Clustering-Based IaaS Cloud Monitoring

Mahmoud Abdelsalam*, Ram Krishnan† and Ravi Sandhu‡
*†‡*Institute for Cyber Security,* *‡*Department of Computer Science,* †*Department of Electrical and Computer Engineering*
University of Texas at San Antonio, San Antonio, Texas, USA
Email: **mahmoud.abdelsalam@utsa.edu,* †*ram.krishnan@utsa.edu,* ‡*ravi.sandhu@utsa.edu*

***Abstract*— Organizations increasingly utilize cloud services such as Infrastructure as a Service (IaaS) where virtualized IT infrastructure are offered on demand by cloud providers. A major challenge for cloud providers is the security of virtual resources provided to its customers. In particular, a key concern is whether, for example, virtual machines (VMs) in the datacenter are performing tasks that are not expected of those machines. Given the scale of datacenters, *continuous* security monitoring of the virtual assets is essential to detect unexpected (and potentially malicious) behavior. In this paper, we develop a continuous monitoring framework for cloud IaaS. The proposed framework uses a modified version of sequential K-means clustering algorithm for anomaly detection based on variations in resource utilization that can be observed when cloud insiders or malware perform malicious tasks on cloud customers' VMs. Our approach assumes no prior knowledge of the installed applications on the VMs. Finally, our experiments are performed on data collected from our OpenStack (a popular open-source cloud IaaS software) testbed based on a standard 3-tier web architecture with the ability to scale-out (i.e., multiple copies of the server are spawned) and scale-back (i.e., the number of copies are reduced) on demand. The experiments are based on real-world as well as synthetically injected anomalies.**

***Keywords*-Anomaly Detection; Cloud IaaS; Insider Threat; Malware; Ransomware**

I. Introduction

Cloud systems are becoming increasingly complex due to the large number of services, resources and tenants (customers) involved. In the case of an Infrastructure as a Service (IaaS) cloud—where virtualized resources such as compute, storage and networking are provided as a service to various tenants by a cloud service provider—the tenants' resource usage ranges from a few VMs to hundreds of VMs. For example, applications that use Hadoop are often deployed over hundreds of VMs. Hence automatic and continuous monitoring of VMs for security has become a necessary task for not only the cloud service providers but also their tenants.

Such a monitoring capability is necessary to address many different concerns and threats that arise in IaaS clouds. For example, tenant users could be malicious and infect other tenants' VMs via co-resident attacks [1]. Some of the end users accessing tenant provided services (e.g., a web application hosted on a 3-tier architecture) could inject malware into its VMs. Such issues need to be continuously and automatically monitored given the scale of modern cloud platforms. It is worth mentioning that cloud security monitoring takes place at many points in the cloud (e.g. cloud services APIs, storage, network or VMs). In this work, we focus our scope on monitoring VMs in IaaS clouds.

In this paper, we investigate an approach to monitor IaaS clouds using clustering of VMs based on similar resource usage and interaction. Most anomaly detection approaches in cloud do not leverage the properties of the application architecture (such as n-tier architecture and Hadoop) or the specific characteristics of cloud (e.g. autoscaling). Instead, they attempt to characterize anomalous behavior of a specific VM without the context of cloud. As a matter of fact, the behavior that is considered anomalous for one VM might not be anomalous for another VM. In practice, a cloud system has many tenants and each tenant has many VMs. Typically, those VMs are not randomly created, but rather created in a systematic manner (e.g. based on a specific autoscaling policy), with each group of VMs performing a specific kind of task. In this paper, we take a holistic view of cloud rather than individual VMs in order to detect anomalies for a given tenant. The major contributions of this paper are two-fold:

- We develop an approach for detecting anomalous behavior of VMs in scenarios involving autoscaling in IaaS clouds by employing machine learning. In particular, we develop a modified version of sequential K-means [2] clustering algorithm.
- We implement this approach in OpenStack, build a realistic hardware testbed using OpenStack, and evaluate its effectiveness against real-world and synthetic anomalies.

To the best of our knowledge, this is the first clustering approach to profile VMs with respect to each other in order to detect anomalous behaviour.

The rest of the paper is organized as follows. Section II describes related work. Section III describes the clustering techniques and their usage for this work. Section IV provides an overview of the proposed framework and explains the methodology for implementing the framework. Section V explains the OpenStack-based tested setup. Section VI discusses the experiments we performed on an OpenStack-based tested and its results. Finally, Section VII concludes our findings and gives some directions for future work.

II. Related Work

Cloud monitoring has recently been the focus of many works. Several approaches have been proposed for cloud security monitoring. One of the most used techniques is Intrusion Detection Systems (IDS). The work in [3] provides a comprehensive overview of the approaches for IDS. Mainly, these are categorized into two main approaches: signature-based and anomaly-based. Signature-based approaches are effective methods for known attacks, however they are not effective to detect unknown attacks or even a variant of a known attack since they depend on signatures of known attacks. Anomaly-based approaches are effective against new attacks because they are behavioral-based. However, they suffer from accuracy rate. This paper's approach is behavior-based anomaly detection for VMs that belong to a specific tenant. As we will see, the behavior-based approach has a significant advantage in malware detection in the context of auto-scaling VMs in IaaS.

The work in [4] give a summary of the different techniques for anomaly detection. In this work, a framework for choosing the right anomaly detection technique is presented. The key components include the research area, nature of data, data labels, anomaly type, application domain and output. These components must be determined carefully based on the context. Although more than one anomaly detection technique may be suitable for a single situation, each technique has its own drawbacks. The work in [4] also categorizes the anomaly detection techniques into classification-based, clustering-based, nearest neighbour-based and statistical-based.

Statistical techniques [5], [6] for anomaly detection suffer from performance overhead due to their complexity, lack of scalability and the need of prior knowledge.

Clustering VMs of similar behaviour has been addressed in [7]–[9], however the purpose of the clustering is mainly focused on the monitoring system scalability (by reducing the number of instances to monitor) in multi-cloud systems rather than security, since an attack can be on one of the unmonitored instances.

Classification of VMs has been previously used for anomaly detection. The work in [10] is based on one class Support Vector Machine (SVM) for detection of malware in cloud infrastructure. The approach gathers features at the system and network levels. The system level features are gathered per process which includes memory usage, memory usage peak, number of threads and number of handles. The network level features are gathered using CAIDA's CoralReef[1] tool. The study shows high accuracy results. However, gathering features per process is an exhausting and intrusive operation. A datacenter with thousands of VMs each with hundreds of processes running can have a significant performance degradation. Furthermore, it is not obvious how one would determine what processes to monitor and how to characterize malicious and benign behaviors of various processes.

Thus, in this paper, we consider VMs as blackboxes and we assume no prior knowledge of applications or processes running on the VMs. We take a cluster-based approach to cluster VMs of similar behavior and look for deviations in their behavior with respect to the behavior of their respective clusters. This is an effective approach in cloud, since clusters of auto-scaling VMs, typically, are expected to exhibit similar behavior within each cluster.

III. Clustering

Clustering is a technique to group similar data samples into clusters. One of the main assumptions, which is essential in using clustering for anomaly detection, is that the number of normal data samples are far greater than the number of anomalies. We believe that this is true in the security domain since having anomalies is not always the case in any system. We refer to anomalies as any malicious behaviour due to system breach or malware. Clustering techniques are not as effective if anomalies create a cluster by themselves. False anomalies remain a big challenge to clustering techniques. We discuss reducing the number of false anomalies in section IV-E.

A. K-means

K-means clustering algorithm is one of the most popular clustering algorithms due to its better performance and simplicity. It groups data samples based on their feature values into k clusters. Data samples that belongs to the same cluster have similar feature values. Knowing the best k value remains a challenge, although there are some proposed approaches [11]. Our framework is intended to be practical so it can be used by cloud customers (tenants) in real-world scenarios where performance and simplicity of security capabilities are given more importance. We make a reasonable assumption that the tenants, at the very least, know of the kind of web application architecture that they are planning to deploy in cloud. For example, a tenant might host a web application on the cloud using a 3-tier web architecture (web servers, application servers, and database servers). Therefore, there are three clusters, so $k = 3$ is an input to the monitoring system. Here are the high-level steps of K-means clustering:

1) Set the number of clusters (k).
2) Initialize k centroids/means (by guessing their initial values or by randomly choosing them).
3) For each data sample, compute the distance to all centroids and assign it to the closest centroid.

[1] CoralReef Suite: https://www.caida.org/tools/ measurement/coral-reef/

4) Modify the values of the centroids based on the new data sample.
5) Repeat steps 3 and 4 until each of the centroid values do not change.

The Euclidean distance is used to compute the distance between a data sample and the centroids. It is defined as:

$$dis(x,c) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where x and c are vectors of quantitative features of the data sample and the centroid respectively.

B. Sequential K-means

Since our framework is meant to be as practical as possible, there are a few assumptions that need to be addressed:

- The data is time series, meaning that we process one data sample at a time.
- A training phase is infeasible since each cloud customer would need to go through a separate training phase as the nature of its data can be very different from those of others.

Since we are dealing with time series data, sequential K-means is used, which is a variation of the original K-means clustering algorithm. In sequential K-means, the data are infinite and arrive one sample at a time. Another difference between K-means and Sequential K-means is that K-means iterates over the same data samples many times until the centroid values no longer changes while Sequential K-means does not as illustrated in Listing 1.

```
make initial guesses for means (centroids)
    $m_1, m_2, ..., m_k$
set the counters $n_1, n_2, ..., n_k$ to zero
until interrupted
  get the next sample $x$
  if $m_i$ is closest to $x$
    increment ni
    replace $m_i$ with $m_i + (1/n_i) * (x - m_i)$
  end_if
end_until
```

Listing 1: Sequential K-means

IV. Framework Overview

This section provides an overview of the proposed framework as well as a description of the methodology to detect anomalies using modified sequential K-means. First, we define the features of the VMs. These features will be collected and used for clustering. Then, the features are normalized since they are not of the same scale. Normalization is done based on the Min-Max [12] approach. Lastly, a real-time clustering (based on modified sequential K-means) is applied and anomalies are detected based on the specified threshold.

Table I: Virtual machines features/metrics

Metric	Description	Unit
CPU util	Average CPU utilization	%
Memory usage	Volume of RAM used by the VM from the amount of its allocated memory	MB
Memory resident	Volume of RAM used by the VM on the physical machine	MB
Disk read requests	Rate of disk read requests	rate/s
Disk write requests	Rate of disk write requests	rate/s
Disk read bytes	Rate of disk read bytes	rate/s
Disk write bytes	Rate of disk write bytes	rate/s
Network outgoing bytes	Rate of network outgoing bytes	rate/s
Network incoming bytes	Rate of network incoming bytes	rate/s

A. Features Definition

VM features are usually divided in two categories: *internal and external.* For the sake of practicality, we assume no prior knowledge of any information internal to the VMs. Our framework deals with VMs as blackboxes. Table I shows the external features that are selected to be collected for every monitored VM. (Note that these are just a selection for illustration purposes–in practice, many more features are available.)

B. Features Normalization

Clustering algorithms can be very sensitive to data scales (since more weight goes to features with higher values). Since data samples are not of the same scale, data normalization is needed. We use a simple data normalization technique called Min-Max. Min-Max normalization is a technique where you can fit the data with a pre-defined boundary. Min-Max normalization is simply defined as:

$$A' = \frac{A - minValue_A}{maxValue_A - A},$$

where A is a data sample and A' is the normalized data sample. Pre-defining the $maxValue_A$ can be tricky for time series data. Thus, we employ a Min-Max normalization based on a fixed-size sliding window. Note that our overall detection approach is not dependant on the specific normalization approach used.

C. Modified Sequential K-means

As stated in section III, one challenge in using clustering is the number of false positives. To reduce the rate of false positives, we add a *stabilizing time* parameter. Stabilizing time is an input parameter which represents the time to wait until each newly created VM is booted up and configured. For example, if a new VM is created by the scaling policy, the cloud system may need to tie it to a load-balancer, bootup the operating system, install a webserver and perform other configurations. The monitoring framework should not be influenced by measurements during this period. Rather, it should wait until the webserver is completely up and running as intended.

```
make initial guesses for means (centroids)
    $m_1, m_2, ..., m_k$
set the counters $n_1, n_2, ..., n_k$ to zero
counter $j$ //Number of current VMs
set $stabilizingTime[1..j]$ to $y$ minutes
set $assigningTime[1..j]$ to $z$ minutes
def $VMClusters[1..k]$ //VM i belongs to cluster
    [1..k]
set $VMClusterCounts[1..j][1..k]$ to zero
until interrupted
  get the next sample $x$
  get VM $v$ //x sample belongs to VM v
  if $stabilizingTime[v]$ > 0
    decrease $stabilizingTime[v]$
    continue //get next sample
  end_if
  if $v$ is assigned to a cluster
    $c$ = $VMClusters[v]$
    if $x$ is not anomaly
      increment $n_c$
      replace $m_c$ with $m_c + (1/n_c) * (x - m_c)$
    else
      //Anomaly code here
      Report anomaly
    end_if
  end_if
  if $v$ is not assigned to any cluster
    if $m_i$ is closest to $x$
      increment $n_i$
      replace $m_i$ with $m_i + (1/n_i) * (x - m_i)$
    end_if
    //If time end assign it to a cluster
    if $assigningTime[v]$ <= 0
      set $VMClusters[v]$ to index of
          $max(VMClusterCounts[v])$
    else
      if $m_i$ is closest to $x$
        increment $VMClusterCounts[v][i]$
      end_if
      decrease assigningTime[v]
    end_if
  end_if
end_until
```

Listing 2: Modified Sequential K-means

Each data sample that belongs to a VM will be clustered according to the clustering algorithm used. Therefore, a data sample from a particular VM can belong to a cluster x at one time while another data sample of the same VM can belong to cluster y. The clustering algorithm will not report this as an anomaly. Since we assume no prior information about each whether a particular VM is a web server or an app server or a DB service (because many VMs can be created automatically by some scaling policy) and cannot decide if a data sample really belongs to a particular cluster or not, this presents a problem. We overcome this problem by slightly modifying the clustering algorithm by adding a new parameter called *assigning time*. This parameter represents the time needed for the monitoring system to make sure that a VM belongs to a certain cluster. Once this is done, all the data samples that arrive for a particular VM will be compared to its assigned cluster. For example, assume there are three clusters: web servers, application servers and database servers. Suppose VM x is newly created. For the first m minutes, x's data samples are compared and provisionally aligned to all the three clusters. The cluster with the maximum number of data samples is the cluster that is assigned to VM x. After that, x's data samples are compared only to its assigned cluster to check for anomalies as well as for updating the cluster's information. Listing 2 shows pseudo-code for the modified sequential K-means.

It is worth mentioning that one of the most critical aspects of cloud monitoring is the complexity of the monitoring system in place. The modified sequential k-means still iterates only once over any data sample, which results in a linear complexity.

D. Anomaly Detection

A sample is considered an anomaly if it is far from its centroid by a specified threshold. There are two parameters that need to be set up by the cloud customer. *Anomaly threshold* ($x\%$) and *Anomaly number* (y/min). Anomalies are detected based on x. If a particular sample is off by $x\%$ from its assigned centroid, then it is marked as an anomaly. Once, there are y anomalies per minute, an alarm will be raised. This is done to reduce the number of false alarms due to behaviour fluctuations.

E. Parameters Tuning

The framework has some important parameters that need to be set up as accurate as possible since they affect the clustering algorithm as well as the anomaly detection. These parameters are given by the cloud customer because they differ based on each scenario.

Clustering parameter: *stabilizing time*(s). This parameter is dependant on each customer scenario because it is affected by the resources assigned to a VM, the operating system boot time and the internal configurations. This parameter can be easily set by having the VM send a signal to the monitoring component after booting and configuration activities are completed.

Anomaly detection parameter: *anomaly threshold*(t). While increasing the threshold reduces the number of false alarms, it also increases the chance of not detecting real anomalies. On the other hand, while decreasing the threshold increases the number of false alarms, it also decreases chance of anomalies getting through undetected. Thus, tuning these parameters are very important.

Normalization parameter: *window size*(w). This parameter represents the window size in which the sliding-window Min-Max normalization should keep record. On one hand, keeping many history samples might affect the new data samples that started shifting toward different values. On

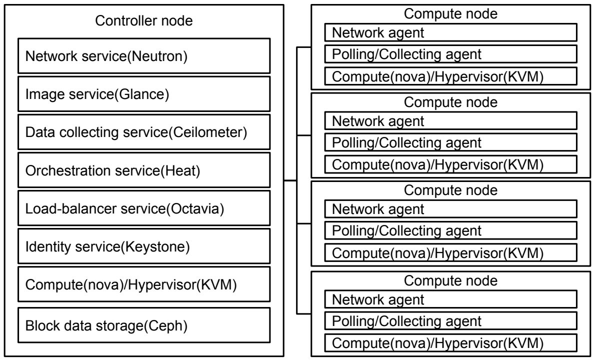

Figure 1: Testbed Setup

the other hand, keeping few history samples will distort the data samples during the normalization stage.

Automatic parameter tuning is a non-trivial challenge for many behavior-based anomaly detection techniques. We plan to further investigate such issues in future work.

V. Experiments Setup

A. Testbed Environment

The cloud testbed used in this work uses OpenStack[2], which is a major cloud orchestration software used by many cloud providers. Figure 1 shows the setup of our cloud testbed that has been built for our experimentation. The testbed is composed of five high-capacity physical nodes. One controller node is responsible for services such as the dashboard, storage, network, identity, and compute. Four compute nodes are responsible just for the compute service. The compute nodes also contain network agents as well as polling agents responsible for collecting data samples. Data collection agents are configured to 30 seconds intervals.

B. Use Case Application

In order to simulate a real environment as much as possible, a three-tier web application is implemented as a use case, which is one of the most common cloud architectures according to [3]. A three-tier web application is an application program that is organized into three major parts, where each tier can be hosted on one or more different hosts. In our case, these hosts are the cloud nodes hosting the compute services.

Figure 2 shows the 3-tier web application built on top of our testbed. The application used for this work is Wordpress[4], a major open-source content management system (CMS) based on PHP and MySQL. In a typical 3-tier web application, a web server hosts the static pages, an application server hosts the application logic, and a database server stores the data. The workflow typically is as follows:

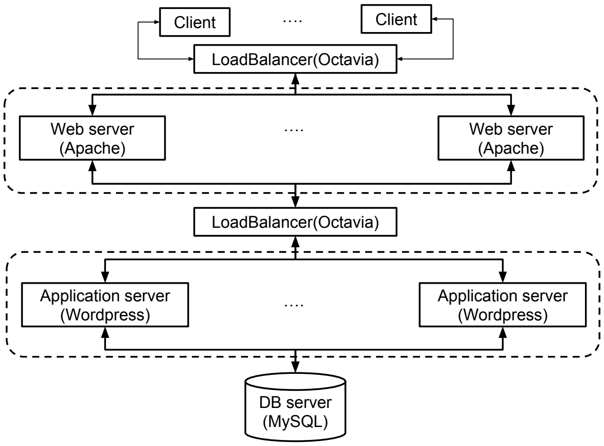

Figure 2: 3-tier Web Application

1) Web server receives a request from a client.
2) Web server replies back if it is a static page request that doesn't need computation in the application logic.
3) Otherwise, it sends the request to an application server.
4) Application server accesses the database server if it needs any stored data and replies back to the web server.
5) Web server replies back to the client.

Separating the application into three tiers allows for the concurrent development and configuration of the three different tiers. It also allows for the scaling of the three-tiers separately depending on demand. In our case, scaling out and back is enabled for the web and application servers but not the database server. We do not scale DB servers for simplicity since that would require replicating and keeping the databases consistent.

The 3-tier web application used for this work utilizes two load balancers. A web server load balancer, which is responsible for distributing the requests to various web servers and an application server load balancer, which is responsible for distributing the requests to various the application servers. The policy of distribution of load can be configured.

C. Traffic Generation

Many works in the literature use the Poisson process for generating traffic because of its simplicity. While still applicable in many cases, it has proven to be inaccurate for Internet traffic. Internet traffic is known to be of self-similar nature [13]–[16]. All of our experiments were conducted twice based on two traffic generation models: *Poisson process* and *ON/OFF Pareto*.

[2] Openstack website. https://www.openstack.org/
[3] Amazon architecture references. https://aws.amazon.com/architecture/
[4] Wordpress website. https://wordpress.org/

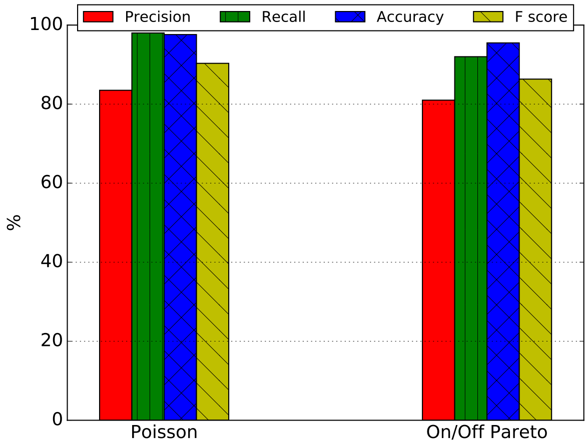

Figure 3: Injected anomaly detection with optimally tuned classifier

A multi-process program is built, acting as large number of concurrent users, to send requests to the webservers' load-balancer. The simulation parameters are as follows:

- Generator: On/Off Pareto, Poisson
- Number of concurrent clients: 50
- Requests arrival rate/hour: 3600
- Type of requests: GET and POST (randomly generated)

The On/Off Pareto input parameters are set according to the NS2[5] tool defaults. The amount of traffic is chosen to stress the VMs to trigger the scalability policy. The scale-out policy is set to scale whenever the CPU util average of a specific tier (e.g. app server tier or web server tier) is above 70% and scale-back when the CPU load average is below 30%.

VI. Results and Discussion

Anomaly injection is randomized along two dimensions: time of injection and magnitude of the anomaly. We explore the effectiveness of our framework based on three use cases. In each use case, Poisson process and On/Off Pareto traffic generation models are used. The duration of all the experiments was one hour.

Evaluation methodology. We use four metrics[6,7] to evaluate the effectiveness and applicability of our approach [17].

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Fscore = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

When the system detects an anomaly, it is considered a *positive* outcome. When the system does not detect an anomaly, the outcome is *negative*. Therefore:

1) True Positive (TP): anomaly occurred and was successfully detected.
2) False Positive (FP): anomaly did not occur and was detected.
3) True Negative (TN): anomaly did not occur and was not detected.
4) False Negative (FN): anomaly occurred and was not detected.

Precision refers to the number of data samples, detected as anomalies, that are actually true anomalous samples. On the other hand, recall refers to the percentage of correctly detected anomalies based on the total number of anomalies of the data samples. Accuracy measures the correct classification of all data samples. The harmonic mean (F Score) is the weighted average of precision and recall.

We now discuss three kinds of attacks that we evaluated. Two of these were simulated and one was using real-world samples of malware. All attacks are injected into a random application server.

A. *Injected Anomalies*

The effectiveness of the framework is tested by injecting anomalies in randomly chosen VMs. Injected anomalies are *cpu, memory and disk intensive*.

Figure 3 shows the detection results of the experiment. The results show that the detector performs similarly well on both traffic models with accuracies over 90%. The precision suffers a considerable amount of FPs due to the nature of the fluctuated traffic load and resource usage.

B. *EDoS*

Not all threats intensively use resources. For instance, Economic Denial of Sustainability (EDoS) [18] attacks try to avoid the intensive resource usage by keeping a low-profile. One form of EDoS is to create many VMs while remaining dormant and idle (which can be done by stealing the credentials of one of the cloud tenants who has authority to create VMs).

Such attacks try to waste resources in way that is not obvious to the tenants. This can impact neighboring VMs because of unavailability of resources. In our experiment, EDoS was simulated by randomly injecting VMs, with the only outcome during the life-time of each injected anomalous VM being TPs or FNs. The injected VMs remain dormant and maintain a low-resource usage profile.

[5] NS2 tool manual. http://www.isi.edu/nsnam/ns/doc/node509.html

[6] Wikipedia F score. https://en.wikipedia.org/wiki/F1_score

[7] Wikipedia precision and recall. https://en.wikipedia. org/wiki/Precision_and_recall

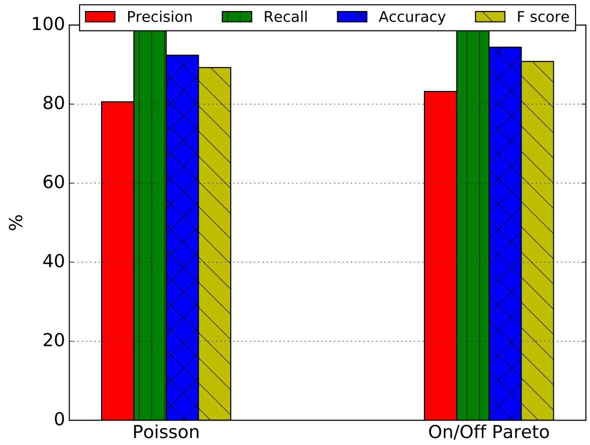

Figure 4: EDoS detection with optimally tuned classifier

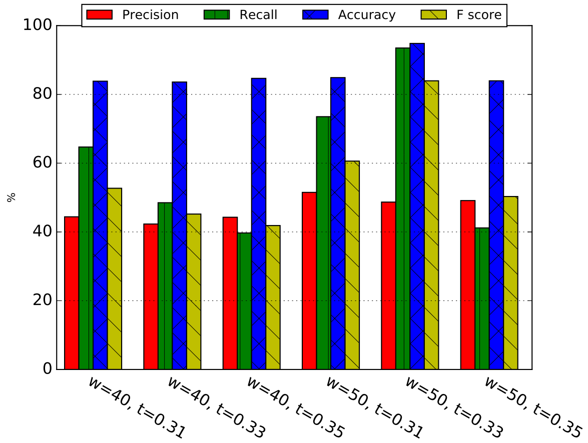

Figure 5: KillDisk ransomware detection - Poisson

Figure 4 shows the detection evaluation performance for this experiment. It is clear that the detector is effective against this kind of EDoS attack since the injected VMs have very different profiles than the 3-tiers (clusters). The results show a loss of precision due to detected FPs. After investigation, this was determined to be due to spawning high load of VMs (by injecting and scaling). The cloud was unresponsive which prevented any traffic load from reaching the VMs resulting in sudden drop in resource usage. As such, this is not caused by poor detection. In fact, this helps security administrator's detecting times when their system is down.

C. Ransomware

Ransomware is popular kind of malware. Netskope[8] quarterly cloud report states that 43.7% of the cloud malware types detected in cloud apps are common ransomware delivery vehicles. Ransomware basically encrypt various files on victim's hard drives and then seeks a ransom to get the files decrypted. KillDisk linux-variant ransomware is used for experiments. The samples were obtained from VirusTotal[9]. The one hour experiment was divided in two phases: normal phase (first 20 minutes) and malicious phase (the next 40 minutes). The malicious phase is the time after the ransomware is injected, with the only valid outcome being TPs or FNs. The ransomware was injected into a random application server due to the fact that applications are more likely vulnerable than the widely used web servers.

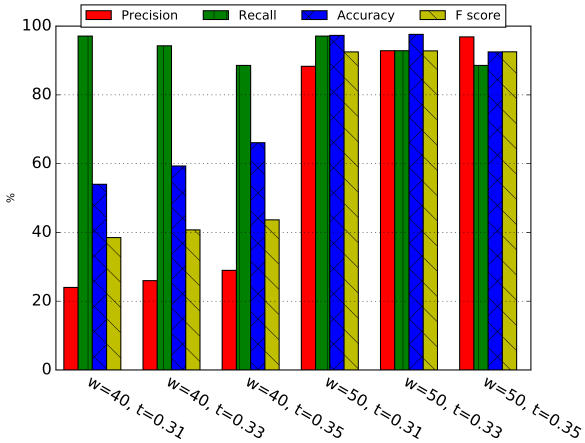

Figure 6: KillDisk ransomware detection - On/Off Pareto

The results of this experiment are shown in Figures 5 and 6 where the bars are produced by calculating the performance metrics for each set of modified sequential k-means parameters. The two most critical parameters are chosen (by experimentation) for optimal ($w = 50$, $t = 0.33$) and near optimal results. In the case of Poisson traffic, the detector suffers many FPs. On the other hand, in the case of On/Off Pareto traffic, it is clear from the results that the detector is effective with detection performance of more than 90% overall.

VII. Conclusion and Future Work

In this paper, we investigated clustering-based cloud monitoring for detecting anomalies in scale-out and scale-back scenarios in IaaS cloud. In order to have a better security in the cloud, we proposed a framework to cover a subset (i.e., VMs resource usage and interaction) of the cloud monitoring space in IaaS. The framework uses clustering for IaaS monitoring in cloud. It uses a modified version of the Sequential K-means algorithm to overcome the problem of high false alarm rate when using clustering. The results showed how the framework can detect anomalies in three scenarios. The experiments show that

[8] Netskope website. https://www.netskope.com

[9] VirusTotal website. https://www.virustotal.com

parameter tuning is a very important issue and is dependent on the use case.

One limitation of our framework is that it is vulnerable to low-profile anomalies and malware. Detecting those types of anomalies are proven to be hard using only resource usage metrics. A general concern is when attacks try to gradually change normal behavior of a VM. However our approach makes the task harder since the change of behavior has to be in all the VMs of the same cluster at the same time. Another limitation is that an expert is necessary for parameter tuning.

In the future, we plan to investigate various issues such as dynamic parameter tuning, which is essential for getting more accurate results. Also, we plan to experiment on different cloud reference architectures other than the 3-tier web architecture. Lastly, we plan to use and compare different machine learning algorithms for anomaly detection.

Acknowledgment

This work is partially supported by DoD ARL Grant W911NF-15-1-0518 and NSF grants CNS-1111925, CNS-1423481, and CNS-1553696.

References

[1] 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 *Proc. of the 16th ACM CCS*. ACM, 2009, pp. 199–212.

[2] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[3] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[5] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *12th IFIP/IEEE IM*. IEEE, 2011, pp. 385–392.

[6] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online detection of utility cloud anomalies using metric distributions," in *2010 IEEE NOMS 2010*. IEEE, 2010, pp. 96–103.

[7] C. Canali and R. Lancellotti, "Automated clustering of virtual machines based on correlation of resource usage," *Communications Software and Systems*, vol. 8, no. 4, pp. 102–109, 2012.

[8] C. Canali and R. Lancellotti, "Automated clustering of vms for scalable cloud monitoring and management," in *Software, 20th SoftCOM, 2012*. IEEE, 2012, pp. 1–5.

[9] C. Canali and R. Lancellotti, "Automatic virtual machine clustering based on bhattacharyya distance for multi-cloud systems," in *Proc. of MultiCloud*. ACM, 2013, pp. 45–52.

[10] M. R. Watson, A. K. Marnerides, A. Mauthe, D. Hutchison *et al.*, "Malware detection in cloud computing infrastructures," *IEEE TDSC*, vol. 13, no. 2, pp. 192–205, 2016.

[11] D. T. Pham, S. S. Dimov, and C. Nguyen, "Selection of k in k-means clustering," *Proc. of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, no. 1, pp. 103–119, 2005.

[12] T. Jayalakshmi and A. Santhakumaran, "Statistical normalization and back propagationfor classification," *International Journal of Computer Theory and Engineering*, vol. 3, no. 1, p. 89, 2011.

[13] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on networking*, vol. 5, no. 6, pp. 835–846, 1997.

[14] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on networking*, vol. 2, no. 1, pp. 1–15, 1994.

[15] A. Varet and N. Larrieu, "Realistic network traffic profile generation: theory and practice," *Computer and Information Science*, vol. 7, no. 2, pp. pp–1, 2014.

[16] M. Wilson, "A historical view of network traffic models," *Unpublished survey paper. See http://www. arl. wustl. edu/ mlw2/classpubs/traffic models*, 2006.

[17] O. Maimon and L. Rokach, *Data mining and knowledge discovery handbook*. Springer, 2005, vol. 2.

[18] G. Somani, M. S. Gaur, and D. Sanghi, "Ddos/edos attack in cloud: affecting everyone out there!" in *Proc. of the 8th SIN*. ACM, 2015, pp. 169–176.