User-Profile-Based Analytics for Detecting Cloud Security Breaches

Trishita Tiwari[*], Ata Turk[*], Alina Oprea[†], Katzalin Olcoz [*‡] and Ayse K. Coskun[*]
[*]Electrical and Computer Engineering, Boston University, Boston, MA, USA
Email: {trtiwari,ataturk,acoskun}@bu.edu
[†]Computer and Information Science, Northeastern University, Boston, MA, USA
Email: a.oprea@northeastern.edu
[‡]Computer Architecture and Automation, Universidad Complutense de Madrid, Spain
Email: katzalin@ucm.es

***Abstract*—While the growth of cloud-based technologies has benefited the society tremendously, it has also increased the surface area for cyber attacks. Given that cloud services are prevalent today, it is critical to devise systems that detect intrusions. One form of security breach in the cloud is when cyber-criminals compromise Virtual Machines (VMs) of unwitting users and, then, utilize user resources to run time-consuming, malicious, or illegal applications for their own benefit. This work proposes a method to detect unusual resource usage trends and alert the user and the administrator in real time. We experiment with three categories of methods: simple statistical techniques, unsupervised classification, and regression. So far, our approach successfully detects anomalous resource usage when experimenting with typical trends synthesized from published real-world web server logs and cluster traces. We observe the best results with unsupervised classification, which gives an average F1-score of 0.83 for web server logs and 0.95 for the cluster traces.**

***Index Terms*—Cloud Security, Virtual Machine Resource Usage, Machine Learning**

I. Introduction

Cloud users are increasingly becoming lucrative targets for cyber attacks. A few examples of recent attacks in the cloud include the iCloud data breach that leaked private images from celebrity accounts [1], botnets used for crypto-currency mining in Amazon cloud services [2], malware that stole credit card information from Chipotle servers [3], and the hack on Sony pictures in 2014 [4]. Indeed, the number and diversity of attacks on cloud-based services continue to expand. The prominent attack vectors used in cloud breaches involve compromise of a user's resources, such as account compromise due to stolen credentials, or VM infection with malware [5]. According to a recent report, 15% of users accessing cloud applications have their accounts compromised [6]. As such, it becomes critical that cloud providers institute more effective security mechanisms to detect and prevent compromise of cloud resources of users.

A. Background and Related Work

Standard security protections implemented by cloud providers (e.g., data encryption, data replication, trusted hardware, and others) offer protection of the cloud infrastructure, but fail to prevent against breaches of user credentials or user computing resources. While other methods such as two-factor authentication attempt to protect cloud users, they are not effective once an account is already compromised. Intrusion Detection Systems (IDS, such as Snort [7]) are typically deployed by cloud providers to detect network-level attacks (such as denial of service and communications with malicious destinations), but they are agnostic to cases in which attackers obtain access to a user's cloud resources. This leaves users of the cloud exposed to new attack vectors, currently largely undetected with existing defense mechanisms.

Machine learning and statistical techniques for attack detection have shown great promise to complement traditional defenses in enterprise settings and private clouds (e.g., [8], [9], [10]); thus, we believe that they can be used for proactive cyber attack detection in public clouds. While there has been some work in using machine learning to improve IDS systems [11], [12], [13], [14], they mainly explore network-related metrics. Other research in this domain includes profiling what a user enters on the terminal and determining which out-of-character commands can be flagged [15]. However, such studies are not targeted towards cloud users and do not explore resource usage data as a means to detect breaches. Even though machine learning and statistical techniques have been applied before in the context of cloud computing to detect performance anomalies [16], [17], [18], optimize resource allocation [19], and reduce energy usage [20], these approaches have not, to the best of our knowledge, been implemented to defend against resource compromises in public cloud.

B. Our Contributions

In this paper, we take the first steps in applying machine learning and statistical methods for detecting resource compromises in public clouds. We propose the use of anomaly detection techniques in user behavior profiling with the goal of detecting various types of breaches that end up with a user's VM's resources being acquired by the attacker. Our methodology involves continuous monitoring of resource usage data (e.g., CPU, disk, memory usage, etc.) and representing it as time-series data. Our algorithms then intelligently analyze the user's short and long-term typical behavior and identify not only extreme outliers, but also many other subtle usage patterns that do not conform to the user's usual behavior. Our techniques learn "legitimate" user profiles over time across multiple dimensions and automatically identify deviations from users' typical observed behavior. These methods require minimal human intervention (i.e., limited mainly to investigate the findings produced by the anomaly detection algorithms), and have the advantage of providing a multi-dimensional, longitudinal view of the cloud infrastructure, which can be extended with other monitored metrics (e.g., power consumption, network usage, etc.).

We experiment with three different techniques: (1) simple statistical approaches such as moving average, exponential average, and percentile-based thresholds; (2) unsupervised classification via One-Class SVM; and (3) regression through Long Short-Term Memory (LSTM) networks, which are recurrent neural network architectures. We test our framework using synthetic data that emulates two real-world published datasets: a NASA web server's logs and Google traces collected from a production cluster. In our experiments, unsupervised classification produces the best results in identifying injected anomalies, with F1-scores ranging from 0.83 to 0.95, but methods like regression based prediction with LSTMs also perform relatively well, with 84% of the data being correctly classified. Other statistical techniques such as moving and exponential averages or percentile based thresholding do not seem to work well with complex patterns found in real world datasets, thereby generating F1-scores as low as 0.40.

The rest of the paper is organized as follows: Section II describes the proposed approach for detecting anomalies. Section III explains how the data used to test our approach has been obtained. Section IV presents the experiments and results. Finally, conclusions are presented in Section V.

II. Proposed Anomaly Detection Approach

In this section, we first introduce our system and threat models and then describe how the proposed anomaly detection engine operates. As seen in Figure 1, the monitoring system collects several metrics (CPU, memory usage, network traffic data, etc.) about user VMs that are critical to their performance on the cloud. Metrics are collected periodically and stored as time-series data. These are then processed to determine if they are sufficiently close to the particular user's "typical" trends or are anomalous. Our assumption is that compromised VMs will most often lead to resource usage trends that are different than the compromised user's typical trends.

We investigate two types of approaches for anomaly detection. The first approach labels entire datasets as regular or anomalous using One-Class SVM. The second approach is more fine grained; instead of classifying entire datasets as regular or anomalous, individual data points are classified. Regression and statistical approaches are investigated at the level of individual data point labeling.

A. System and Threat Models

We consider a public infrastructure-as-a-service (IaaS) cloud in which users can pay and request compute and storage resources. The cloud provider fully controls the cloud infrastructure (compute, storage, networking) and is responsible for protecting the infrastructure against breaches. We assume that the provider itself is trustworthy, but we do not have any guarantees about the cloud physical infrastructure, which might be targeted by various attacks (e.g., denial-of-service attacks, malware, side-channel attacks, etc.).

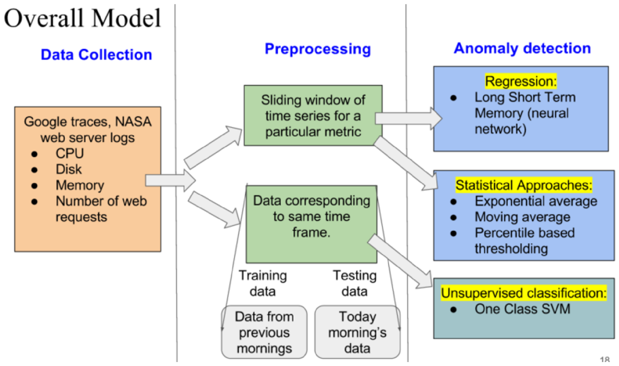

Overall Model
Data Collection
Preprocessing
Anomaly detection
Google traces, NASA web server logs
CPU
Disk
Memory
Number of web requests
Sliding window of time series for a particular metric
Data corresponding to same time frame.
Training data
Testing data
Data from previous mornings
Today morning's data
Regression:
Long Short Term Memory (neural network)
Statistical Approaches:
Exponential average
Moving average
Percentile based thresholding
Unsupervised classification:
One Class SVM
18

Fig. 1: Overall anomaly detection model.

While we expect most of the users to run legitimate workloads in their virtual machines, we would like to be resilient against potentially malicious users or users whose resources might get compromised. We assume that attackers could get access to legitimate cloud resources, such as user credentials (e.g., through leakage in public code repositories), or users' VMs (e.g., by installing malware and escalating privileges). Our goal is to detect these *resource compromise* attacks proactively.

B. Monitoring VMs

Our dataset includes the following metrics: CPU usage, memory usage, and network usage data (i.e., the intensity of the network traffic), as these metrics are highly likely to reflect changes in a compromised VM. Cloud providers can also easily monitor other metrics that might be enlightening: disk usage, number of VMs associated with a particular user's account, number of processes on the VM, disk and memory I/O frequency, disk and memory read-write frequency. In fact, using as many metrics as possible is beneficial as it increases the probability of even stealthy attacks showing up in at least one of the metrics.

C. One-Class SVM for Anomaly Detection

One-Class SVM is an unsupervised learning model that performs novelty detection. Essentially, it treats all of its training data as "normal", and fits a boundary that encompasses these training sets. When testing the model, One-Class SVM checks whether the testing dataset fits within the boundary it had found during training. If it does, then the dataset is classified as normal, else it is labeled anomalous.

We extract 10 features from each time-series before feeding it into the SVM. These include percentiles (25^{th}, 50^{th}, 75^{th}), average, standard deviation, minimum, maximum, skew, kurtosis and the period (specifically, the period of the component signal that contributes the maximum power to the time-series as seen in the discrete Fourier transform). Utilizing features as opposed to training the SVM on raw data-points from each time-series helps both to improve the accuracy of the classification, and also reduces overhead by compressing each time-series into a selected set of features representing the most important attributes.

We trained the SVM with numerous instances of regular time-series data corresponding to each metric (CPU, memory, etc.) for certain fixed time windows (e.g., one day's data, one week's data, etc.). Note that a separate model of the SVM was developed for each time window and metric pair. We then tested each model of the SVM with regular and anomalous data that corresponded to the same metric and time window (e.g., if we trained the SVM with regular data corresponding to one day's CPU usage of user X, we tested it with both regular and anomalous one day CPU usage corresponding to the same user).

D. Regression via LSTMs

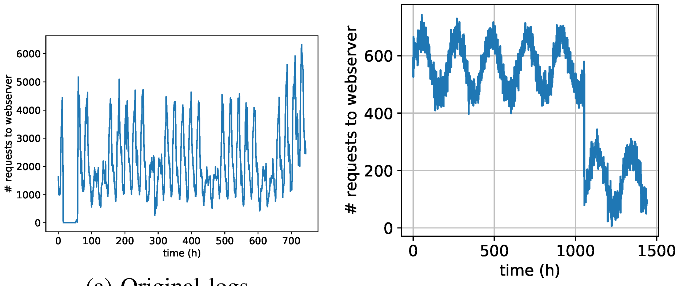
LSTM stands for Long Short Term Memory, and it is a special type of recurrent neural network (RNN) with better long-term retention than traditional neural networks. Our approach in this paper involves feeding the LSTM a subsection of the time-series data representing the regular resource usage from the past (e.g., last 3 week's CPU usage), and then asking it to forecast the remaining data points. This forecast should reflect the patterns seen in the regular training data. However, the remaining part of the time-series could either represent regular or anomalous data. Then, for each forecasted data-point in the LSTM's prediction, we compute its deviation from the corresponding actual data-point. If the actual data point deviates from the forecast beyond a certain threshold (empirically chosen to be 10% for these experiments), then we classify the actual data as anomalous. Note that like the SVM, we had separate LSTM models for each metric.

E. Statistical Approaches for Anomaly Detection

We have implemented the following simple statistical approaches: (1) moving averages, (2) exponential averages, and (3) percentile-based thresholding. Like LSTMs, these approaches also classify individual data points within a time-series as regular or anomalous. Generally speaking, in order to classify the current data-point, we compute a few statistics on a sliding window of data points (e.g. 100) previous to the current point. Based on how the current point compares to these statistics, we can classify it as regular or anomalous.

1) Moving averages: In this approach, the moving average and standard deviation of the sliding window of the 100 most recent data points are calculated. If the current data point does not lie within 2.5 standard deviations of the sliding window, it is labeled as anomalous. We determined the threshold of 2.5 standard deviation empirically.

2) Exponential averages: This approach is similar to the moving average, except that the sliding window of the 100 most recent data points is exponentially smoothed before calculating the average and standard deviation. Exponential smoothing re-weighs the data-points to ensure that the recent data is given more weight than older data. Again, we then check to see if the data point to be classified falls within 2.5 standard deviations of the weighted sliding window. If it falls outside of this range, it is labeled as anomalous.



(a) Original logs

(b) Synthetic week-long data

Fig. 2: NASA Web server logs from August 1995 and synthetic logs created to obtain a larger dataset.

3) Percentile-based thresholding: This is a simple technique which flags the current data-point as anomalous if it falls outside the 10^{th} and 90^{th} percentile of the sliding window of the 100 previous data-points.

III. Data Collection

We tested our approach with two types of test cases: (1) detecting attacks on web servers, and (2) detecting attacks on user machines. In this section we will explain how we obtained the datasets needed for both.

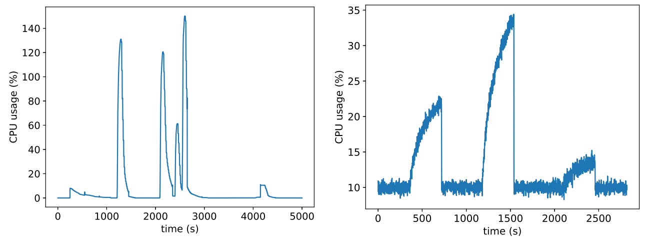
A. Network Traffic Data: NASA Web Server Logs

In order to detect web-server based attacks, we took inspiration from two instances of published month long apache web-server logs from July 1995 and August 1995 at NASA [21]. The data within the logs was parsed and converted to a time-series that represented the number of HTTP requests to the server as a function of time (in hours). Figure 2a shows a time series representation of the dataset.

We can see that there are day/night patterns from the peaks and valleys, and also weekly patterns from the higher peaks on weekdays and lower ones on weekends. Since this data set was too small to train machine learning models, we sought to synthetically generate a larger version of the dataset. We experimented with different functions and parameters and found that the following function emulates weekly cycles of the data (Note that $square(x, 0.7)$ in the equation represents a square wave with a duty cycle of 0.7):

$$100sin(7x) + 200 + 200square(x, 0.7) + 200$$

Of course, noise was added to all parameters of the function in order to simulate real world variations in the dataset. We chose a combination of a sine and square wave because the peaks and valleys of the sine wave represent the day-night fluctuations, and on-off behavior of the square wave represents weekday



(a) Original data

(b) Synthetically generated data

Fig. 3: CPU usage from Google trace and synthetically generated CPU usage data.

and weekend trends. Figure 2b shows what the generated data looks like for a 1 week window.

B. CPU and Memory Data: Google Traces

Users tend to run specific types of jobs on their machines, which consequently result in certain patterns in their resource usage data. In order to capture this, we used published user level data collected from a Google 12.5k-machine cluster. The trace spans a month-long period in May 2011 [22]. Each trace included hashed usernames (of Google engineers) and timestamps for mean CPU usage rate, canonical memory usage, and mean disk I/O time, etc., and so it was possible to extract a user level time-series for each of these metrics. Out of these available metrics, we used CPU and memory usage for our analysis, as suggested by the documentation for the traces. Figure 3a shows a sample of CPU usage for a user in the trace that spans 4500 seconds.

Interestingly, the pattern shown in Figure 3a seems to be prevalent for many users, each application run appears in the form of a peak in CPU usage, interspersed by times of inactivity. Furthermore, these patterns observed in CPU usage also manifest themselves in the memory usage data.

Similar to the previous dataset on Web-servers, we had to synthetically emulate this data in order to get a dataset large enough to train models. We used the following equation that generates one peak (representing a single application run):

$$x^{-2} + 1000log(x - 720) + 1000$$

Again, noise was added to all parameters of this function as well in order to simulate real world variations in the dataset. This specific combination of a negative exponent and logarithmic functions was chosen because the first part of each application run shows a logarithmic like increase, and the second part shows a decay that could be modeled by x^{-2}. Figure 3b shows what the generated data looks like for CPU usage of a different number of application runs.

C. Anomalous Data

To generate anomalous time-series, we varied each parameter one-by-one in the generating function in 10% increments (starting from -100% variation, to -90% variation, all the way up to -30%. We then did the same for the positive variations, i.e., we started at +30% variation, then 40% variation, going all the way up to +100% variation). We created 500 anomalous datasets per variation. Each dataset had 1440 data-points. Note that we omitted parameter variations in the range -30% to +30% as such small variations would typically be characteristic of healthy data, not anomalous data.

IV. Experiments and Results

In this section we present the results obtained for the three classification methods applied to the two test cases. All of the data generation and experimentation was conducted on an Ubuntu 16.04 LTS machine with 4 CPUs and 8GB memory.

A. Implementation Details

For our One-Class SVM classifier we used Python Scikit Sklearn library [23]. We trained the classifier with numerous instances of regular time-series data, and tested it with both regular and anomalous datasets. Our LSTM model has 1 input layer, 1 hidden layer with 4 neurons, and 1 output layer. Each neuron uses the sigmoid activation function. This network has been implemented with the Keras library [24] using the TensorFlow [25] backend. The statistical techniques are implemented using the Numpy [26] and Scipy [27] libraries in Python.

B. Classification: One-Class SVM

For One-Class SVM, features were extracted from all 500 instances of each group of anomalous datasets and also from 500 instances of regular datasets used for testing. These 1000 feature sets were then fed one-by-one to the SVM, which then labeled each one as regular or anomalous. This process was repeated for time-series spanning a number of different windows for both test cases, as outlined below.

1) Classifying Web-server logs: The classifier was tested with three windows of data for this use case: (1) daily, (2) five consecutive week days, and (3) weekly data.

2) Classifying user application runs: Here, we train the SVM on 2 windows: datasets representing (1) one application run, and (2) multiple consecutive application runs.

We train separate models for each of dataset window (for both web server data and application runs), and each model is trained with 1000 datasets of 1440 data points each. For instance, with daily web server data, each model is trained with 1000 generated datasets, with each dataset representing one day's web server traffic. We also ensured that all 1000 datasets had sufficient variation by varying each parameter (amplitude, offset, period, etc.) in the generating function from 0-30% (parameters were varied randomly). Each group of testing datasets included 500 instances of regular data (healthy, without anomalies), and 500 instances of anomalous data. The anomalous datasets were generated by systematically varying the same parameters of the generating function (amplitude, period, offset, etc.) by more than 30%, as outlined in section III C.

Figure 4 shows the F1-score as a function of variations in each parameter for the one day window of web server

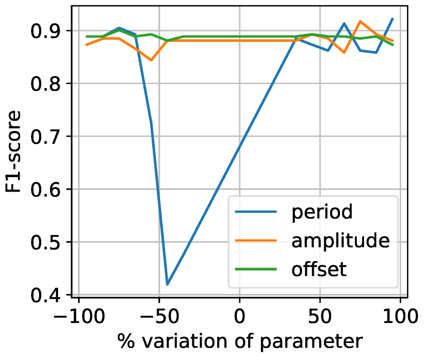


Fig. 4: One-Class SVM results: 1 Day's web server traffic.

data. The F1-scores were computed according to the following equation:

$$F1\,Score = \frac{Precision * Recall}{Precision + Recall} \tag{1}$$

where Precision and Recall are defined as follows:

$$Precision = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Positives} \tag{2}$$

$$Recall = \frac{\#True\ Positives}{\#True\ Positives + \#False\ Negatives} \tag{3}$$

We see that typically, the F1-scores increase as the variation in the parameters deviates more from the normal range. This gives most graphs somewhat of a "U" shaped curve. However, interestingly, two of the parameters (offset and amplitude) generated near-perfect F1-scores of approximately 0.9 for the web server data spanning one day (Figure 4), regardless of how much variation there was in those parameters. This could perhaps happen because the one day pattern was simple enough for even subtle variations to be spotted easily. We also see that there are some differences in trends seen for each parameter. This is because the features extracted from the time-series capture variations in each parameter to different extents.

C. Regression: LSTMs

We first trained the LSTM with regular time-series data to learn the legitimate behavior, and then tested with anomalous data. To generate the anomalous time-series, each of the 500 instances from each group of anomalous datasets was appended to a regular time-series. So in essence, the first part of each of the testing time-series data was regular, and the latter, appended part, was anomalous. The LSTM was then trained on the regular part of each time-series, and asked to predict the remainder of the time-series. This forecast reflected the patterns seen in the regular training data. And so, for each predicted data point within the remainder of the time-series, the deviation between the corresponding data point in the actual series is measured. If this deviation surpassed a certain threshold (empirically chosen to be 10% for these experiments), then we labeled the actual measured datapoint as anomalous. Figure 5 shows the percentage of datapoints classified as anomalous for a sample anomaly case. Here, we see that the LSTM result shows an "inverted U" shaped trend, which is expected as the anomalies are more likely to be detected if they are more conspicuous. The percentage of points identified as anomalous is over 90% for 80-100% variation in the parameters and, as expected, gets much lesser than 20% when the anomalous data is similar to the legitimate data.

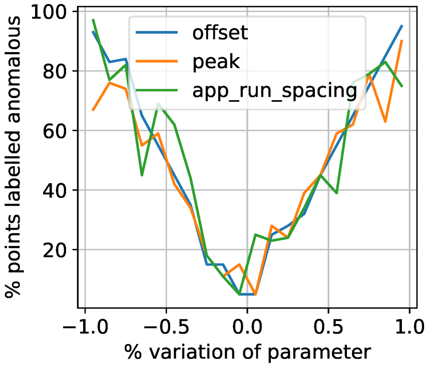
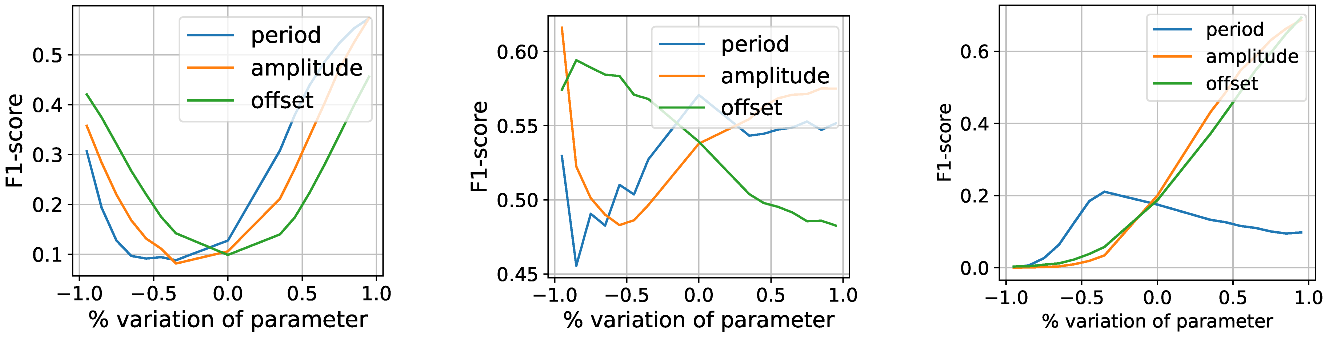


Fig. 5: LSTM result: CPU usage for 1 user application run.

D. Statistical Approaches

Within statistical approaches, each testing time-series was generated the same way as they were in LSTMs – by appending an anomalous time-series to a regular time-series. During testing, each data-point in the appended anomalous time-series was subject to classification. This classification was based on some statistics (moving average, exponential average, and percentile thresholds) computed on the previous 100 datapoints in the time series. In Figure 6, we show the F1-scores for a select few anomalous cases. Precision and recall (the two metrics required to obtain an F1-score) were computed based on the percentage of false positives, false negatives, true positives, etc. (as shown in Eq. 2 and Eq. 3) that the statistical approaches caught within each anomalous time-series.

Most of the graphs show us the expected "U" shaped trends, as the F1-scores should be lowest when anomalous datasets are not very different from the regular datasets. However, interestingly, we see a lop-sided pattern in some of the graphs, where the F1-scores are a lot higher when the argument deviation is positive as opposed to negative (or vice-versa). For instance, one sees this in the 1 Day data for percentile thresholding (Figure 6) , where the F1-scores are around 0.05 when the parameter deviations are at -100%, and are at 0.80 when the deviations are at +100%. This may be because the respective techniques that show such results might not be able to capture the essence of the data when its arguments differed in a negative sense. Lastly, the reason why the statistical approaches underperform the above two methods is that the classification of the datapoints depends entirely on one statistic. It does not take into account any patterns or long term trends while making the decision.



(a) Moving avg. 1 Day

(b) Exponential avg. Week days

(c) Percentile threshold 1 Day

Fig. 6: Moving average, exponential average and percentile thresholding results over web server logs.

TABLE I: Web Server Logs

	One day	**Consecutive weekdays**	**One week**
	F1-Score **CCR %** **avg lag**	**F1-Score** **CCR %** **avg lag**	**F1-Score** **CCR %** **avg lag**
One-Class SVM	0.87	0.89	0.74
LSTM	85%	83%	82%
Moving Avg	0.38,11.0	0.38,90.7	0.55, 42
Exponential Avg.	0.40,13.6	0.48,10.2	0.51,20
Percentile threshold	0.31,19.1	0.31,33	0.40,25

TABLE II: User Application Runs

	One application **F1-Score** **CCR %** **avg lag**	**Multiple applications** **F1-Score** **CCR %** **avg lag**
One-Class SVM	0.93	0.96
LSTM	87%	83%
Moving Avg.	0.57,15.8	0.32,18.5
Exponential Avg.	0.50,11.3	0.23,10.9
Percentile threshold	0.36,33.5	0.25,24.4

E. Aggregate Results

Table I summarizes the aggregated statistics for each approach (when averaged over all trials–including all experiments with all variations) on Web server logs and Table II shows the ones corresponding to user application runs. For LSTMs, we show the Correct Classification Rate (CCR) as a percentage, and for One-Class SVM, we show the F1-score. For statistical approaches, the first number in each cell is the F1-score and the second number represents how long it took for each metric to detect the first anomalous datapoint in the anomalous datasets for different parameter variations. For instance, a value of 20 would mean that the first anomalous point was detected 20 datapoints after the beginning of the actual anomalous section of the time-series. We see that for our statistical approaches, the lag between when the anomalies start and when they are detected is reasonable (detection happens between 10-90 data points after the anomaly ensues).

We also executed a 5-fold cross validation on our One-Class SVM model, whose results classified 21% of the data incorrectly. We observed the best results with One-Class SVM, which gave an average F1-score of 0.83 for the web server case, and 0.95 for the latter. However, while the LSTM network underperforms One-Class SVM, it is capable of learning and forecasting much more complex patterns in the time-series data. Finally, even the statistical approaches discussed have their own strengths as they do not require training/re-training and so take considerably lesser time in identifying anomalous data.

V. CONCLUSIONS AND FUTURE WORK

Our overall motivation for this research was to go beyond traditional rule based intrusion detection systems, and employ a sophisticated mathematical model specifically for VM resource usage on the cloud. Indeed, our approach seems to work well; we see the best results with LSTMs and One-Class SVM, as they are capable of learning the most complex patterns. Statistical techniques applied on a sliding window of previous data do not seem to work very well as classifying current data based on a single statistic forces us to disregard other valuable information.

The next steps involved would be to test such an approach with real resource usage data instead of generated data. Another interesting aspect comes within the statistical techniques, where we had to assign thresholds above which data points were declared anomalous. While our thresholds were fixed for all types of data, it would be interesting to see how the optimal values for these would vary based on different data. Indeed, some types of data can be a lot noisier than others, and so such data should have higher tolerance thresholds than other types of data. If implemented in commercial systems, such a program could help safeguard cloud users from a wide variety of threats, thereby saving data and resources.

ACKNOWLEDGMENT

This work has been partially funded by the Undergraduate Research Opportunities Program (UROP) at Boston University; by the National Science Foundation under Grants Numbered 1347525 and 1149232; the EU (FEDER) and the Spanish MINECO, under grant TIN 2015-65277-R; the MassTech Collaborative Research Matching Grant Program and the commercial partners of the Massachusetts Open Cloud, which include Brocade, Cisco, Intel, Lenovo, Red Hat and Two Sigma; and by the NSF grant CNS-1717634.

References

[1] D. Chronicle., “The fappening’s list of celebrities whose private pics got leaked,” Aug 2017. [Online]. Available: http://www.deccanchronicle.com/technology/in-other-news/270817/the-fappenings-list-of-celebrities-whose-private-pics-got-leaked.html

[2] A. Greenberg, “How hackers hid a money-mining botnet in the clouds of amazon and others,” Jun 2017. [Online]. Available: http://www.wired.com/2014/07/how-hackers-hid-a-money-mining-botnet-in-amazons-cloud/

[3] J. Renfeldt, “Chipotle hit with malware that stole credit cards,” Jun 2017. [Online]. Available: http://www.jr-tech.com/2017/06/12/chipotle-hit-with-malware-that-stole-credit-cards/

[4] A. Peterson, “The sony pictures hack, explained,” Dec 2014. [Online]. Available: https://www.washingtonpost.com/news/the-switch/wp/2014/12/18/the-sony-pictures-hack-explained/?utm_term=.8c5a0634122d

[5] Cloud Security Alliance, “The notorious nine: Cloud computing top threats in 2013,” Report available from www.cloudsecurityalliance.org, 2013.

[6] Netskope, “Cloud report,” Report available from www.netskope.com/netskope-cloud-report, 2015.

[7] B. Caswell, J. Beale, and A. Baker, *Snort intrusion detection and prevention toolkit*, 2007. [Online]. Available: https://www.snort.org/

[8] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, “Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks,” in *Proc. 29th Annual Computer Security Applications Conference (ACSAC)*, 2013, pp. 199–208.

[9] T. Nelms, R. Perdisci, and M. Ahamad, “ExecScent: Mining for new command-and-control domains in live networks with adaptive control protocol templates,” in *Proc. 22nd USENIX Security Symposium*, 2013.

[10] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, “Detection of early-stage enterprise infection by mining large-scale log data,” in *Proc. 25th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015.

[11] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *Security and Privacy (SP), 2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 305–316.

[12] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, “Intrusion detection by machine learning: A review,” *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[13] T. Ahmed, B. Oreshkin, and M. Coates, “Machine learning approaches to network anomaly detection,” in *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*. USENIX Association, 2007, pp. 1–6.

[14] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, “Network anomaly detection with the restricted boltzmann machine,” *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[15] T. Lane and C. E. Brodley, “An application of machine learning to anomaly detection,” in *Proceedings of the 20th National Information Systems Security Conference*, vol. 377, 1997, pp. 366–380.

[16] H. Nguyen, Y. Tan, and X. Gu, “PAL: Propagation-aware anomaly localization for cloud hosted distributed applications,” in *Proc. ACM Workshop on Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques (SLAML)*, 2011.

[17] D. Dean, H. Nguyen, and X. Gu, “UBL: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems,” in *Proc. ACM International Conference on Autonomic Computing (ICAC)*, 2012.

[18] D. J. Dean, H. Nguyen, P. Wang, X. Gu, A. Sailer, and A. Kochut, “Perfcompass: Online performance anomaly fault localization and inference in infrastructure-as-a-service clouds,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2015.

[19] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proc. 15th ACM Workshop on Hot Topics in Networks (HotNets)*, 2016.

[20] F. Farahnakian, P. Liljeberg, and J. Plosila, “Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning,” in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE, 2014, pp. 500–507.

[21] J. Dumoulin, “Nasa http webserver logs.” [Online]. Available: http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

[22] C. Reiss, J. Wilkes, and J. L. Hellerstein, “Google cluster-usage traces: format + schema,” Google Inc., Mountain View, CA, USA, Technical Report, Nov. 2011, revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078195

[24] F. Chollet *et al.*, “Keras,” https://github.com/fchollet/keras, 2015.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[26] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[27] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001, [Online; accessed today]. [Online]. Available: http://www.scipy.org/