previously classified packet. This helps decide whether to let a packet enter or not. Similar to that, Lonea et al. [13] also proposed a classification technique based on Intrusion detection system (IDS). The detection module analyse the alerts generated by each VM using the Dempster-Shaferther theory (quantitative solution classifier) in 3-valued logic and fault tree analysis (FTA). Although Dempster-Shaferther is able to produce powerful results when observations about attacks come from different sources, it becomes unsuitable when one source produces multiple observations [21].

Among other approaches, the work of Iyengare et al. [22] proposes a Multilevel Thrust Filtration (MTF) that contains four detection and prevention modules, which are traffic analysis, anomaly detection, anomaly classification, and attack prevention. The proposed method filters the incoming packets and detects four types of traffic congestion, which are spoofing, ash crowd, DDoS, and aggressive legitimate traffic. A similar approach has been proposed by Jeyanthi and Iyengar [18]. The main feature of this approach is the ability to increase the attack detection accuracy because multiple stages of detection are used. However, it is associated with a significant overhead since multiple algorithms and techniques should be used.

Cooperative IDSs in cloud have been proposed in several works. For example, Teng et al. [23] propose an approach that aggregates two different detectors: feature and statistical detectors. The feature detector adopts SNORT to separate events based on Transmission Control Protocol (TCP). The statistical detector cooperates with SNORT by using data packets from it to find whether an event is an attack or not. If the rate of packets obtained exceeds the predefined threshold, this means that there is an actual attack. Similarly, Man and Huh [24] and Singh et al. [25] propose a cooperative IDS between different cloud regions. Their approach enables exchanging alerts from multiple places (detectors). The proposed approach allows the exchange of security information between interconnected clouds. Ghribi [26] proposes a middleware IDS. The approach allows a cooperation between three layers: Hypervisor-based, Network-based, and VM-based IDS. If an attack was found in a layer, it cannot be executed in the other layers. Chiba et al. [27] also propose a network-based cooperative IDS to identify network attacks in the cloud environment, which is performed by monitoring traffic while maintaining performance and service quality. Recently, Wahab et al. [28, 29] propose a game theoretic-based IDS. The approach that they used enables a CP to optimally distribute its resources among VMs in such away to maximize the detection of distributed attacks. The main limitation of the cooperative IDS is that they work in the assumption that all nodes are trustable, which makes it vulnerable to malicious insiders.

Cloud-based DoS attacks mitigation approaches are also proposed in several works. For example, Yu et al. [30] propose a dynamic resource allocation feature for VMs. This allows attacked VMs to acquire extra resources during DoS attacks. When a DDoS attack occurs, the cloud hires the idle resources to clone sufficient attack prevention servers for the attacked VMs in order to guarantee the quality of service for the users by filtering out attack packets. This approach is beneficial in an environment where DoS attacks are frequently generated. However, it can be exploited by selfish VMs to acquire and use much resources even though there is no attack. Also, Somani et al. [31] propose auto-scaling decisions by differentiating between legitimate and attack traffic. Attack traffic is detected based on the workload of human behavior. The advantage of this approach is that it gives more attention to serve legitimate clients by making accurate and proper autoscaling decisions. However, a workload of human behavior can be emulated. This makes an attacker able to deplete cloud resources.

The summary of the existing works is given in Table 3. The aforementioned works paved the way to understand the issues behind improving the detection accuracy in cloud environments. Our proposed model offers two major features. The first is the aspect of detection under changing environment. To the best of our knowledge, our work is the first to consider the detection problem under changing environment in virtualized clouds. The second is allowing VMs to share information about their current application metrics (e.g, number of clients, requests and sales) to the hypervisor, which in turn allows distinguishing between legitimate high load and DoS attacks. It also enables the hypervisor to identify the compromised VMs that try to claim and consume more resources.

The proposed framework

In this section, we describe the key constituents of our framework. The framework contains a set of components, each of which exhibits a set of modules. Figure 1 illustrates the framework structure and describes the modules of each component. These components include: data gathering, data and load analysis, and detection.

Table 3 Cloud-based detection approaches

Researchers	Approach
Lonea et al. [13]	Profile-based detection
Gupta et al. [6]	Profile-based network detection
Masood et al. [7]	Web-behavior-based detection
Anusha et al. [8]	Web-behavior-based detection
Kwon et al. [9]	Web-behavior-based detection
Palmieri et al. [15]	Machine learning-based detection
Choi et al. [16]	Data mining-based detection
Jeyanthi and Mogankumar [17]	Data mining-based detection
Jeyanthi et al. [18]	Data mining-based detection
Iyengare et al. [22]	Multiple stage detection
Jeyanthi and Iyengar [18]	Multiple stage detection
Teng et al. [23]	Cooperative-based detection
Man and Huh [24]	Cooperative-based detection
Singh et al. [25]	Cooperative-based detection
Chribi et al. [26]	Cooperative-based detection
Wahab et al. [28, 29]	Game theocratic-based detection

We use the Linux Trace Toolkit next generation (LTTng) [32] to gather the VMs performance metrics. LTTng is a powerful, low impact and lightweight [33] open source Linux tracing tool. It provides precise and detailed information on the underlying kernel and user-space executions. LTTng contains different trace points in various modules of the operating system kernel. Once a predefined trace point is reached, it generates an event containing a time-stamp, CPU number and other run-time information related to the running processes.

Data analysis

This component is responsible for analysing data obtained from the data gathering component. We divide this component into the four modules: Trace abstractor, preparing dataset for normal and anomaly activities, training and create SVM classifier, and resources adjustment analysis.

Trace abstractor

The trace file size is usually so large that it is difficult to analyze and understand the system execution. Most of the time, another analysis tool is required to abstract the low-level events and represent them as higher-level events, thus reducing the data to be analyzed. Trace abstraction is typically required to compute statistics of complex system metrics that are not directly computable from the low-level trace events [34]. For instance, to compute synthetic metrics such as "number of HTTP connections", "CPU usage", and "number of different types of system and network attacks", raw events must be aggregated to generate high-level events. Then, the desired metrics must be extracted and computed. Table 4 gives examples of a higher-level event generated from low-level events. The details of the trace abstraction tool used

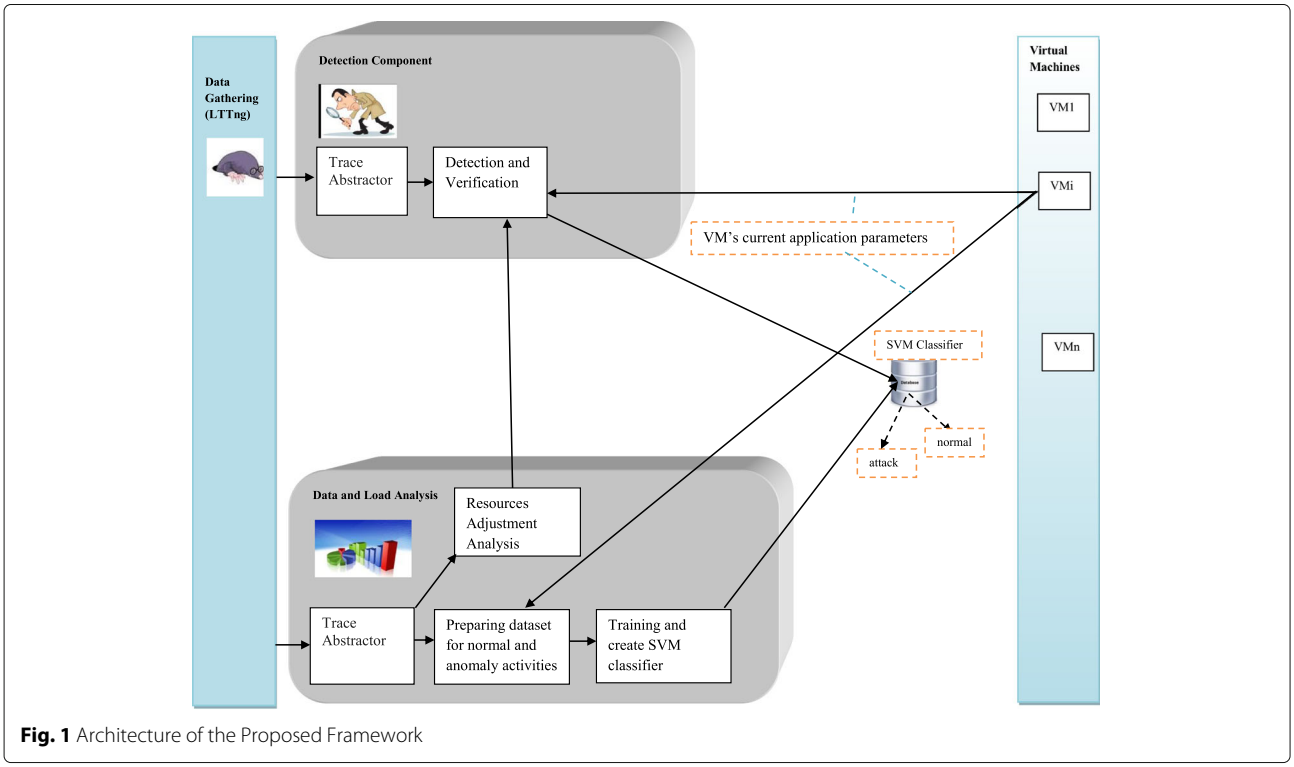


Fig. 1 Architecture of the Proposed Framework

Table 4 Abstracting example

Low-level events	Higher-level event
Socket create	HTTP connection
Socket call	
Socket send	
Socket receive	
Socket close	

to generate such high-level meaningful events can be found in [33].

Preprocessing and training modules

In this phase, the SVM [35] classification technique is used to analyse the collected data and classify the VMs load. SVM is a classification technique that employs a nonlinear mapping to convert the original data into higher-dimensional data in order to find a hyperplane that best separates the training tuples based on their classes. The hyperplane is determined using support vectors and margins in such a way that maximizes the hyperplane's margins with the aim of delivering more accurate results when classifying future data tuples [12]. We use SVM thanks to its ability to generate very accurate classifiers (especially in binary classifications) [35] and its effectiveness in high dimensional datasets consisting of a large number of attributes [36]. Moreover, it is robust against outliers and overfitting [37, 38].

The training dataset is generated during the monitoring of the host to determine which metrics reflect the malicious behavior and which ones reflect the normal behavior. As shown in the proposed architecture (Fig. 1), VMs are allowed to share their application/business metrics to be considered among the features used to train the SVM. The VM application/business metrics are discussed later in the previous section.

Each VM can be either under DoS attack or normal. Thus, class label $y_i \in \{\text{attack, normal}\}$. Given the training datasets $(x_i, y_i)...(x_n, y_n)$, x_i is the VM metrics values used for the training. n is the number of metrics values, the objective is to find the hyperline that offers a maximum margin (Fig. 2) such that:

$$w * x + b = 0 \tag{1}$$

Where w is a weight vector (Eq. 6) and b is a threshold.

Thus, the training data should satisfy:

$$w * x_i + b \geq -1 \; for \; all \; attack \; data \; x_i \tag{2}$$

$$w * x_i + b \leq +1 \; for \; all \; normal \; data \; x_i \tag{3}$$

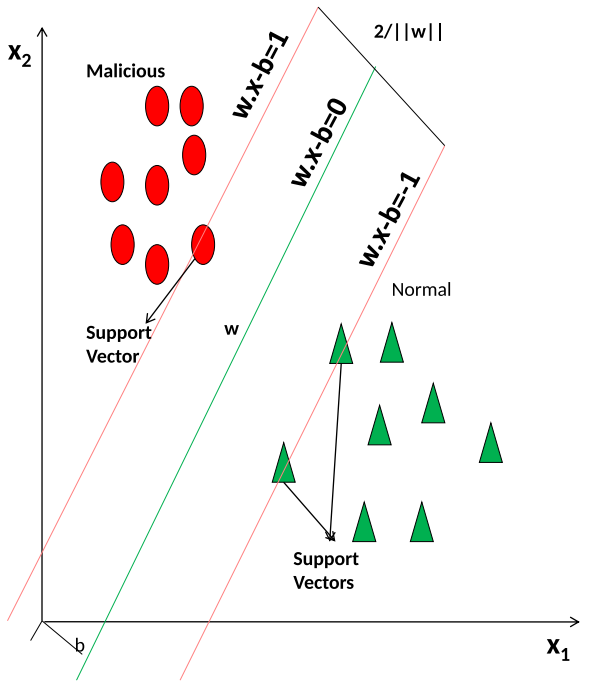


Fig. 2 The value of w affects the position of the hyperplane

The problem is converted to finding the optimal hyperplane (Eq. 1), which can be turned into a convex optimization problem [12]:

$$\left\{ \begin{array}{l} min \; \tau(w) = \dfrac{\|w\|}{2} \\ Subject \; to \; y_i(\langle w, x_i \rangle + b) \; for \; all \; i = 1, ..., n \end{array} \right\} \tag{4}$$

The convex optimization can be solved using Lagrange multipliers [12]:

$$\left\{ \begin{array}{l} maximize \; L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j K\left(x_i, x_j\right) \\ Subject \; to \; \sum_{i=1}^{n} y_i \alpha_i = 0 and 0 \leq \alpha_i \leq C \forall 1 \leq i \leq n \end{array} \right\} \tag{5}$$

where α_i is the Lagrange multipliers [12], $K(x_i, x_j)$ represents the kernel function (e.g., linear, polynomial, etc.) and C is a constant for determining the trade-off between margin maximization and training error minimization [12].

By solving Eq. 5 we get [12]:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{6}$$

Finally, the decision attack function is given by:

$$f(x, \alpha, b) = \{\pm 1\} = sgn\left(\sum_{i=1}^{n} y_i \alpha_i K(x, x_i) + b\right) \quad (7)$$

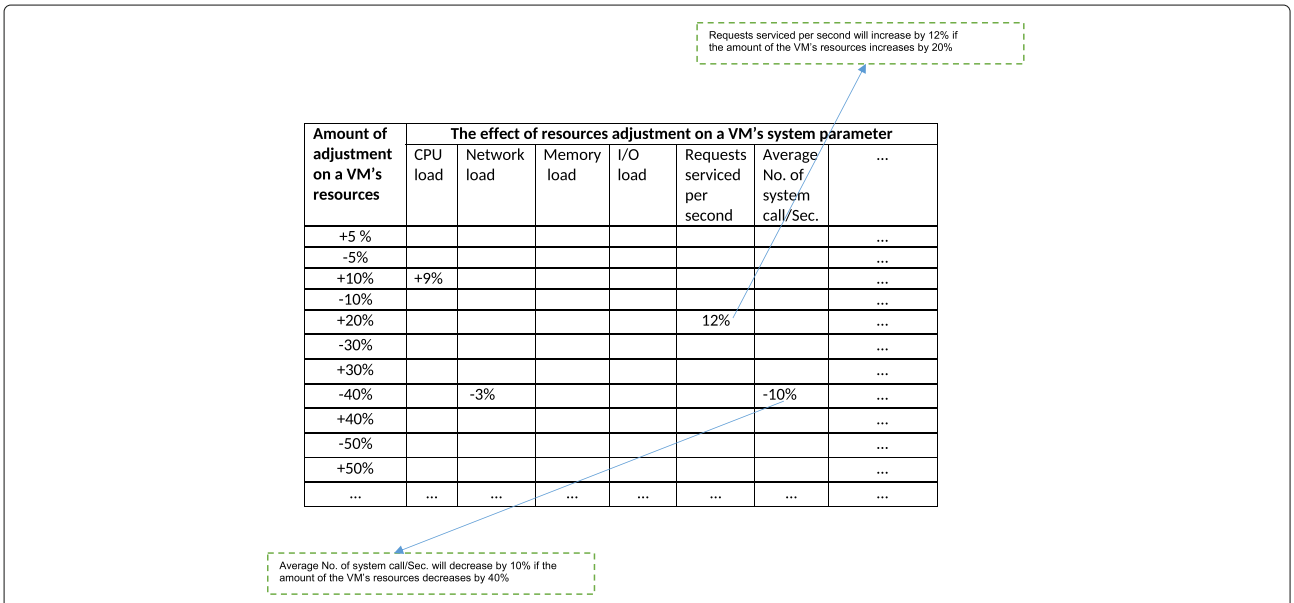
Resources adjustment analysis

The impact of resources adjustment appears on the collected data, which makes the SVM classifier unsuitable in the light of the new adjustment in the VMs resources. In other words, the collected data will be affected by the new adjustments, which leads to an inaccurate classification of that data. To address this issue, we should determine the effect of resources adjustment on the collected data (we will discuss later in this section how to calculate the effect of resources adjustments). The effect of resources adjustment reflects to what extent the modified data (after new resources adjustment) deviates from the original data (the data that meets the basic infrastructure).

Having the effect of resources adjustment, two approaches to solve the detection problem under changing environment can be considered. The first approach is to consider this effect during the training of the SVM classifier. This can be done by generating new sub features for every feature (features represent system metrics in this context) used in the training. In fact, the new sub features are the result of applying the effect of resources adjustment on the basic feature. For example, if the original feature used to train the SVM classifier is 10, 20, 30, 40 for CPU, memory, I/O, and network, respectively, then the SVM classifier is also trained to classify in the presence of 50% adjustment on the VM resources by adding to the training set the following new sub feature: (10 + 10 * 50%), (20 + 20 * 50%), (30 + 30 * 50%) and (40 + 40 * 50%) for CPU, memory, I/O, and network respectively. This makes the SVM classifier be trained not only on the original infrastructure but also on the new infrastructure after resources adjustment.

The second approach is to account for the resources adjustments effect by a separate filter. The filter is used as a preprocessing step, prior to classification, to get rid of the effect of resources adjustment on the collected data, in order to normalise the data with respect to the original infrastructure on which the training was performed, before passing it to the SVM classifier. We adopt the second approach in the proposed framework for the two following reasons. On the one hand, the first approach requires generating a huge training dataset, since we have to generate many sub-features for each single feature. This results in more overhead during the SVM training. The second approach does not require any change in the dataset. On the other hand, training an SVM with all possible adjustments (as is the case for the first approach) may lead to an overfitting. Specifically, the classifier might work correctly in the presence of the trained adjustment; However, the classifier accuracy will go down in the absence of such adjustments.

The effect of resources adjustment. The effect of resources adjustment on VMs is studied during the running of the VMs to find out to what extent their system metrics are affected by the adjustments. Although such a process requires changing the resources of the VMs which in turn may affect the performance of the application running inside these VMs, the impact of an adjusting and



Amount of adjustment on a VM's resources	The effect of resources adjustment on a VM's system parameter						
	CPU load	Network load	Memory load	I/O load	Requests serviced per second	Average No. of system call/Sec.	...
+5 %							...
-5%							...
+10%	+9%						...
-10%							...
+20%					12%		...
-30%							...
+30%							...
-40%		-3%				-10%	...
+40%							...
-50%							...
+50%							...
...	...	...	...	...	...	...	...

Fig. 3 Table used for filtering out the effect of resources adjustments on a VM system metrics

monitoring the effect is acceptable since it is done during a short time period ($\approx$ the time needed to capture VMs system metrics). Note that resources adjustment process can be done by exploring and monitoring the effect of all possible resources adjustments performed on the system metrics of the VMs. More specifically, we maintain a filter of resources adjustments effect (as in Fig. 3), which is used as a preprocessing step prior to classification (i.e., SVM), to filter out the effect of resources adjustments from the collected data. In other words, this helps to get rid of the noise that may show up on the collected data (due to the new adjustments) and may considerably decrease the accuracy of the detection. Algorithm 1 is used to determine the effect of all possible resources adjustments on the system metrics of the VMs.

Algorithm 1: Determining the effect resources adjustments on VMs system metrics.

Input : List of VMs' possible resources adjustments *adj*[]
Output: The effect of resources adjustment on VMs system metrics $\{eff_1[][], eff_2[][], \ldots, eff_n[][]\}$

1 **repeat**
2 **foreach** *Running VM j* **do**
3 Monitor and store *j*' system metrics in array U_1 (before adjustments)
4 **foreach** *Adjustment adj* $\in$ *adj []* **do**
5 Apply an adjustment of amount *adj* on *j*'s resources
6 Monitor and store *j*' system metrics in array U_2
7 Remove the adjustment of amount *adj* and restore *j*'s default resources
8 **foreach** *index i of* U_1 **do**
9 $BeforeAdj = U_1[i]$
10 $AfterAdj = U_2[i]$
11 **if** $|BeforeAdj - AfterAdj| < \epsilon$ **then**
12 $eff_j[adj][i] = 0$
13 **else if** $BeforeAdj < AfterAdj$ **then**
14 $eff_j[adj][i] = \frac{adj*100}{AfterAdj} * BeforeAdj$
15 **else**
16 $eff_j[adj][i] = - \frac{adj*100}{AfterAdj} * BeforeAdj$
17 **end**
18 **end**
19 **end**
20 **end**
21 **until** ε *elapses*;

In Algorithm 1, for each VM^j ($j \in$ VMs) in a certain host, the algorithm monitors and determines the current system metrics of j to be stored in array U_1 [] (line 3). Then, for each possible resources adjustment *adj*, the algorithm applies resources adjustment of value *adj*, to see the effect of *adj* on *j*'s system metrics. The list of all possible resources adjustments, which is given as an input in Algorithm 1, can be selected manually by the cloud administrator. The administrator can consider these adjustments that have significant impacts on VMs' system metrics and also had been requested in the past by the client according to the pay-as-you go business model. This, in turn reduces unnecessary study of unneeded adjustments and thus decreases the processing time of Algorithm 1. Note that we apply adjustments on VMs resources using control groups (cgroups) [11], which is a Linux Kernel feature that limits and allocates resources to VMs — such as CPU time, system memory, network bandwidth, or combinations of these resources. The system metrics are re-computed after each adjustment process (line 6). Then, the algorithm computes the effect of an adjustment *adj* on VM *j*'s system metrics by finding the percentage of change on the *j*'s system metrics (line 8 - 16). If the new calculated metric U_2 [i]'s value is within a small range of the old value $U_1[i]$, the effect will be considered as 0. This is described in the Algorithm 1 as $eff_j[adj][i] = 0$ (line 12), which means that the effect of an adjustment *adj* on that i-th metric of VM j is equal to 0. This indicates that the resources adjustment had no effect on that given metric. However, if the new calculated metric's value considerably differs from the old one, the algorithm computes the percentage of change on the old value such that: $eff_j[adj][i] = \frac{adj*100}{AfterAdj} * BeforeAdj$ (line 14) for a positive change (i.e., new-value > old-value) and -$\frac{adj*100}{AfterAdj} * BeforeAdj$ (line 16) for a negative change (i.e., new-value < old-value). The algorithm is used for each possible adjustment in order to have a filter of resources adjustments effect for each VM. The filter is then used during the detection step (Detection component Section) to filter out the collected data from the effect of resources adjustment. This adapts the environment of the detection to the original environment (i.e., the environment in which the SVM classifier was created). Note that the whole process is repeated periodically after a certain fixed period of time ε (line 17) to capture the new effect of the given resources adjustment on VMs system metrics.

It should also be noticed that some resources adjustment may have no impact on the system metrics of the VMs. In fact, it can depend on resources and the underlying usage model. For example, for a given load, if we consume 60% of the CPU (100% available with no restriction), when a restriction of 70% is imposed, we will consume 60% / 70% = 85% of the available CPU. The number of CPU seconds will remain the same though. For this purpose, we have considered in the resource adjustment algorithm the following condition: If ($|BeforeAdj -$

$|AfterAdj| < \epsilon$) (line 11), which means that the collected data are the same before and after adjustments. The effect of resources adjustment in this case is equals to 0, as given in Algorithm 1 ($eff_j[adj][i] = 0$).

Detection component

This component is used for identifying DoS attacks. It monitors the system and performs tracing abstraction, similar to the steps used in "Data analysis" section and "Preprocessing and training modules" section. Along with these steps, the module performs the detection algorithm described in Algorithm 2. The following section presents the details of this component.

Detection algorithm

Algorithm 2 that is used for detecting DoS attacks works as follows. For each VM^j ($j \in$ VMs) in a certain host, the algorithm uses the model obtained from the previous section to calculate the VMs resources load (calculated load) with respect to the declared metrics of the VMs (line 4). This step is important to minimise unnecessary false positive alarms during flash events. The calculated resources load is compared with the actual resources load to determine if the calculated resources load is within a small range of the actual resources load (line 5). If this is not the case, the detection process starts by filtering out the effect of resources adjustment on *j*'s system metrics using the resources adjustment effect of *j*(given as input in Algorithm 2) (i.e., $newU[i] = U[i] \pm (U[i] * eff_j[VMA^j][i])$ (line 6-10) from the collected data. The objective is to adjust the data for the original infrastructure on which the training was performed before passing it to the SVM classifier. The Algorithm passes then the modified collected data to SVM to predict the result (line 11). If the result $r =$ "attack", the algorithm identifies a DoS attack (line 12-13). If the calculated resources load is within a short distance of the actual load, the algorithm identifies the resources load as normal (line 16-17).

The main complexity of the proposed approach lies in the SVM training, which is commonly known to be $O\left(n^3\right)$ [38], where n represents the training set size. Although this might be infeasible for very large datasets, the training process is performed only once, and its overhead can then be neglected [39, 40]. Moreover, recently, more and more techniques are being proposed for efficient SVM training [41–43]. As for the prediction process in Algorithm 2, the complexity lies in three parts. In the first part, the algorithm filters out the effect of resources adjustment from the collected data (line 6-10). The computation complexity for this part is $O(n)$, where n is the number of the input metrics of SVM. The second part is to find if the predicted and calculated VM resources load are similar (line 5), which is of constant time complexity $O(1)$. The last part of the algorithm is to predict the modified collected data using SVM (line 11). The computational complexity of SVM-based prediction is $O(n)$, where n is also the number of the input metrics of SVM. Therefore, the overall computation complexity of the proposed detection Algorithm is $O(n) + O(n) + O(1) \approx O(n)$.

Algorithm 2: Detection Algorithm

Input : VMs' alpha values $\{\alpha_1, \alpha_2, ..., \alpha_n\}$
Input : VMs' beta values $\{\beta_1, \beta_2, ..., \beta_n\}$
Input : VMs' declared application metrics $VMT^j \quad \forall\, j \in VMs$
Input : VMs' effect of resources adjustment on their system metrics $\{eff_1[\,][\,], eff_2[\,][\,], \ ... \ , eff_n[\,][\,]\}$
Input : Amount of adjustment applied to VMs resources $VMA^j \quad \forall\, j \in VMs$
Output: Attack Boolean *Dec*

```
1  foreach VM j do
2      Determine j's current resources load crt_load.
3      Monitor and store j's system metrics in array U.
4      calc_load= α_j + β_j * VMT^j.
5      if |calc_load − crt_load| > ϵ then
6          foreach index i of U do
7              if eff_j[VMA^j][i] ≥0 then
8                  newU[i] = U[i] - (U[i] * eff_j[VMA^j][i])
9              else
10                 newU[i] = U[i] + (U[i] * eff_j[VMA^j][i])
11             end
12         end
           /* classifying data (i.e., newU
              using the SVM                  */
13         r = predict(newU, SVM)
14         if r=="attack" then
15             Dec = True
16         end
17         else
18             Dec = False
19         end
20     end
21 end
22 else
23     Dec = False
24 end
```

Verification and resources allocation

The proposed detection framework allows a VM to regularly declare its current application metrics, which enables the hypervisor to correlate these metrics with the actual resources load and decide if it is coherent or not (compromised VM trying to claim and consume more resources). In fact, the VM may have no incentive to declare its metrics. Moreover, the VM may lie about its current metrics either because of its selfish strategy (in order to obtain more resources) or because the VM has been compromised.

Algorithm 3: Verification and Resources Allocation Algorithm

Initialisation:
Input : VMs' alpha values $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$
Input : VMs' beta values $\{\beta_1, \beta_2, \ldots, \beta_n\}$
Input : VMs' declared application metrics $VMT^j \quad \forall\, j \in VMs$
Output: Amount of resources granted/revoked to/from each VM

1 **foreach** *VM j* **do**
2 Determine *j*'s current resources load *crt_load*
3 *calc_load*= $\alpha_j + \beta_j * VMT^j$
4 **if** $|calc\_load - crt\_load| < \epsilon$ **then**
5 Grant resources to *j*
6 **end**
7 **else**
8 Revoke resources from *j*
9 **end**
10 **end**

To address the aforementioned problems, we propose a verification algorithm (Algorithm 3). Our solution motivates the VM to declare its current application metrics by granting resources to the VM whose calculated resources load (obtained from the VM's declared metrics) falls within a close range of the VM's actual resources load (line 4-5). On the other hand, the hypervisor revokes resources from the VM whose calculated resources load and actual resources load do not match (line 6-7). This dissimilarity, in most cases, is either because the VM has lied about its declared metrics, or because the VM has been compromised. The amount of resources revoked from the VM can be decided by the system administrator, who clearly knows the real impact of adjusting the VM's resources on their performance. However, we suggest that the amount be proportional to the magnitude of the difference between the calculated resources load and the current resources load. In other words, the larger the difference between the calculated resources load and current resources load is, the more resources should be revoked from the VM. This encourages VMs to truthfully declare their metrics used to calculate the resources load.

Security analysis of the proposed framework

The main objective of the proposed framework is to enhance the detection of DoS attacks under changing environment. In this section, we analyse the effectiveness of our framework in the presence of flash events, DoS attacks and compromised VMs.

Flash events

A flash event occurs when there is an unusual surge of legitimate traffic. Our model is able to distinguish between a flash event and DoS attacks since our framework allows VMs to declare their current application metrics (e.g., number of clients) and motivates them to do that by granting them extra resources (Algorithm 3). The declared metrics can represent flash events. The declared metrics are then used to calculate the resources load according to the model in the previous section. The calculated and actual resources load are then compared to see if they approximately match. If so, the hypervisor will know and understand that the VM is under an unusual surge of legitimate requests and will grant the VM more resources to serve better during this period. Otherwise, if the calculated and actual resources load are largely different, the hypervisor revokes some resources from the VM. This strategy motivates the VM to truthfully declare its peak load and limit the illegal use of resources by the attacker, in case the VM has been compromised.

DoS attacks

The main purpose of the proposed framework is to detect DoS attacks. Our model achieves this by using SVM. The framework trains SVM classifiers on the normal and malicious features to achieve the learning of the classifier. Our framework then monitors the incoming features and predicts the system status. Moreover, the proposed detection algorithm supports the detection under a changing infrastructure. The algorithm checks if no modification in the VM's resources (i.e., granting or revoking resources) has occurred. If so, the algorithm passes the collected data directly to the SVM classifier, without applying any filtering strategy. Otherwise, the algorithm filters the effect of resources adjustments in order to adjust the collected data to the original infrastructure (on which the training was performed), before passing it to the SVM classifier. This is done by removing the effect of noise caused by resources adjustments.

Proposition: The accuracy of the detection will not be affected after filtering out the effect of resources adjustment.

Proof Consider that the collected data U has been affected by resources adjustment in such a way that makes U change by percentage $\%eff$. The $\%eff$ can be a positive (e.g., +5%), negative (e.g., -5%) or 0 (as the example given in Resources adjustment analysis section, where some adjustments (e.g., CPU) do not result in any change in the collected data) . If $\%eff$ is positive, the value of U changes to $U + (\%eff * U)$. In this case, our detection algorithm removes the adjustment (Algorithm 2 line 5-9) as follows:

$$U + (\%eff * U) - (\%eff * U) = U \tag{8}$$

On the other hand, if $\%eff$ is negative, the value of U changes to U - (%eff*U). In this case, the detection

algorithm removes the adjustment (As in Algorithm 2) as follows:

$$U - (\%eff * U) + (\%eff * U) = U \quad (9)$$

Also, if $\%eff = 0$, the value of U changes to U - $(0 * U) = U$. In this case, the detection algorithm removes the adjustment as follows:

$$U - (0 * U) + (0 * U) = U \quad (10)$$

In the aforementioned three situations, the collected data U can be recovered, which means that the detection will be performed as if no adjustment had been applied. □

Robustness against compromised VMs

The proposed framework is able to detect the compromised VMs that try to claim receiving an unusual load of client requests, to be allowed to consume more resources. The compromised VM can hide that its compromised by mimicking normal load and/or flash crowds [44]. The hypervisor calculates the VM load (resources load) based on the compromised VM's current declared application metrics and compares it with the actual resources load. If the calculated resources load is not within a short range of the actual resources load, there will be a high probability that the VM has been compromised. A possible strategy that a compromised VM may use, is trying to find α and β in order to obtain the model for calculating the resources load $load = \alpha + \beta * VM\_par$. This can be done by using different values of α and β. For every α and β, the compromised VM sees the response from the hypervisor (the response is the resources given for the unusual declared metrics). If the compromised VM did not receive a response, the compromised VM tries other values of α and β, until the correct α and β values are obtained (receiving a response from the hypervisor). Although this can be possible, the number of trials will be very high, which makes it infeasible for the attacker, since α and β can be any real number from a large interval. In addition, the attacker will typically not have the opportunity to do a large number of attempts in trying to guess α and β. After several wrong guesses (e.g., 10 wrong attempts), the hypervisor would consider the VM as a compromised and prevent it from using resources.

Experimental results and analysis

In this section, we first explain the experimental setup used to perform our experimentation and then study the performance of the proposed detection approach.

Experimental setup

To evaluate our model, we chose to create our custom test environment. We prefered to use our own materials (e.g., resources) instead of using rented resources from existing CPs (e.g., Amazon EC2) for the following three reasons: 1) Most of the CPs including Amazon EC2 have restriction rules regarding any security testing and evaluating on their resources and systems [5]. 2) All large CPs list DoS attacks' testing and evaluating as a non-permissible action [5]. 3) No CP allows its users with direct access to the host. Therefore, gathering information (i.e. performance information) is a quite difficult task. Our testbed consists of three machines. One machine is used as a virtual machine host and the other machines are used as client and attack emulator. All machines are attached directly to a Linksys 1000 Mb/s SOHO switch. Our test network is completely disconnected from the network of our institution as well as from the Internet to avoid the leakage of the DoS attacks. The detection algorithm (Algorithm 2) is implemented in Python and the BoNeSi program is used [45] to generate attack-level and normal traffic. We used BoNeSi as a traffic generation tool because it allows us to simulate floods from large-scale bot networks. Moreover, BoNeSi tries to avoid the generation of packets with easily identifiable patterns, which can be quickly filtered out [45]. The virtual machine host used in the experiment is an Intel Core i7-4790 CPU 3.60 GHz Processor with 16 GB RAM. We installed the Apache 2.2 Web Server on the targeted VMs. The network interface is at 1000 Mb/s. We chose KVM [46] as our hypervisor-based virtualization. Indeed, KVM runs on the unmodified Linux kernel and is thus compatible with the standard performance tracing tools (e.g., LTTng), unlike Xen.

In order to simulate a real-world DoS attack, the CAIDA "DDoS Attack 2007" dataset [47] has been used as a baseline for extracting the features required to simulate attack traffic. Since it is not possible to mine normal traffic data from CAIDA's dataset because it is collected at Darknet and has no normal traffic, we used another dataset to capture normal traffic. For this purpose, we used a traffic trace of a 5-minute non-flash-event activity before the first semi-final match of the 1998 FIFA World Cup' dataset [48]. Table 5 shows the traffic features and characteristics extracted from the CAIDA and 1998 FIFA World Cup' datasets (the same features used in [49]). We used then BoNeSi as a traffic generator to be run based on information given in Table 5.

Table 5 Attack and normal traffic features extracted from CAIDA and FIFA World Cup' datasets

Characteristic	DoS attacks	Normal traffic
Packet size	64k	64k
No. of sources	859	73
Packet rate	125,705	385

Training phase

During the generation of the attack and normal traffic using BoNeSi, we monitored the following metrics: CPU, memory, I/O and network load at different time intervals. We can also monitor and use high-level metrics, as illustrated in the trace abstractor section ("Detection algorithm" section). However, we prefer to use these relatively basic metrics as they are widely used to describe the anomaly caused by a DoS attack [3]. The length of each interval is 30 s. For each interval, we used LTTng to generate the trace data. To extract CPU, memory, I/O and network loads from the trace data, we used the LTTng-analyses packages [50], which are a set of executable analyses to extract and visualise monitoring data and metrics from LTTng kernel traces. We created a training dataset that contains these performance metrics. The dataset is used then to train the SVM classifier to be able to distinguish between normal behavior and DoS attacks of the VM. We train an SVM classifier on our dataset using the 10-fold cross-validation model. We used a linear kernel function as it is considered more efficient for real-time applications [51]. It enjoys as well faster training and classification. Moreover, with the linear function, less memory is required as compared to the non-linear kernels [51].

Testing phase

In order to test the proposed model in the presence of resources adjustments, we created a testing dataset that is suitable for each type of adjustments. To do this, we monitored the metrics (CPU, memory, I/O and network load) in the same way used to generate the training dataset. However, in this phase, we performed some resources adjustments on the VMs during data collection, in order to study the effectiveness of our proposed model in such a case. We used the API of libvirt that employs cgroups [10] to adjust and limit VMs resources. Using cgroups allows exploiting Linux Kernel features that limit and allocate resources to VMs — such as CPU time, system memory, network bandwidth, or combinations of these resources [11]. This is performed on the two types of traffic datasets: attack and normal traffic. Our detection algorithm (Algorithm 2) is then executed on these datasets. The detection algorithm applies the filter of resources adjustment effect (obtained from executing Algorithm 1) on the testing dataset before passing it to the SVM classifier. The length of time used to determine the effect of resources adjustment in Algorithm 1 is 30 seconds.

We used to evaluate the accuracy of the proposed model the false positive, false negative, attack detection and accuracy rate.

Accuracy Rate =

$$100\% \times \frac{Total\ of\ correctly\ classified\ tuples}{Total\ of\ tuples} \tag{11}$$

Attack Detection Rate =

$$100\% \times \frac{Total\ of\ attacks}{Total\ of\ detected\ attacks} \tag{12}$$

False Positive Rate=

$$100\% \times \frac{Total\ of\ misclassified\ tuples}{Total\ of\ normal\ tuples} \tag{13}$$

False Negative Rate =

$$100\% \times \frac{Total\ of\ misclassified\ tuples}{Total\ of\ attack\ tuples} \tag{14}$$

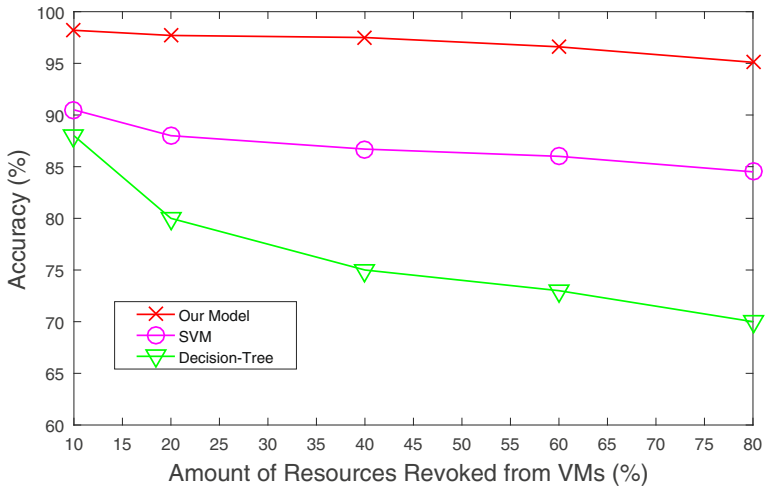


Fig. 4 Accuracy with respect to (w.r.t.) amount of revoked resources

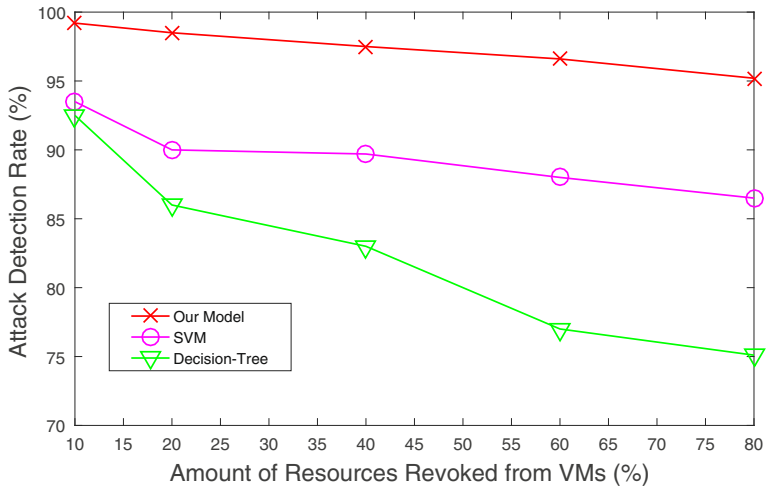


Fig. 5 Attack detection rate w.r.t. amount of revoked resources

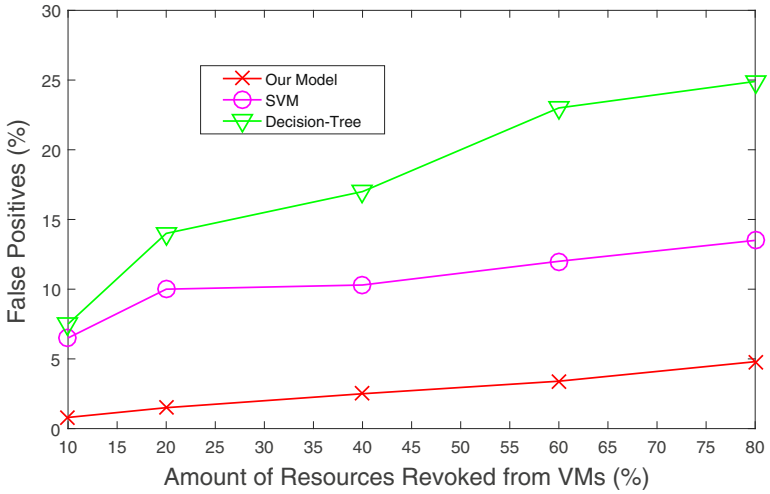


Fig. 6 False positive percentage w.r.t. amount of revoked resources

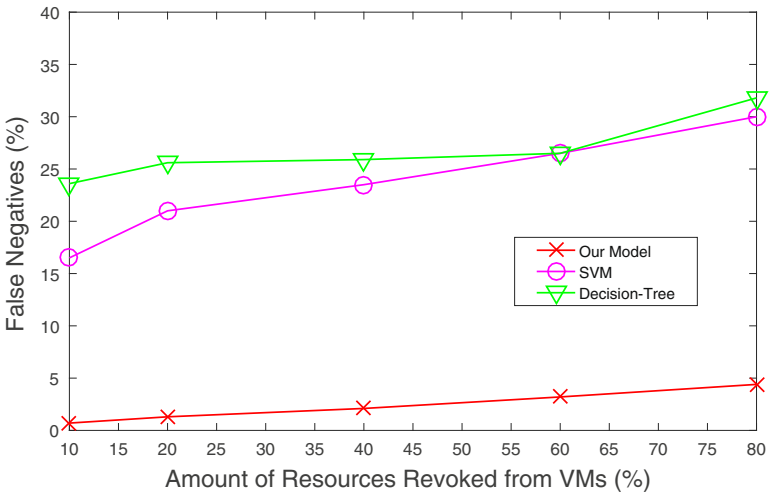


Fig. 7 False negative percentage w.r.t. amount of revoked resources

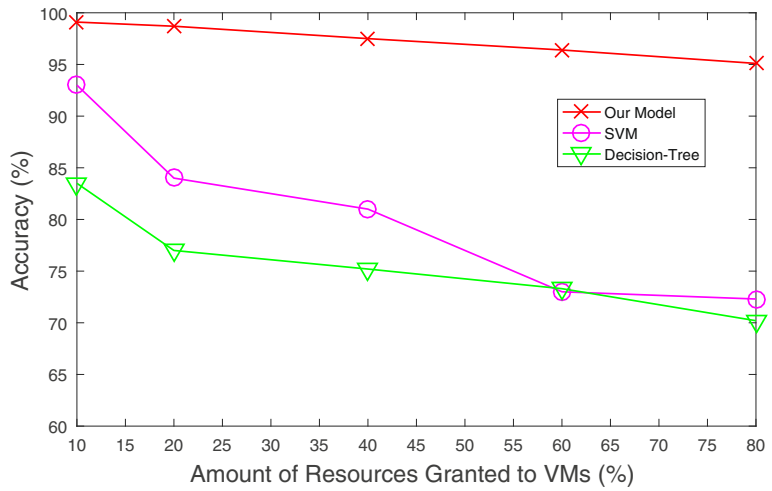


Fig. 8 Accuracy w.r.t. amount of granted resources

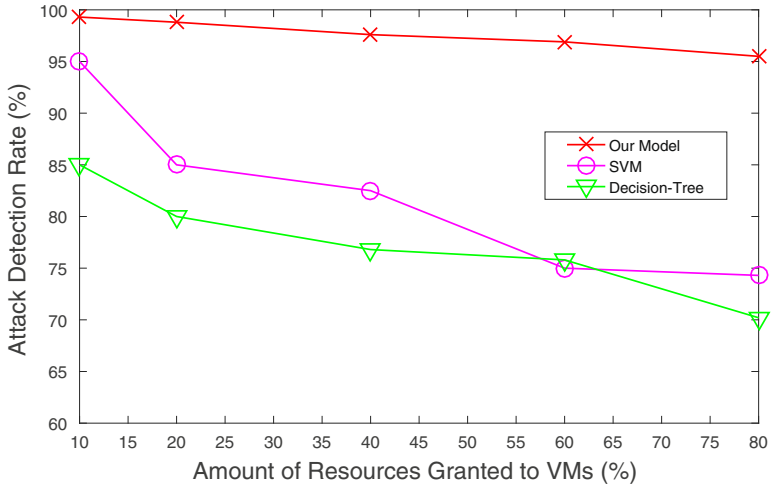


Fig. 9 Attack detection rate w.r.t. amount of granted resources

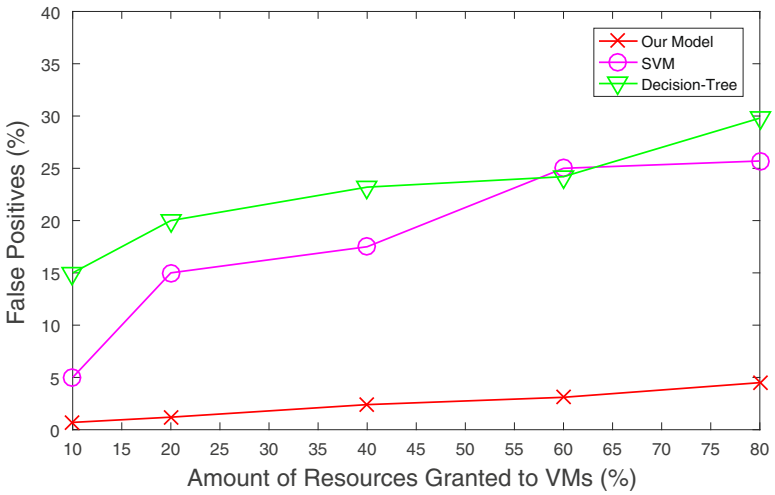


Fig. 10 False positive percentage w.r.t. amount of granted resources

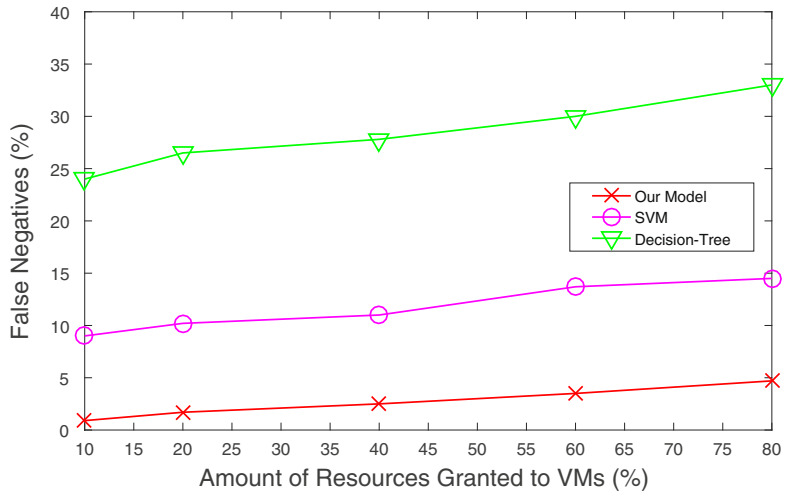


Fig. 11 False negative percentage w.r.t. amount of granted resources

The next section contains the results, compared to the traditional SVM and Decision-Tree detection [52] techniques. The traditional-SVM uses the SVM classifier directly, without applying any filtering strategy that can cope with the effect of the resources adjustment on the detection performance. Similarly, the Decision-Tree detection technique employs the Decision-Tree classification technique, without applying any filtering process. We train the traditional-SVM and Decision-Tree classifier on our dataset using the 10 fold cross-validation model.

Experimental results

We study in Figs. 4, 5, 6, and 7 the performance of our framework with respect to the amount of resources revoked from VMs. The results reveal that our framework is resilient to the decrease in the VMs resources. More specifically, Figs. 4 and 5 show respectively that the average accuracy and attack detection rates obtained by the proposed model at different percentages of revoked resources (from 10 to 80%) are 97.02 and 97.4%. These results are better than the results obtained using the traditional-SVM (87.14% for accuracy and 89.54% for attack detection rate) and Decision-Tree (75.44% for the accuracy and 79.04% for attack detection rate). As for the false alarms, Figs. 6 and 7 show respectively that the false positive and false negative rates obtained using our model at different percentages of revoked resources (from 10 to 80%) are 2.6 and 2.34%. These results are also better than the results obtained using traditional-SVM (10.46% for the false positive and 23.5% for the false negative) and Decision-Tree (17.28% for the false positive and 26.68% for the false negative).

Moreover, Figs. 8, 9, 10, and 11 study the performance of our framework with respect to the amount of resources granted to VMs. These results reveal that our framework is also resilient to the increase in VMs resources. Figures 8 and 9 show respectively that the average accuracy and attack detection rate obtained by the proposed model at different percentages of granted resources (from 10 to 80%) are 97.36 and 97.62%. These results are better than the results obtained using the traditional-SVM (80.66% for accuracy and 82.36% for attack detection rate) and Decision-Tree (75.84% for the accuracy and 77.56% for attack detection rate). As for the false alarms, Figs. 10 and 11 show respectively that the false positive and false negative rates achieved by the proposed model at different

Table 6 Amount of accuracy preserved by our model when revoking resources from VMs

Resources revoked from VMs	Accuracy preserved
10%	98.79%
20%	98.28%
40%	98.08%
60%	97.18%
80%	95.67%
Average	97.60%

Table 7 Amount of accuracy preserved by our model when granting resources to VMs

Resources granted to VMs	Accuracy preserved
10%	99.69%
20%	99.29%
40%	98.08%
60%	96.98%
80%	95.77%
Average	97.96%

Table 8 Kernel functions comparison using the proposed detection approach

Kernel function	Performance metric			
	Accuracy (%)	Attack detection rate (%)	False positive rate (%)	False negative rate (%)
Linear kernel	97.19	97.51	2.49	2.50
Multilayer percepton kernel	97.03	97.09	2.90	2.92
Quadratic kernel	97.54	97.30	2.69	2.71
Polynomial kernel	97.51	98.23	1.99	2.11
Gaussian kernel	96.98	98.05	1.94	2.09

percentages of revoked resources (from 10 to 80%) are 2.38 and 2.66%. These results are also better than the results obtained using the traditional-SVM (17.64% for the false positive and 11.68% for the false negative) and Decision-Tree (22.44% for the false positive and 28.26% for the false negative).

The reason why our model performs better than the traditional-SVM and Decision-Tree approaches is that the proposed model takes into account the resources adjustments that occur in the VMs infrastructure. The detection algorithm (Algorithm 2) filters (using the filter obtained from Algorithm 1) the effect of resources adjustments in order to make the collected data (testing dataset) cope with the original infrastructure (on which the training was performed) before passing it to the SVM classifier. This is done by removing the effect of resources adjustments (Algorithm 2, line 5-9). This is unlike the traditional-SVM and Decision-Tree techniques, where the effect of the resources adjustments is totally ignored. Therefore, their accuracy for detecting DoS attacks is affected.

We calculate the percentage of accuracy that can be preserved using our model under changing infrastructure. To do so, we run our model without adjustments applied to the VMs resources. The accuracy of the detection obtained without adjustment was 99.40%. This result is used as a baseline to calculate the percentage of accuracy that our model can preserve under adjustments. For this purpose, we determine the amount of accuracy preserved under different amounts of adjustments and calculate the average, as in Table 6 for revoking-based adjustments and Table 7 for granting-based adjustments. The results show that the percentage of accuracy that can be preserved is 97.60% for the revoking adjustments and 97.96% for the granting adjustments. This means that, by using our model, the accuracy got decreased under the effect of resources adjustments by only 1.79% for the revoking adjustments and 1.43% for the granting adjustments, which has no significant impact and can be neglected.

It should be noticed also that the SVM kernel used in the experiments is the linear kernel. However, our results will not significantly change if we use another non-linear kernel (e.g., Quadratic kernel). In fact, we tested our detection model using different kernels and by considering different values of resources adjustment (granting and revoking adjustments from 10 to 80%). Table 8 shows the performance metrics obtained (the average result has been selected) for the different non-linear kernels. It presents the comparisons between different non-linear kernels using the proposed detection model. The results show that there is no significant difference in the false positive, false negative, attack detection and accuracy rates between the different kernel functions.

Conclusion

We present an SVM-based framework for detecting DoS attacks in a virtualized cloud under changing infrastructure. Our solution collects some system metrics to train the SVM classifier to be able to distinguish between normal and malicious (i.e., a DoS attack) activities of the VM. The hypervisor then monitors and quantifies the effect of performing resources adjustments on the collected data. This information is then used to maintain a filter of resources adjustments effect. The filter is used as a preprocessing step prior to classification to get rid of the noise that may show up on the collected data, and that may considerably decrease the accuracy of the detection. Moreover, our solution motivates VMs to declare their current business metrics to the hypervisor, to enable the hypervisor to correlate these metrics with the actual resources load and decide if it is coherent or not. This increases the possibility of identifying compromised VMs, trying to claim and consume more resources. Experimental results show that our model performs better than the traditional-SVM and Decision-Tree approaches in the presence of infrastructure adjustments, in terms of false positive, false negative, attack detection and accuracy rate. The results also show that the percentage of accuracy that can be preserved under resources adjustments using our model is 97.60% for the revoking adjustments and 97.96% for the granting adjustments. Our results also show that the accuracy got decreased under the effect of resources adjustments by only 1.79% for the revoking adjustments and 1.43% for the granting adjustments, which has no significant impact and can then be neglected.

Acknowledgements
The financial support of the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

Authors' contributions
AB built the state of the art of the field, defined the objectives of this research, did the analysis of the current cloud-based detection approaches and their limitations. He implemented the approach presented in this paper, as well as the experiments. MB and MD initiated and supervised this research, lead and approved its scientific contribution, provided general input, reviewed the article and issued his approval for the final version. All authors read and approved the final manuscript.

Authors' information
Adel Abusitta is a Ph.D. student in Computer Engineering at Ecole Polytechnique de Montreal, Canada. He holds a M.Sc. degree in computer science from the University of Jordan, Jordan. The main topics of his current research activities are security in Cloud Computing, network security, and authorship analysis.
Martine Bellaiche is Assistant Professor in Department of Computer and Software engineering of Ecole Polytechnique of Montreal. She received a MSc in Computer Science from University of Montreal in 1985 and the Ph.D in Telecommunications from INRS in 2007. His research interests are in Network Security with special focus on attack, in Network Sensor Security and cloud computing security.
Michel Dagenais is professor at Ecole Polytechnique de Montreal in the department of Computer and Software Engineering. He authored or co-authored over one hundred scientific publications, as well as numerous free documents and free software packages in the fields of operating systems, distributed systems and multicore systems, in particular in the area of tracing and monitoring Linux systems for performance analysis. Most of his research projects are in collaboration with industry and generate free software tools among the outcomes. The Linux Trace Toolkit next generation, developed under his supervision, is now used throughout the world and is part of several specialized and general purpose Linux distributions.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Received: 10 July 2017 Accepted: 19 March 2018
Published online: 13 April 2018

References

1. Wong Q (1998) Salesforce Pushed Silicon Valley Into the Cloud. http://www.newsfactor.com. Accessed 11 July 2016
2. Gonzales D, Kaplan JM, Saltzman E, Winkelman Z, Woods D (2017) Cloud-trust–A security assessment model for infrastructure as a service (IaaS) clouds. IEEE Trans Cloud Comput 5(3):523–536
3. Modi C, Patel D, Borisaniya B, Patel H, Patel A, Rajarajan M (2013) A survey of intrusion detection techniques in cloud. J Netw Comput Appl 36(1):42–57
4. Tsai H-Y, Siebenhaar M, Miede A, Huang Y, Steinmetz R (2012) Threat as a service? Virtualization's impact on cloud security. IT Prof Ma 14(1):32
5. Shea R, Liu J (2012) Understanding the impact of denial of service attacks on virtual machines. In: Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service. IEEE Press. p 27
6. Gupta S, Kumar P (2013) Vm profile based optimized network attack pattern detection scheme for ddos attacks in cloud. In: International Symposium on Security in Computing and Communication. Springer. pp 255–261
7. Masood M, Anwar Z, Raza SA, Hur MA (2013) Edos armor: a cost effective economic denial of sustainability attack mitigation framework for e-commerce applications in cloud environments. In: Multi Topic Conference (INMIC), 2013 16th International. IEEE. pp 37–42
8. Koduru A, Neelakantam T, et al (2013) Detection of economic denial of sustainability using time spent on a web page in cloud. In: Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference On. IEEE. pp 1–4
9. Kwon H, Kim T, Yu SJ, Kim HK (2011) Self-similarity based lightweight intrusion detection method for cloud computing. In: Asian Conference on Intelligent Information and Database Systems. Springer. pp 353–362
10. (2016) Control Groups Resource Management. https://libvirt.org/cgroups.html. Accessed 6 July 2016
11. Navratil M, Dolezelova M, Ondrejka P, Majorainova E, Prpic M, Landmann R, Silas D (2016) Managing System Resources on Red Hat Enterprise Linux 6. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html-single/Resource_Management_Guide/. Accessed 6 July 2016
12. Meyer D, Leisch F, Hornik K (2003) The support vector machine under test. Neurocomputing 55(1):169–186
13. Lonea AM, Popescu DE, Tianfield H (2013) Detecting ddos attacks in cloud computing environment. Int J Comput Commun Control 8(1):70–78
14. Ficco M, Rak M (2015) Stealthy denial of service strategy in cloud computing. IEEE Trans Cloud Comput 3(1):80–94
15. Palmieri F, Fiore U, Castiglione A (2014) A distributed approach to network anomaly detection based on independent component analysis. Concurrency Comput Pract Experience 26(5):1113–1129
16. Choi J, Choi C, Ko B, Kim P (2014) A method of ddos attack detection using http packet pattern and rule engine in cloud computing environment. Soft Comput 18(9):1697–1703
17. Jeyanthi N, Mogankumar P (2014) A virtual firewall mechanism using army nodes to protect cloud infrastructure from ddos attacks. Cybernet Inform Technol 14(3):71–85
18. Jeyanthi N, Iyengar N (2013) Escape-on-sight: an efficient and scalable mechanism for escaping ddos attacks in cloud computing environment. Cybernet Inform Technol 13(1):46–60
19. Chonka A, Abawajy J (2012) Detecting and mitigating HX-DoS attacks against cloud web services. In: Network-Based Information Systems (NBiS), 2012 15th International Conference on. IEEE. pp 429–434
20. Chonka A, Xiang Y, Zhou W, Bonti A (2011) Cloud security defence to protect cloud computing against http-dos and xml-dos attacks. J Netw Comput Appl 34(4):1097–1107
21. Wahab OA, Otrok H, Mourad A (2014) A dempster–shafer based tit-for-tat strategy to regulate the cooperation in vanet using qos-olsr protocol. Wirel Pers Commun 75(3):1635–1667
22. Iyengar NCSN, Ganapathy G, Mogan Kumar P, Abraham A (2014) A multilevel thrust filtration defending mechanism against ddos attacks in cloud computing environment. Int J Grid Utility Comput 5(4):236–248
23. Teng S, Zheng C, Zhu H, Liu D, Zhang W (2014) A cooperative intrusion detection model for cloud computing networks. Int J Secur Appl 8(3):107–118
24. Man ND, Huh E-N (2012) A collaborative intrusion detection system framework for cloud computing. In: Proceedings of the International Conference on IT Convergence and Security 2011. Springer. pp 91–109
25. Singh D, Patel D, Borisaniya B, Modi C (2016) Collaborative ids framework for cloud. Int J Netw Secur 18(4):699–709
26. Ghribi S (2016) Distributed and cooperative intrusion detection in cloud networks. In: Proceedings of the Doctoral Symposium of the 17th International Middleware Conference. ACM. p 7. https://doi.org/10.1145/3009925.3009932
27. Chiba Z, Abghour N, Moussaid K, Rida M, et al (2016) A cooperative and hybrid network intrusion detection framework in cloud computing based on snort and optimized back propagation neural network. Procedia Comput Sci 83:1200–1206
28. Wahab OA, Bentahar J, Otrok H, Mourad A (2017) I know you are watching me: Stackelberg-based adaptive intrusion detection strategy for insider attacks in the cloud. In: IEEE International Conference on Web Services (ICWS). IEEE. pp 728–735
29. Wahab OA, Bentahar J, Otrok H, Mourad A (2017) Optimal load distribution for the detection of vm-based ddos attacks in the cloud. IEEE Trans Serv Comput
30. Yu S, Tian Y, Guo S, Wu DO (2014) Can we beat ddos attacks in clouds? IEEE Trans Parallel Distributed Syst 25(9):2245–2254
31. Somani G, Johri A, Taneja M, Pyne U, Gaur M. S, Sanghi D (2015) Darac: Ddos mitigation using ddos aware resource allocation in cloud. In: International Conference on Information Systems Security. Springer. pp 263–282

32. Desnoyers M, Dagenais MR (2006) The lttng tracer: A low impact performance and behavior monitor for gnu/linux. In: OLS (Ottawa Linux Symposium). Citeseer Vol. 2006. pp 209–224
33. Ezzati-Jivan N, Dagenais MR (2012) A stateful approach to generate synthetic events from kernel traces. Adv Softw Eng 2012:6
34. Ezzati-Jivan N, Dagenais M. R (2013) A framework to compute statistics of system parameters from very large trace files. ACM SIGOPS Oper Syst Rev 47(1):43–54
35. Heller KA, Svore KM, Keromytis AD, Stolfo SJ (2003) One class support vector machines for detecting anomalous windows registry accesses. In: Proc. of the Workshop on Data Mining for Computer Security. vol. 9
36. Shon T, Moon J (2007) A hybrid machine learning approach to network anomaly detection. Inf Sci 177(18):3799–3821
37. Wahab OA, Bentahar J, Otrok H, Mourad A (2015) Misbehavior detection framework for community-based cloud computing. In: Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference On. IEEE. pp 181–188
38. Wahab OA, Mourad A, Otrok H, Bentahar J (2016) Ceap: Svm-based intelligent detection model for clustered vehicular ad hoc networks. Expert Syst Appl 50:40–54
39. Auria L, Moro RA (2008) Support vector machines (svm) as a technique for solvency analysis
40. Konar A, Chakraborty UK, Wang PP (2005) Supervised learning on a fuzzy petri net. Inf Sci 172(3):397–416
41. Fine S, Scheinberg K (2001) Efficient svm training using low-rank kernel representations. J Mach Learn Res 2:243–264
42. Tsang IW, Kwok JT, Cheung P-M (2005) Core vector machines: Fast svm training on very large data sets. J Mach Learn Res 6:363–392
43. Dong J-X, Krzyzak A, Suen CY (2005) Fast svm training algorithm with decomposition on very large data sets. IEEE Trans Pattern Anal Mach Intell 27(4):603–618
44. Kandula S, Katabi D, Jacob M, Berger A (2005) Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2. USENIX Association. pp 287–300
45. Markus-Go (2015) BoNeSi - the DDoS Botnet Simulator. https://github.com/Markus-Go/bonesi. Accessed 6 July 2016
46. (2016) Kernel Virtual Machine. http://www.linux-kvm.org/page/Main_Page. Accessed 6 July 2016
47. (2007) The CAIDA DDoS Attack 2007 Dataset. http://www.caida.org/data/passive/ddos-20070804_dataset.xml. Accessed 10 July 2016
48. Arlitt M, Jin T (1998) 1998 World Cup Web Site Access Logs. http://ita.ee.lbl.gov/html/contrib/WorldCup.html. Accessed 10 July 2016
49. Bhatia S, Schmidt D, Mohay G, Tickle A (2014) A framework for generating realistic traffic for distributed denial-of-service attacks and flash events. Comput Secur 40:95–107
50. (2016) LTTng Analyses. https://github.com/lttng/lttng-analyses. Accessed 6 July 2016
51. Maji S, Berg AC, Malik J (2013) Efficient classification for additive kernel svms. IEEE Trans Pattern Anal Mach Intell 35(1):66–77
52. Kohavi R, Quinlan JR (2002) Data mining tasks and methods: Classification: decision-tree discovery. In: Handbook of Data Mining and Knowledge Discovery. Oxford University Press, Inc. pp 267–276