Online Malware Detection in Cloud Auto-scaling Systems Using Shallow Convolutional Neural Networks

Mahmoud Abdelsalam, Ram Krishnan, Ravi Sandhu

Online Malware Detection in Cloud Auto-Scaling Systems Using Shallow Convolutional Neural Networks

Mahmoud Abdelsalam[1,2], Ram Krishnan[1,3], and Ravi Sandhu[1,2]

[1] Institute for Cyber Security and Center for Security and Privacy Enhanced Cloud Computing
[2] Department of Computer Science
[3] Department of Electrical and Computer Engineering
University of Texas at San Antonio, San Antonio, Texas, USA
{mahmoud.abdelsalam, ram.krishnan, ravi.sandhu}@utsa.edu

Abstract. This paper introduces a novel online malware detection approach in cloud by leveraging one of its unique characteristics—auto-scaling. Auto-scaling in cloud allows for maintaining an optimal number of running VMs based on load, by dynamically adding or terminating VMs. Our detection system is online because it detects malicious behavior while the system is running. Malware detection is performed by utilizing process-level performance metrics to model a Convolutional Neural Network (CNN). We initially employ a 2d CNN approach which trains on individual samples of each of the VMs in an auto-scaling scenario. That is, there is no correlation between samples from different VMs during the training phase. We enhance the detection accuracy by considering the correlations between multiple VMs through a sample pairing approach. Experiments are performed by injecting malware inside one of the VMs in an auto-scaling scenario. We show that our standard 2d CNN approach reaches an accuracy of $\simeq 90\%$. However, our sample pairing approach significantly improves the accuracy to $\simeq 97\%$.

Keywords: Security · Auto-Scaling · Online Malware Detection · Cloud IaaS · Deep Learning · Convolutional Neural Networks

1 Introduction

Cloud computing characteristics [15] enable novel attacks and malware [5,9–12,22]. In particular, cloud has become a major target for malware developers since a large number of Virtual Machines (VMs) are similarly configured. Automatic provisioning and auto configuration tools have led to the widespread use of auto-scaling, where VMs scale-in/out on demand. Applications utilizing auto-scaling architectures[4] is one of the most prevalent in cloud. As a result, a malware that infects one VM can be easily reused to infect other VMs that are similarly configured or imaged. To that end, cloud has become a very interesting target to most attackers.

[4] Amazon architecture references. https://aws.amazon.com/architecture/

In malware analysis, files are scanned before execution on the actual system either through static or dynamic analysis. Once an executable/application is deemed to be benign, it executes on the system without further monitoring. Such methods often fall short in the case of cloud malware injection [11], a threat where an attacker injects a malware to manipulate the victim's VMs, because the initial scan is usually bypassed or malware is injected into an already scanned benign application. Consequentially, the need for online malware detection, where you continuously monitor the whole system for malware, has become a necessity.

Few works [1,6,17,20,21] exist in the domain of *online* malware detection in cloud. Typically, machine learning is used in online malware detection. First, a set of system features are selected and used to build a model. Then, this model is used for malware detection. Some works use system calls while others use performance metrics. Although such works target cloud systems in some sense, there is no real difference between standard online malware detection methods and cloud-specific methods except in the features selected for machine learning, where cloud-specific methods restrict the selection of features to those that can only be fetched through the hypervisor. One can argue that such works focus on malware detection in VMs running on a hypervisor.

However, what makes cloud computing powerful is the novel characteristics that they support [15] such as on-demand self-service, resource pooling and rapid elasticity via auto-scaling. In this paper, we explore malware detection approaches that can leverage specific cloud characteristics. In particular, we focus on auto-scaling. The high-level idea is that in an auto-scaling scenario, where multiple VMs are spawned based on demand, each of those VMs is typically a replica. This means the "behavior" of those VMs need to closely correspond with each other. If a malware were to be injected online into one of those VMs, the infected VM's behavior will likely deviate at some point in time. Our work seeks to detect such deviations when they occur. A sophisticated attacker can attempt to simultaneously inject malware into multiple VMs, which could induce similar behavior across those VMs, and thereby escape our detection mechanisms. This is an interesting challenge and we plan this for future work. This paper focuses on malware detection when exactly one of the VMs in an auto-scaling environment is compromised.

In terms of the approach, first, we introduce and discuss a cloud-specific online malware detection approach. It applies 2d CNN, a deep learning approach, for online malware detection by utilizing system process-level performance metrics. A 2d input matrix/image is represented as the *unique processes* $\times$ *selected features*. We assume that similarly configured VMs should have similar behavior, so we train a single model for VMs that belongs to the same group such as the group of application servers in a 3-tier auto-scaling web architecture of web servers, application servers and database servers. Next, we introduce a new approach that leverages auto-scaling. Here, we consider correlations between multiple VMs by pairing samples from pairs of those VMs. Samples collected at the same time from multiple VMs are paired and fed into CNN as a single sample.

CNN is chosen because of its simplicity and training speed as opposed to other deep learning approach (e.g. Recurrent Neural Networks). Also, for the sake of practicality, we show that even a shallow CNN (LeNet-5) trained only for a few epochs can be

effective for online malware detection. In summary the contributions of this paper are two-fold:

- We introduce a 2d CNN based online malware detection approach for *multiple* VMs.
- We improve 2d CNN by introducing a new approach by pairing samples from different VMs to accommodate for correlations between those VMs.

To the best of our knowledge, our work is the first to leverage cloud-specific characteristics for online malware detection. The remainder of this paper is organized as follows. Section 2 discusses related work on cloud-specific online malware detection. Section 3 explains the key intuition about the idea presented. Section 4 outlines the methodology including the architecture of the CNN models. Section 5 describes the experiments conducted and results. Finally, Section 6 summarizes and concludes this paper.

2 Related Work

Many research works address the problem of online malware detection using different set of features and machine learning algorithms. Some works [6, 8, 14] focus on using systems calls while others [3, 18, 19] focus on using API calls. Others [16, 23] focus on using memory features or performance counters [7].

Only a few research works address the problem of cloud malware detection since many of the standalone malware detection approaches work for detecting malware in single VMs in the cloud as well. Most, if not all, of the cloud-specific malware detection techniques falls under the online malware detection category (including anomaly detection approaches). Furthermore, they all focus on extracting features from the hypervisor since it adds another security layer.

Dawson et al. [6] focus on rootkits and intercept system calls through the hypervisor to be used as features. Their system call analysis is based on a non linear phase-space algorithm to detect anomalous behavior. Evaluation is based on the dissimilarity among phase-space graphs over time.

Wang [20] introduced Entropy based Anomaly Testing (EbAT), an online analysis system of multiple system-level metrics (e.g. CPU utilization and memory utilization) for anomaly detection. The proposed system used a light-weight analysis approach and showed a good potential in detection accuracy and monitoring scalability. However, the evaluation used did not show pragmatic and realistic cloud scenarios.

Azmandian et al. [4] propose an anomaly detection approach where all features are extracted directly from the hypervisor. Various performance metrics are collected per process (e.g., disk i/o, network i/o) and unsupervised machine learning techniques like K-NN and Local Outlier Factor (LOF) are used.

Classification of VMs is used for anomaly detection. Pannu et al. [17] propose an adaptive anomaly detection system for cloud. It focused mainly on various faults within the cloud infrastructure. Although this work is not directly addressing malware, such technique is valid for malware detection since malware can cause faults in VMs, thus worth mentioning. It used a realistic testbed experimentation comprising 362-node

cloud in a university campus. The results showed a good potential with over 87% of anomaly detection sensitivity. One of the drawbacks of this work lies within using two-class SVM. Therefore, it suffers from a data imbalance problem (where there is an imbalance of data from various classes during the training period), which led to several false classification of new anomalies.

The work by Watson et al. [21] is similar to [17] but directly addressed detecting malicious behavior in the cloud. It tried to overcome the drawbacks in [17] by using 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 the CAIDA's CoralReef[5] tool. The study shows high accuracy results; however, it uses known-to-be highly active malware that easily skew the system's resource utilization (e.g., by forking many processes).

In our earlier work [1], we showed that malware can be effectively detected using black-box VM-level performance and resource utilization metrics (such as CPU and memory utilization). Although, the work showed promising results for highly active malware (e.g., ransomware), it is not as effective for low-profile malware that would not impact black-box level resource utilization significantly. Subsequently, we introduced a CNN based online malware detection method for low-profile malware [2]. This work utilized resource utilization metrics for various processes within a VM. The method was able to detect low-profile malware with accuracy of $\simeq 90\%$. Although, this work yielded good results, it targeted a single VM much like other related works. Unlike our prior work and other related works, this paper targets malware detection when multiple VMs are running, while leveraging specific cloud characteristics such as auto-scaling.

3 Key Intuition

In classification-based process-level online malware detection methods, a machine learning model is trained on benign and malicious samples of processes where the goal is to classify a new input sample. The data collection phase, usually, works by running a VM for some time (benign phase) and then injecting a malware (malicious phase) while logging the required data. This is referred to as a single run. The data set includes multiple runs with same/different malware which is later divided into training and test data sets. In other words, given sample X at time t (X_t), the task is to compare X_t to previously seen samples of the training data set. For a single run, we deal with individual samples of a single VM. Thus, we refer to this approach as Single VMs Single Samples (SVSS) which is shown in Figure 1.

SVSS can work in an auto-scaling scenario where we have a trained model for each auto-scaling tier; however, input samples will lose some information. Note that multiple runs of a single VM is not the same as multiple VMs running at the same time. The reason depends mostly on the architecture in place. If a VM has some effects over another VM, then input samples from single VM in multiple runs will lose this

[5] CoralReef Suite: https://www.caida.org/tools/measurement/coralreef/

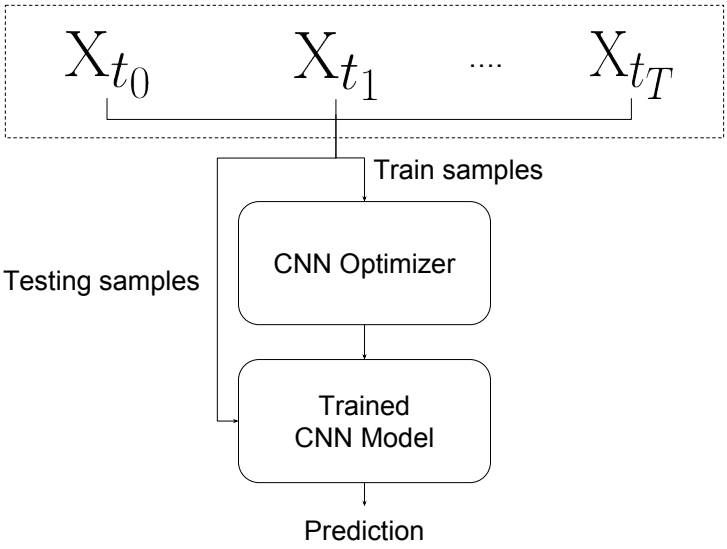


Fig. 1: Single VMs Single Samples (SVSS)

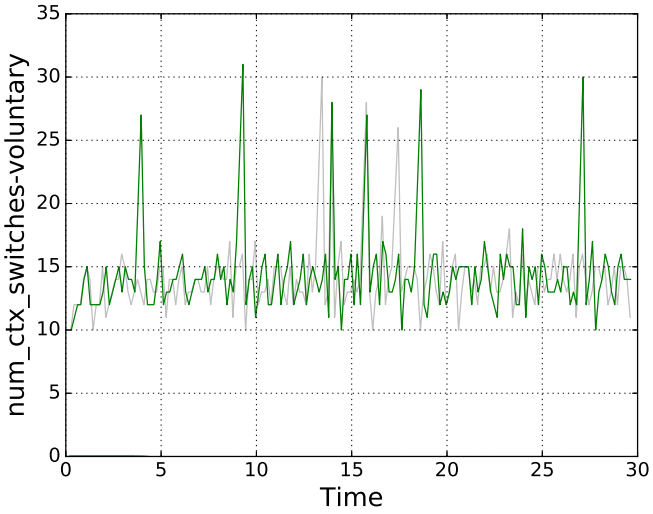


Fig. 2: Number of used voluntary context switches over 30 minutes for two different runs of the same unique process.

information. To that end, we extend SVSS and build an auto-scaling testbed where we can learn from multiple VMs running at the same time. We refer to it as Multiple VMs Single Sample (MVSS).

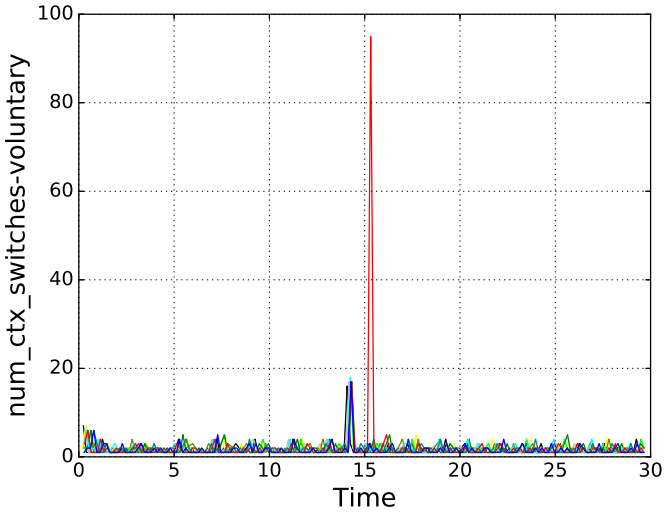


Fig. 3: Number of used voluntary context switches over 30 minutes for one run of 10 VMs in an auto-scaling scenario.

The MVSS approach, however, has a disadvantage in the context of process-level performance metrics. Processes have a very dynamic nature, meaning spikes are always happening. These spikes are mostly due to sudden events or traffic surges. For example, Figure 2 shows two different runs of the same process for the number of voluntary context switches. No malware is running inside either of the two VMs. During the training phase, two patterns will be learned, a smooth recurring up and down pattern and a pattern where there can be some spikes. During the testing phase, if either pattern is seen, it will be regarded as benign.

On the other hand, Figure 3 shows one run of the same unique process in 10 VMs (belongs to the same group of VMs in an auto-scaling scenario). VMs are running at the same time in an auto-scaling scenario. The red colored process belongs to a VM where a malware was injected. There are two major spikes in the figure. The first spike happened in the same unique process of all the VMs. If one of the processes did not have that spike and it was classified as benign, it might be a misclassification since such spike should happen to all VMs at the same time. The second spike (caused by the malware injected) is only observed in the infected VM which should be classified as malicious. In simple words, observing a noticeable enough spike by a particular process should be classified as malicious (i.e., second spike). However, sudden behavior changes can happen (i.e., first spike) and flagging an observed spike always as malicious can cause many misclassifications. As such, considering multiple VMs, spikes that are observed in a particular process in all VMs at the same time shouldn't be classified as malicious since it can be caused by any sudden change in behavior (e.g. sudden increase in the

Table 1: Process-level performance metrics

Metric Category	**Description**
CPU information	CPU usage percent, CPU times in user space & kernel space, CPU times of children processes in user space & system space.
Context switches	Number of context switches voluntary & involuntary
IO counters	Number of read requests, write requests, read bytes, written bytes, read chars, written chars
Memory information	Amount of memory swapped out to disk, Proportional set size, Resident set size, Unique set size, Virtual memory size, Number of dirty pages, Amount of physical memory, text resident set, Memory used by shared libraries
Network information	Number of received bytes, Number of sent bytes
Others	Process status, Number of used threads, Number of opened file descriptors

number requests to web server). MVSS and SVSS will lose such correlations between VMs since they learn from individual samples regardless the scenario.

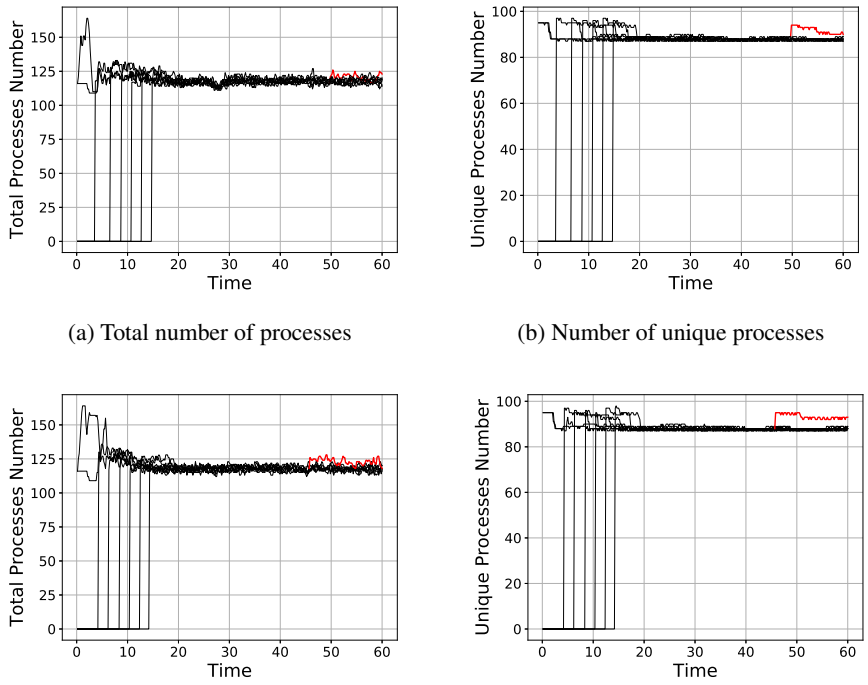
Consequentially, we introduce a new approach where the correlation of multiple VMs is utilized by pairing samples (at the same time). In other words, given sample X of VM vm_i at time t ($X_{vm_{it}}$), the idea is to compare $X_{vm_{it}}$ to previously seen paired samples of multiple VMs. We refer to this approach as Multiple VMs Paired Samples (MVPS).

4 Methodology

Detailed explanation of CNN is left out of this paper. However, it will suffice to say that CNN is a deep learning approach used extensively in image recognition. Hence, it takes 2d images as input. In our work, the first dimension represents the processes in the system and the second dimension represents the features collected for each process. Consider a sample X at a particular time t, that records n features (performance metrics) per process for m processes in VM vm, such that:

$$\mathbf{X}_{vm_t} = \begin{bmatrix} & f_1 & f_2 & \dots & f_n \\ p_1 & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_m & \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

Table 1 shows the process-level performance features which can be fetched through the hypervisor. CNN requires a specific process to remain in the same row (in the input matrix) for all inputs in a single run. This means that process ID (PID) can not simply be used directly. Processes get killed and get created frequently so a PID identifying one process might identify a different process later on. For that reason, we define a *unique process* which is identified by three elements: process name (name), command line used



(a) Total number of processes

(b) Number of unique processes

(c) Total number of processes

(d) Number of unique processes

Fig. 4: Total number of standard processes versus the number of unique processes in VMs in an auto-scaling scenario.

to execute the process (cmd) and hash of binary executable (if applicable). In addition, unique processes help in smoothing the number of processes in a highly active server since most malware creates new non-unique processes. Figures 4 (a) - (b) and (c) - (d) show two different experiments (each with a different malware) where the number of total standard processes are compared to the number of unique processes. Red portions are the start of malware execution. As shown in the figure, the total number of processes in such a highly active VM does not help much in revealing the malware behavior as opposed to the number of unique processes. Throughout this paper the terms process and unique process are used interchangeably where both refer to unique process.

4.1 Malware Detection in Multiple VMs using Single Samples (MVSS)

This is a relatively straight-forward task. We target multiple VMs in an auto-scaling scenario. Figure 5 shows the approach used in MVSS. In MVSS we have samples $X_{vm_{itk}}$ from multiple VMs running at the same time, where X is a sample of VM vm_i at time t_k. Samples from many runs are collected and are fed to the CNN optimizer where the learning process takes place. Then the trained CNN model is used for predictions.

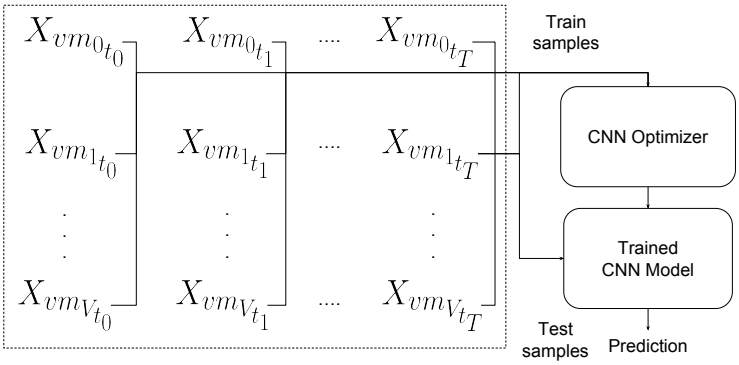


Fig. 5: Multiple VMs Single Sample (MVSS)

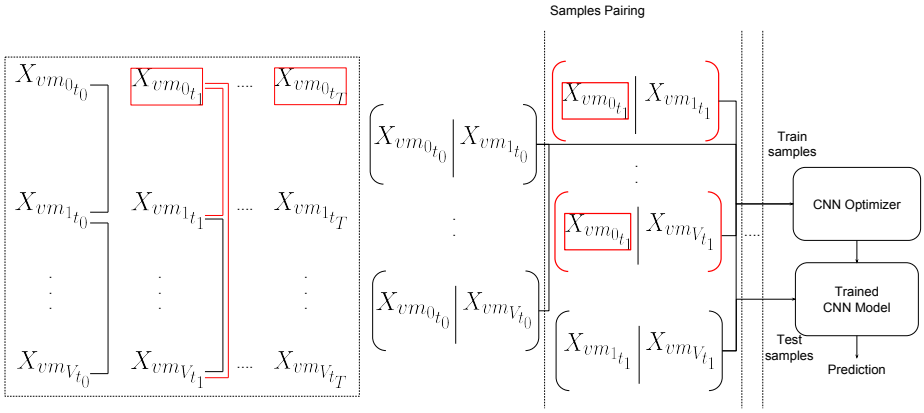


Fig. 6: Multiple VMs Paired Samples (MVPS)

4.2 Malware Detection in Multiple VMs using Paired Samples (MVPS)

The MVPS approach is inspired by the duplicate questions problem in online Q&A forums like Stack Overflow and Quora. The problem focuses on determining semantic equivalence between questions pairs. It is a binary classification problem where two questions Q_1 and Q_2 are given and the task is to determine whether they are duplicates.

Based on the aforementioned assumption that VMs that belong to the same group should behave similarly, we use the same analogy to tackle our problem. To that end, we change the formalization of our problem by using the above duplicate questions problem concept except, in our case, we are given two samples $X_{vm_{i_{t_k}}}$ and $X_{vm_{j_{t_k}}}$ from different VMs, where $X_{vm_{i_{t_k}}}$ is a 2d matrix (image in CNN terminology) that belongs to vm_i at time t_k and $X_{vm_{j_{t_k}}}$ is a 2d matrix that belongs to vm_j at the same time t_k. Figure 6 shows the pairing samples approach. Our goal is to find whether $X_{vm_{i_{t_k}}}$ and $X_{vm_{j_{t_k}}}$ are duplicates (similar). This is done by pairing the two samples

as an input to CNN. Two samples are considered similar if they are benign, whereas two samples are considered not similar if either one of them is malicious (red bordered samples are malicious).

By pairing samples, we are actually taking into account the correlations between samples of different VMs. This is due to the fact that CNN works by finding spatial correlation within images. MVPS works in an auto-scaling scenario where there are at least two VMs of the same group. Note that it is important that we only pair samples of the same time as pairing samples of different times might have completely different values if the behavior of the VMs has changed over time. For example, a web server handling one request per sec at time t_1 will have a completely different behavior than a web server handling 100 requests per sec at time t_2. Consequentially, pairing two samples taken at two different times might mislead the classifier if the behavior of the VMs changed over time.

Pairing all samples is a very time consuming operation. In addition, that will introduce a class imbalance problem since we are only infecting a single VM. Although, we believe that infecting multiple VMs is hard to occur at the exact same time in practice, not the least because a malware needs time to infect other similarly configured VMs. Like mentioned earlier we set this for future work. Consequentially, as shown in Figure 6, we pair a malicious sample with all benign samples from other machines at a particular time. On the other hand, we pair each benign sample sequentially with the sample of the following VM.

5 Experiment Setup and Results

5.1 CNN Model Architecture

A deep CNN model would require considerably larger processing power. In reality, this might not be affordable. For the sake of practicality, we chose to work with a shallow CNN. We show that even a shallow CNN can achieve near optimal results in our pairing approach. Figure 7 shows the CNN model used in this work. We chose LeNet-5 [13] CNN model. Although, it is currently by no means one of the state-of-the-art CNN models, its shallowness makes it one of the best candidates in practice. Note that in the context of online malware detection, the model might need to be trained multiple times based on the deployed workloads in place. For example, a 3-tier web architecture and a Hadoop architecture might need different trained models.

The CNN model receives a standardized 2d matrix. Lenet-5 CNN consists of 7 layers (excluding the input layer). The input layer is a 2d matrix of 120×45 (120×90 for MVPS), representing a sample of maximum 120 processes and 45 features. For empty processes (i.e., processes that do not run at the start time but might start in the future), rows are padded with zeros. The first layer is a convolutional layer with 32 kernels of size 5×5 with zero padding ending. This results in a 32 feature maps of size 120×45. The second layer is a max pool layer of size 2×2 which downsizes each dimension by a magnitude of 2, resulting in a 32 feature maps of size 60×23 (60×45 for MVPS). The third layer, another convolutional layer with 64 kernels of size 60×23, is followed by a max pool layer which results in 64 down sized feature

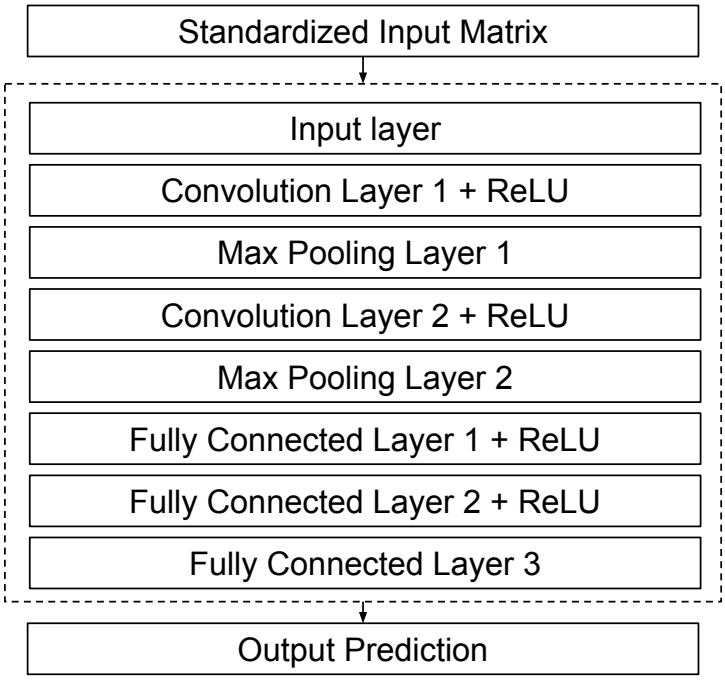


Fig. 7: CNN Model (LeNet-5)

maps of size 30×12 (30×23 for MVPS). Fifth and sixth layers are fully connected layers of size 1024 and 512, respectively. Last layer is another fully connected layer of size 2, representing prediction class (malicious or benign).

ReLU activation is used after every convolutional and fully connected layer (excluding the last fully connected layer). Adam Optimizer, a stochastic gradient descent with automatic learning rate adaptation, is used to train the model. Adam optimizer learning rate is a maximum change threshold to control how fast the learning process can be (set to $1e-5$). The optimizer works by minimizing the loss function (mean cross entropy). Random grid search is used to tune the CNN parameters (e.g., mini batch size).

5.2 Experimental Setup

Our experiments were conducted on an Openstack testbed. Figure 8 shows the 3-tier web architecture built on top of our testbed, with auto-scaling enabled on the web and application server layers. The scalability policy is based on the average CPU utilization of the total VMs of each tier (scalability group). It scales-out if the average CPU utilization is above 70% and scales-in if the average CPU utilization is less than 30%. The number of servers spawned in each tier were between 2 and 10 based on the traffic

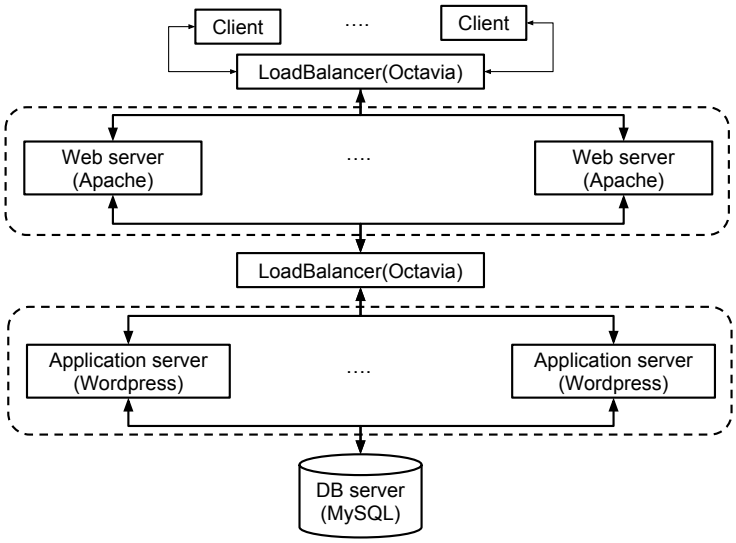


Fig. 8: 3-tier Web Application

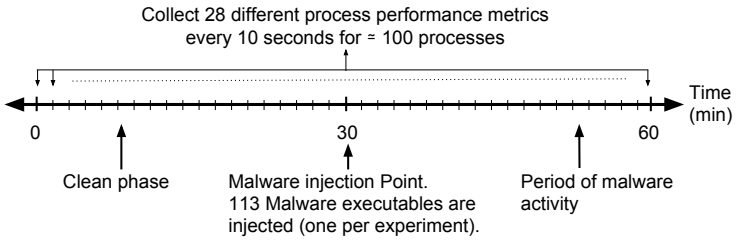


Fig. 9: Data collection overview

load. Traffic was generated based on ON/OFF Pareto distribution with parameters set according to the NS2[6] tool defaults.

The data collection process is shown in Figure 9. Each of our experiments was 1 hour long. The first 30 minutes is the clean phase. The second 30 minutes is malicious phase where a malware is injected. A set of 113 malware were used for each of the different experiments. Malware binaries were randomly obtained from VirusTotal[7]. All firewalls were disabled and an internet connection was provided to avoid any hindrance to the malware's malicious intentions. Samples were collected at 10 second intervals, so during a single experiment 360 samples were collected for one VM.

[6]NS2 manual. http://www.isi.edu/nsnam/ns/doc/node509.html
[7]VirusTotal website. https://www.virustotal.com

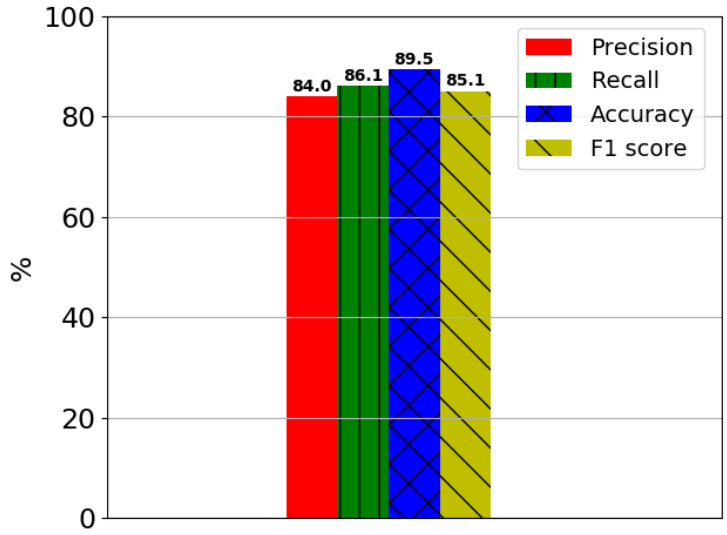


Fig. 10: Optimized MVSS CNN classifier results

5.3 Evaluation

We use four evaluation metrics.

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

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

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

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

Precision is the number of correct malware predictions. Recall is the number of correct malware predictions over the number of true malicious samples. Accuracy is the measure of correct classification. F score is the harmonic mean of precision and recall. True Positive (TP) refers to malicious activity that occurred and was correctly predicted. False Positive (FP) refers to malicious activity that did not occur but was wrongly predicted. True Negative (TN) refers to malicious activity that did not occur and was correctly predicted. False Negative (FN) refers to malicious activity that occurred but was wrongly predicted.

5.4 MVSS and MVPS Results

Like most standard machine learning classification problems, data was split into three sets: training (60%), validation (20%) and testing (20%) sets. We split on the 113 experiments to 67, 23 and 23 respectively. This ensures that validation and testing phases are

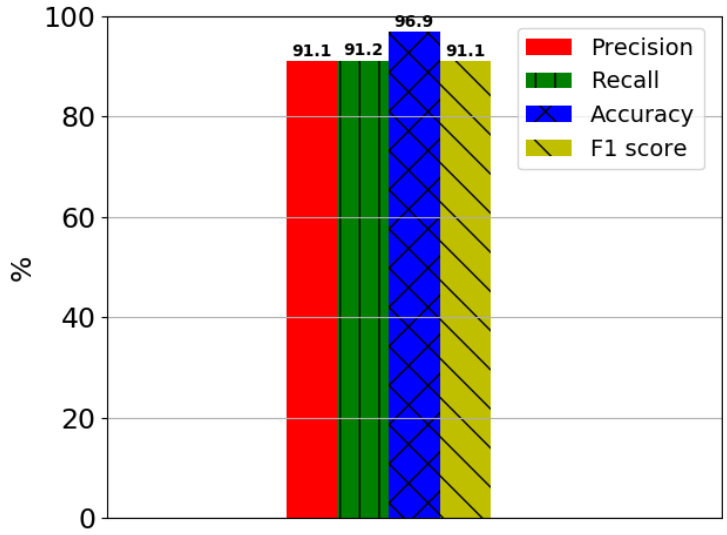


Fig. 11: Optimized MVPS CNN classifier results

exposed to unseen malware. After training the model on the training set, validation set is used to tune the model parameters as well as choosing the highest accuracy model. The model is evaluated on the validation set after each epoch and the highest accuracy model is chosen. Then the testing set is used to test the chosen model (optimized classifier).

Figure 10 shows the results of MVSS optimized classifier. The optimized classifier yields accuracy of $\simeq 90\%$ while precision, recall and fscore are $\simeq 85\%$ on the test data set. This approach achieved good results compared to the similar simple 2d CNN approach in [2]. There are two reasons for this improvement. First, increasing the number of data (113 malware experiments as opposed to 25). Second, using data from multiple VMs as opposed to a single VM; however, we still had to filter part of the data to balance our data sets (i.e., balance the ratio of benign to malicious samples).

Figure 11 shows the results of MVPS optimized classifier. There is a significant increase in the four evaluation metrics when compared to the MVSS classifier. The optimized chosen MVPS classifier had a highest accuracy of $\simeq 98.2\%$ during the validation phase. It yielded a $\simeq 96.9\%$ accuracy on the test data set. Fscore, recall and precision all jumped to $\simeq 91\%$ on the test data set. The main reason for this high improvement is that the MVPS approach finds correlations between the multiple VMs running at the same time which is very beneficial in an auto-scaling scenario.

In both cases, mini-batch size of size 64 and learning rate of $1e-5$ yielded the best results. The CNN model was trained only for 20 epochs. Note that we do not use a dropout layer (to avoid over-fitting) since it is not useful when using a shallow CNN trained for only a few epochs.

6 Conclusion And Future Work

In this paper, we introduced an online malware detection approach to leverage the behavior correlation between multiple VMs in an auto-scaling scenario. The approaches introduced used 2d CNN for malware detection. First, we introduced the MVSS method which targets multiple VMs using single individual samples. MVSS achieved good results with an accuracy of $\simeq 90\%$. Then, we introduced MVPS which targets multiple VMs using paired samples. MVPS takes the previous approach a step forward by pairing samples from multiple VMs which helps in finding correlations between the VMs. MVPS showed a considerable improvement over MVSS with an accuracy of $\simeq 96.9\%$. In the future, we plan to use different use case scenarios such as Hadoop and Containers as well as perform an analysis using different CNN models architecture. We also plan to perform an analysis to evaluate the effectiveness of ordering the processes and features in the input matrix. Finally, we plan to develop techniques to handle the situation when multiple VMs are infected simultaneously by an attacker. One direction, instead of using pairs of samples, is to use tuples of samples (3-tuple, 4-tuple or more).

Acknowledgment

This work is partially supported by NSF CREST Grant HRD-1736209, DoD ARL Grant W911NF-15-1-0518, and NSF CAREER Grant CNS-1553696.

References

1. Abdelsalam, M., Krishnan, R., Sandhu, R.: Clustering-based IaaS cloud monitoring. In: 10th IEEE CLOUD. IEEE (2017)
2. Abdelsalam, M., Krishnan, R., Sandhu, R.: Malware detection in cloud infrastructures using convolutional neural networks. In: 11th IEEE CLOUD. IEEE (2018)
3. Alazab, M., Venkatraman, S., Watters, P., Alazab, M.: Zero-day malware detection based on supervised learning algorithms of api call signatures. In: Proceedings of the Ninth Australasian Data Mining Conference-Volume 121. pp. 171–182. Australian Computer Society, Inc. (2011)
4. Azmandian, F., Moffie, M., Alshawabkeh, M., Dy, J., Aslam, J., Kaeli, D.: Virtual machine monitor-based lightweight intrusion detection. ACM SIGOPS Operating Systems Review **45** (2011)
5. Dahbur, K., Mohammad, B., Tarakji, A.B.: A survey of risks, threats and vulnerabilities in cloud computing. In: ISWSA (2011)
6. Dawson, J.A., McDonald, J.T., Hively, L., Andel, T.R., Yampolskiy, M., Hubbard, C.: Phase space detection of virtual machine cyber events through hypervisor-level system call analysis. In: 2018 1st International Conference on Data Intelligence and Security (ICDIS). pp. 159–167. IEEE (2018)
7. Demme, J., et al.: On the feasibility of online malware detection with performance counters. In: ACM SIGARCH Computer Architecture News. vol. 41. ACM (2013)
8. Dini, G., Martinelli, F., Saracino, A., Sgandurra, D.: Madam: a multi-level anomaly detector for android malware. In: International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. pp. 240–253. Springer (2012)

9. Gholami, A., Laure, E.: Security and privacy of sensitive data in cloud computing: a survey of recent developments. arXiv preprint arXiv:1601.01498 (2016)
10. Grobauer, B., Walloschek, T., Stocker, E.: Understanding cloud computing vulnerabilities. IEEE Security & Privacy **9** (2011)
11. Gruschka, N., Jensen, M.: Attack surfaces: A taxonomy for attacks on cloud services. In: IEEE CLOUD. pp. 276–279 (2010)
12. Jensen, M., Schwenk, J., Gruschka, N., Iacono, L.L.: On technical security issues in cloud computing. In: IEEE CLOUD (2009)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
14. Luckett, P., McDonald, J.T., Dawson, J.: Neural network analysis of system call timing for rootkit detection. In: 2016 Cybersecurity Symposium (CYBERSEC). pp. 1–6. IEEE (2016)
15. Mell, P., Grance, T., et al.: The nist definition of cloud computing (2011)
16. Ozsoy, M., Donovick, C., Gorelik, I., Abu-Ghazaleh, N., Ponomarev, D.: Malware-aware processors: A framework for efficient online malware detection. In: High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on. pp. 651–661. IEEE (2015)
17. Pannu, H.S., Liu, J., Fu, S.: Aad: Adaptive anomaly detection system for cloud computing infrastructures. In: Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on. pp. 396–397. IEEE (2012)
18. Pirscoveanu, R.S., Hansen, S.S., Larsen, T.M., Stevanovic, M., Pedersen, J.M., Czech, A.: Analysis of malware behavior: Type classification using machine learning. In: Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), 2015 International Conference on. pp. 1–7. IEEE (2015)
19. Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., Yagi, T.: Malware detection with deep neural network using process behavior. In: COMPSAC. vol. 2. IEEE (2016)
20. Wang, C.: EbAT: online methods for detecting utility cloud anomalies. In: Proc. of the 6th Middleware Doctoral Symp. ACM (2009)
21. Watson, M.R., et al.: Malware detection in cloud computing infrastructures. IEEE TDSC **13** (2016)
22. Xiao, Z., Xiao, Y.: Security and privacy in cloud computing. IEEE Communications Surveys & Tutorials **15** (2013)
23. Xu, Z., Ray, S., Subramanyan, P., Malik, S.: Malware detection using machine learning based analysis of virtual memory access patterns. In: 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE (2017)