Detecting Malicious Assembly with Deep Learning

M. Santacroce[1], Daniel Koranek[2], David Kapp[2], Anca Ralescu[1], and R. Jha[1]
[1]Department of Electrical Engineering and Computer Science
University of Cincinnati, Cincinnati, Ohio, 45221, United States
[2] Air Force Research Laboratory, WPAFP, Ohio
Email: santacml@mail.uc.edu; Rashmi.jha@uc.edu

***Abstract*— We present a novel method for detecting malware at the assembly level using deep, sequential, convolutional neural networks accompanied by techniques for assembly processing. Relatively significant results are achieved, showing the burden of malware detection could be offloaded to a neural network without the preprocessing of current methods.**

I. Introduction

Cyber security is of the utmost importance in the present day, and with the advent of deep neural networks and an abundance of sample data on the internet, many studies are being conducted to identify malware through machine learning. A traditional route to take is to abstract or simplify the malware code using characteristics like the system calls made to the operating system kernel. While these strategies achieve competitive performance, their implementation would require significant pre-processing and processing time, as do commercial anti-malware programs. Additionally, the malware detection process itself is vulnerable if the malware is able to prevent the analysis from occurring or avoid it altogether [1].

Improvements to malware detection systems could thus consist of detection at the assembly level, running on hardware that operates parallel to the central processor. Little work exists in this direction mainly because of computational burden. Recent work showed that building neural networks for classifying extremely long sequences of byte code as benign or malicious can achieve competitive performance [2]. The byte code, however, comes from executables and is open to the same issues as previously stated. Assembly code is much closer to the machine code running on the metal, while byte code includes substantially more information about the program as it is meant to be interpreted and compiled for various processors.

This study investigates the use of deep learning methods, at assembly code and possibly machine code level, opening the doors to malware detection with specialized chips working parallel to processors. New methods such as eliminating unnecessary tokens and using max-pooling as a form of down-sampling, are introduced.

II. Approach

The format of the input to the neural network is as important as the network itself. 80 samples of malware were obtained from [3], and 80 samples of benign software were obtained by scanning the C:\ drive of a Windows computer and randomly selecting executables, which were then disassembled using the Linux utility "objdump".

Once the input is disassembled, the next step was to break up the input into arrays of tokens using automata. All hexadecimal numbers were replaced with the same token in order to obtain a workable vocabulary. The same treatment was used for the declaration of functions. Following tokenization, each unique token receives a random integer value. The resultant vector per file was 45500 lines of 20 tokens per line, using 0s as padding. Tokenization would be possible in hardware although difficult, however, results suggest using raw machine code is possible.

Next was deciding how the network would capture the context of the instructions. Natural language processing often uses various embeddings [4],[5], to solve this issue. The most successful strategy attempted was to use max-pooling with window size 10 as the first layer of the network followed by a traditional embedding.

The convolutional neural network used was built in Keras [6], with global average pooling in the final layer as a regularizer, to prevent overfitting [7]. The network is composed of the max pooling and embedding, a convolutional layer of 200 3 x 3 filters, and the hyperbolic tangent activation function, followed by a max pooling layer of 3 x 3 windows. These two layers are repeated, followed by a global average pooling layer, and wrapped up with a densely connected layer.

III. Results

The dataset was split into 120 training and 40 validation samples. Binary cross-entropy was used as a loss function (1 for malware, 0 for benign), and the network was trained for 50 epochs. At epoch 43, 100% of the training data was classified correctly, and 90% of the validation set was classified correctly, with 18 True Positives, 3 False Positives, 17 True Negatives, and 2 False Negatives. These results show great performance as False Negatives are the largest concern in classifying malware.

While the network is not ready for commercial use, these results display a clear correlation in classifying malware using the network. It is likely that the network is overfitting to the training data, and a larger dataset would greatly help.

IV. Microsoft Malware Challenge

To ensure there was not inherent bias in the dataset generated, and to cover a wider variety of test cases, the network was also run on a dataset provided by the Microsoft Malware Classification Challenge [8]. The dataset provides many thousands of samples of 9 classes of Malware and no benign samples.

For this dataset, raw machine code was carefully extracted from samples and used directly as the input without tokenization. Various permutations of the previous network architecture were tried and a maximum accuracy of 86% accuracy was achieved on 500 samples.

While this method is not competitive to current results in this domain, the network achieves some success, proving the concept has merit. The drop in accuracy is likely because classifying type of malware is a different task than classifying benign or malicious in general, and there is much room for optimization of the network. Additionally, using the raw machine code instead of tokenizing the assembly code greatly adds credibility to possibly making the network in hardware.

V. Conclusions

The presented work indicates that the proposed techniques of malware identification can be implemented at the assembly level, providing a more robust solution for securing computing devices against malware. This report opens the doors for further research in this topic area. The small size is likely a large factor in the relatively low accuracy reached by the network. The reasoning for its use is the computational limitations of the system running the network for training.

The highest priority for future work is obtaining a much larger dataset and trying larger or deeper neural network architectures. Additionally, a plethora of techniques exist for embeddings and sequence processing that can be tried. Despite the small dataset and relatively small architecture, however, a clear distinction can be seen between benign software and malware when using deep, sequential, convolutional neural networks. If the techniques presented in this paper are developed to their fullest potential it may yield malware-detecting hardware that can operate parallel to a CPU.

Acknowledgment

The work is currently supported by National Science Foundation under ECCS 1556294, and CNS-1556301 awards.

References

[1] Xu Chen, J. Andersen, Z. M. Mao, M. Bailey and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN),* Anchorage, AK, 2008, pp. 177-186.

[2] E. Raff1, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, “Malware Detection by Eating a Whole EXE”, arXiv:1710.09435 [stat.ML], Oct. 2017.

[3] Dasmalwerk.eu. (2018). *DAS MALWERK*. [online] Available at: http://dasmalwerk.eu [Accessed 28-Mar.-2018].

[4] C., Ronan and J. Weston. “A unified architecture for natural language processing: deep neural networks with multitask learning.” *ICML* (2008).

[5] D. Tang, "Learning sentiment-specific word embedding for twitter sentiment classification." *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2014.

[6] F. Chollet, "Keras Documentation", *Keras.io*, 2018. [Online]. Available: https://keras.io. [Accessed: 28- Mar- 2018].

[7] M. Lin, Q. Chen , S. Yan, “Network in Network”, arXiv:1312.4400 [cs.NE], Dec. 2013

[8] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, M. Ahmadi, “Microsoft Malware Classification Challenge”, arXiv:1802.10135 [cs.CR], Feb. 2018