Reducing Generalization Error Using Autoencoders for The Detection of Computer Worms

Nelson Ochieng[1]*, Waweru Mwangi[2] and Ismail Ateya[1]

[1]Strathmore University, Kenya,
[2]Jomo Kenyatta University of Agriculture and Technology, Kenya
**nochieng@strathmore.edu*

ABSTRACT

This paper discusses computer worm detection using machine learning. More specifically, the generalization capability of autoencoders is investigated and improved using regularization and deep autoencoders. Models are constructed first without autoencoders and thereafter with autoencoders. The models with autoencoders are further improved using regularization and deep autoencoders. Results show an improved in the capability of models to generalize well to new examples.

Keywords: Computer worm detection, generalization capability of machine learnings, autoencoders, deep autoencoders, regularization

1. INTRODUCTION

Computer and network resources are important for the operational effectiveness and efficiency of organizations. These resources may also be leveraged for competitiveness. A number of critical threats exist that could hamper these benefits. The most prevalent threats and attacks reported by pre-eminent security organizations include worms, botnets, Trojans, ransomware, spam, phishing, web attackers and crypto-miners [1].

This study has as its scope computer worm detection using machine learning in an organization’s network. More specifically, the study investigates and reduces generalization error of the detection model using autoencoders. In [2], they defines a computer worm as a process that can cause a (possibly evolved) copy of it to execute on a remote computational machine. Worms self-propagate across computer networks by exploiting security or policy-flaws in widely used network services. Unlike computer viruses, worms do not require user intervention to propagate nor do they piggy-back on existing files. Their spread is very rapid [3,4] with the ability to infect as many as 359,000 computers in under 14 hours, or even faster. Computer worm defence involves prevention and detection. Prevention may not always be wholly possible due to inherent vulnerability in all systems. The approach this study takes is to profile and detect attacks. Once detection has happened other mitigation measures may be taken.

A number of approaches for computer worm detection have been reviewed in literature. These include content-based signature schemes, anomaly detection schemes and behavioral-signature based detection. Approaches that utilized machine learning have particularly yielded good results [5,6,7,8,9,10,11,12]. Machine learning has been defined as a field of study that gives computers the ability to learn without being explicitly programmed [13]. The use of machine learning can help correlate events, identify patterns and detect anomalous behavior to improve the security posture of any defence program [14]. The studies that are most similar to the present study are here reviewed.

There exist a few studies that are most similar to the present work [15,16, 17]. In [15], they use auto-encoders for dimensionality reduction and feature extraction. Auto-encoders are unsupervised learning algorithms and do not require labels for training. Three Deep Neural Networks (DNN) were used with the features extracted from the Auto- encoders and an accuracy of 99.21% obtained. The present study deviates from this approach by using the Auto-encoders to achieve generalization of the model and not accuracy alone.

Another study that also comes close is that by [16] who propose a deep learning architecture using stacked auto-encoders with input being windows portable executable programming interface (API) calls extracted from the windows portable executable (PE) files. It uses unsupervised feature learning then supervised fine-tuning. The model outperforms Artificial Neural Networks (ANN), Support Vector Machines (SVM), Naive Bayes (NB) and Decision Trees (DT). Again, generalization error is not investigated. In [17], they use deep belief networks (DBNs) implemented with a deep-stack of de-noising auto-encoders generating an invariant compact representation of the malware behavior. The study claims 98.6% accuracy in classifying new malware variants. As before, generalization error is not investigated. Other studies use slightly different approaches. Then, in [18] they propose using two DNNs with the first limited to only determining if the traffic is normal or suspicious and the second classifying the traffic as a multi-class problem.

In [19], they use convolution neural networks for detection based on image similarity and claims an accuracy of 98%. [20] investigate the class imbalance problem with methods like random sampling and cluster-based sampling. They use auto-encoders for dimensional reduction. In their work, Random Forest outperforms deep neural networks with a claimed accuracy of 99.78%. Then, in [21] they train convolutional neural networks (CNN) translating the malware classification into an image classification. An accuracy of 99.97% is claimed. In [22], they trains a CNN on dynamic behavioral features of the PE files (n-grams used to create images) to detect and classify obscure malware. An accuracy of 97.97% is claimed. In [23], they use SVM to perform clustering as well as dimensionality reduction. In [24], they use Independent component analysis (ICA) to separate features extracted from network traces into two estimated distributions of malware and benign traffic. In [25], they uses K-Means clustering algorithm and thereafter performs boosting using Genetic Algorithm. In [26], they explains that the fundamental goal of machine learning is to generalize beyond the examples in the training set. It would be useful to investigate the generalization capability of the learners. Last, in [27], they discusses a number of ways of reducing the generalization error including stopping the training as soon as performance on a validation dataset starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization (sparse autoencoder), and drop out among others. Dropout refers to dropping out units (visible and hidden) in a neural network. The choice of which units to drop is random. They explain that dropout can be interpreted as a way of regularization by adding noise to a neural network’s hidden units. Other ways of reducing the generalization error include limiting the number of nodes in the hidden layer of the network (undercomplete autoencoder) and adding random noise to the inputs and letting the autoencoder recover the original noise free data (denoising autoencoder).

2. MATERIALS AND METHODS

2.1 DATASETS

The datasets used for the experiments were requested and obtained from the University San Diego California Center for Applied Data Analysis (USCD CAIDA). The center operates a network telescope that consists of a globally rooted /8 network that monitors large segments of lightly used address space. There is little legitimate traffic in this address space hence it provides a monitoring point for anomalous traffic that represents almost 1/256th of all IPv4 destination addresses on the Internet.

Two sets of datasets were requested and obtained from this telescope. The first is the Three days of Conficker Dataset [28] containing data for the three days between November 2008 and January 2009 during which Conficker worm attack (Emile, 2009) was active. This dataset contains 68 compressed packet capture (pcap) files each containing one hour of traces. The pcap files only contain packet headers with the payload having been removed to preserve privacy. The destination IP addresses have also been masked for the same reason. The other dataset is the Two Days in November 2008 dataset [29] with traces for the 12th and 19th November 2008, containing two typical days of background radiation just prior to the detection of Conficker.

The datasets were processed using the CAIDA Corsaro software suite, a software suite for performing large-scale analysis of trace data. The raw pcap datasets were aggregated into the FlowTuple format. This format retains only selected fields from captured packets instead of the whole packet, enabling a more efficient data storage, processing and analysis. The 8 fields are source IP address, destination IP address, source port, destination port, protocol, Time to Live, TCP flags, IP packet length and Value which represents the number of packets in the interval whose header fields match this FlowTuple key. The instances in the Three Days of Conficker dataset have been further filtered to retain only instances that have a high likelihood of being attributable to Conficker worm attack. In [29], they discusses Conficker's TCP scanning behavior (searching for victims to exploit) and indicates that it engages in three types of observable network scanning via TCP port 445 or 139 (where the vulnerable Microsoft software Windows Server service runs) for additional victims. The vulnerability allowed attackers to execute arbitrary code via a crafted RPC request that triggers a buffer overflow. These include local network scanning where Conficker determines the broadcast domain from network interface settings, scans host nearby other infected hosts and random scanning. Other distinguishing characteristics include TTL within reasonable distance from Windows default TTL of 128, incremental source port in the Windows default range of 1024-5000, 2 or I TCP SYN packets per connection attempt instead of the usual 3 TCP SYN packets per connection attempt due to TCP's retransmit behavior.

2.2 MACHINE LEARNING EXPERIMENTS

Pandas library [30] was used for the machine learning experiments. The number of instances was originally 2000 with the variables as 130. These variables included source port, destination port, protocol, time to live (TTL), IP packet length, Value (the number of instances whose signature are similar), packet source country, well known source and destination ports, among others. The variables had integer values. The examples were then split into train and test sets using the train_test_split function with the proportion of 0.33 and stratify sampling. For modeling, the keras library [31] was used. The first model was built using an input dimension of 130 and ReLu activation function (Rectified Linear Activation) and 9 nodes. For the output layer the model had 130 neurons and the sigmoid function for the activation. The model type chosen was Sequential. This was an easy way to build a model and allowed building the model layer by layer. Dense was the layer type used. Dense is a standard layer type that works for most cases. In a dense layer, all nodes in the previous layer connect to the nodes in the current layer. The activation function allows the model to take into account nonlinear relationships. For the loss function, binary_crossentropy was used. The metrics used was accuracy. For training the model, the fit () function was used with the following parameters: training dataset, validation dataset, 50 epochs and a batch size of 32. The training and validation loss models were then plotted. The second model to be built was with activity regularization included. L1 regularization was used. The regularizers class from keras library was used. ReLu and sigmoid activations were used for the layers. For model compilation, binary_crossentropy loss function was used together with adam optimizer and accuracy metrics. The training and validation loss models were again plotted. The third model built was using deep autoencoders. The input dimension was still 130 but with 9,6,4 nodes for the hidden layers and 130 nodes for the output layer. The ReLu and sigmoid activation functions were again used. The results are presented and discussed in Section 3.

3. RESULTS AND DISCUSSION

Before using autoencoders, the neural network model plotted was as shown in Figure 1. Models were then built using autoencoders to attempt to improve the generalization ability. The second model built without using regularization or deep autoencoders gave the result as illustrated in Figure 2.

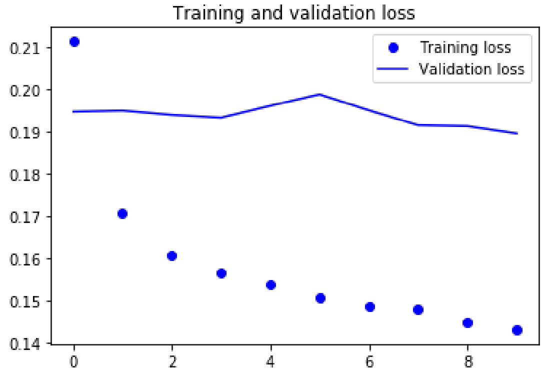

FIGURE 1. Training and validation loss for the neural network model without regularization

The validation loss curve drops and then rises again. This shows over-fitting. The model does not generalize well to new examples.

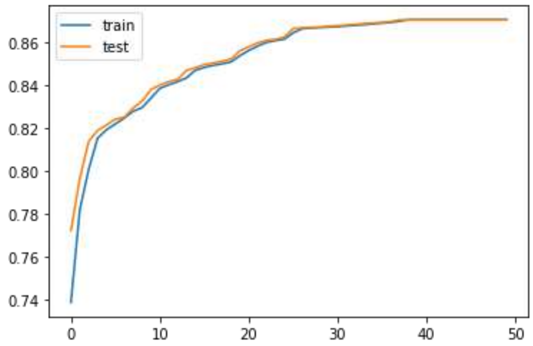

FIGURE 2. Train and validation accuracy plots for shallow autoencoder

The accuracies reported were high for both training and validation. The validation fluctuations were narrowed hence an improved model. The number of epochs before which the accuracy was flat is about 5. With sparsity constraint, the model was slightly smoothened out. The results reported were as shown in Figure 3.

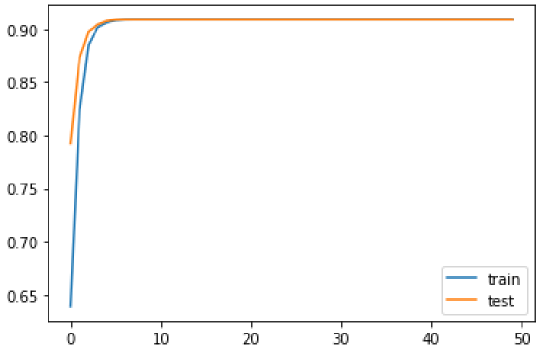

FIGURE 3. Train and validation accuracies for a model with sparsity constraint

The last model developed was one with deep autoencoders. The results are as illustrated in Figure 4. Very high accuracies were realized and the train and test curves were almost in sync. The generalization capability of the model is hence greatly improved.

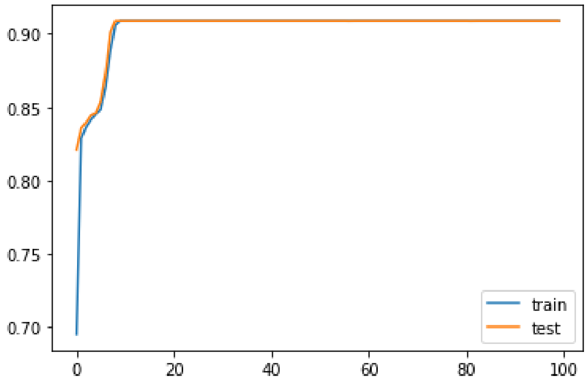

FIGURE 4. Training and Test accuracies for deep autoencoder

4. CONCLUSION

This study sought to investigate and improve the generalization capability of autoencoders for the detection of computer worms. A literature discussion of similar studies was first presented. It was evident that many studies had left out the critical element of generalization performance capability hence the need for this study. Models were developed first with no autoencoders and later with autoencoders and with sparsity constraints and also with deep autoencoders. Results presented show improved model performance in terms of generalization. Future studies can be done to compare models based on a combination of parameters such as accuracy and generalization ability. In addition, more measures to improve generalization ability can be investigated and empirical evidence adduced.

ACKNOWLEDGEMENTS

This work was supported by the African Center for Technology Studies, ACTS and IBM as part of a postdoctoral research fellowship.

REFERENCES

[1] Defending against today's critical threats, CISCO Cybersecurity Series. Threat Report, [online] 2020, www.cisco.com/go/securityreports (Accessed: 5th April, 2020)

[2] D. Ellis, "Worm Anatomy and model," in *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, 2003.

[3] D.Moore, C. Shannon, et al., "Code red: a case study on the spread of victims of an internet worm," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pp. 273-284, 2002

[4] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, "Inside the slammer worm," *IEEE security & privacy*, 99 (4), pp. 33-39, 2003

[5] M.G. Shultz, E. Eskin, F. Zadok, S.J. Stolfo, "Data Mining methods for detection of new malicious executables," *In Proceedings 2001 IEEE Symposium on Security and Privacy*, 2001

[6] J.Z. Kolter, M.A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, 2006

[7] P. Vinod, V. Laxmi, M.S. Gaur, G. Chauhan, "Detecting malicious files using non-signature-based methods," *International Journal of Information and computer security*, 2014

[8] J. Bai, J. Wang, G. Zou, "A malware detection scheme based on mining format information," 2014

[9] M. Alazah, S. Venkatranan, P. Watters, "Zero-day malware detection based on supervised learning algorithms of API call signatures," *Data mining and analysis proceedings in the 9th Australasian data mining conference*, 2020

[10] S. Muazzam, W. Morgan and L. Joohan, "Detecting Internet worms using data mining techniques," *Journal of Systematics Cybernetics and Informatics*, 2009

[11] R. Benchea and D. T. Gavrilut, "Combining restricted boltzmann machine and oneside perceptron for malware detection," *Springer International Publishing*, 2014

[12] N. Ochieng, W. Mwangi and I. Ateya, "Optimizing Computer Worm Detection Using Ensembles," *Security and Communication Networks*, 2019.

[13] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210-229, 1959

[14] A. L. Buckzak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18(2), pp. 1153-1176, 2015

[15] H. Rathore, S. Agarwal, S.K. Sahay and M. Sewak, "Malware detection using machine learning and deep learning,". *In International Conference on Big Data Analytics, Springer, Cham*, pp. 402-411, 2018

[16] W. Hardy, L. Chen, S. Hou, Y. Ye, Y and X. Li, "DL4MD: A deep learning framework for intelligent malware detection,". *In Proceedings of the International Conference on Data Mining (DMIN)*, pp. 61, 2016

[17] O.E. David and N.S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," *In 2015 International Joint Conference on Neutral Networks (IJCNN)*, pp. 1-8, 2015

[18] H.H. Al-Maksousy, M.C. Weigle and C. Wang, "NIDS: Neutral Network based Intrusion Detection System," *International Symposium on Technologies for National Security, IEEE*, pp. 1-6, 2018

[19] R. Kumar, Z. Xiaosong, R.U. Khan, I. Ahad and J. Kumar, "Malicious code detection based on image processing using deep learning," *In Proceedings of*

the 2018 International Conference on Computing and Artificial Intelligence, ACM, pp. 81-85, 2018

[20] M. su, S.K. Sahay and H. Rathore, "An investigation of a deep learning-based malware detection system,". *In Proceedings of the 13th International Conference on Availability, Reliability and Security, ACM*, p. 26, 2018

[21] M. Kalash, M. Rochan, N. Mohammed, N.D. Bruce, Y. Wang, Y and F. Iqbal, "Malware classification with deep evolutional neural networks,". *In 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), IEEE*, pp. 1-5, 2018

[22] SL, S. D. SL and C.D. Jaidhar, "Windows malware detector using convolutional neural network based on visualization images," *IEEE Transactions on Emerging topics in Computing*, 2019

[23] M. Lòpez- Viscaìno, C. Dafonte, F. Nòvoa, D. Garabato and M. Álvarez, "Network data unsupervised clustering to anomaly detection," *Multidisciplinary Digital Publishing Institute Proceedings*, vol. 2(18), pp. 1173, 2018

[24] H. Mekky, A. Mohaisen and Z.L. Zhang, "Separation of benign and malicious network events for accurate malware family classification," *In 2015 IEEE Conference on Communications and Network Security (CNS)*, pp. 361-365, 2015

[25] A. Martin, H.D. Menèndez and D. Camacho, (2016, July). "Genetic boosting classification for malware detection," *In 2016 IEEE Congress on Evolutionary Compultation (CEC)*, pp. 1030-1037, 2016

[26]. P.M. Domingos, "A few useful things to know about machine learning," *Commun.acm*, vol. 55(10), pp. 78-87, 2012

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15(1), pp. 1929-1958, 2014

[28] CAIDA USCD Network Telescope "Three days of Conficker" - http://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml

[29] Emile Aben, "Conficker/Conflicker/Downadup as seen from the USCD network telescope," *Technical Report, Caida*, [online] 2009, https://www.caida.org/research/security/ms08-067/conficker.xml (Accessed: 5th April 2020)

[30] W. McKinney et al., "Data structures for statistical computing in python," *In Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, 2010

[31] F. Chollet et al., "Keras," (*GitHub*). [online] 2015, https://github.com/fchollet/keras (Accessed: 6th April, 2020)