A Hybrid System for Malware Detection on Big Data

Article

Accepted version

A. De Paola, S. Gaglio, G. Lo Re and M. Morana

In Proceedings of the IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)

Publisher: IEEE

A Hybrid System for Malware Detection on Big Data

Alessandra De Paola, Salvatore Gaglio, Giuseppe Lo Re and Marco Morana
Università degli Studi di Palermo, Palermo, Italy
{alessandra.depaola, salvatore.gaglio, giuseppe.lore, marco.morana}@unipa.it

***Abstract*—In recent years, the increasing diffusion of malicious software has encouraged the adoption of advanced machine learning algorithms to timely detect new threats. A cloud-based approach allows to exploit the big data produced by client agents to train such algorithms, but on the other hand, poses severe challenges on their scalability and performance. We propose a hybrid cloud-based malware detection system in which static and dynamic analyses are combined in order to find a good trade-off between response time and detection accuracy. Our system performs a continuous learning process of its models, based on deep networks, by exploiting the growing amount of data provided by clients. The preliminary experimental evaluation confirms the suitability of the approach proposed here.**

I. Introduction

Nowadays, the accidental execution of malicious software coming from different communication channels makes IT systems constantly exposed to risks. As a consequence, malware detection represents one of the most critical issues faced by computer security scientists. One of the most promising approach followed to guarantee a high detection rate, even with the constant increase of threats, is the adoption of cloud-based solutions. Such an approach allows to remotely perform malware detection through machine learning techniques capable of analyzing huge amounts of potentially infected files. Moreover, the big data produced by a plethora of client agents can be exploited in order to progressively refine the malware detection system. The amount of such potential malware data flow depends on several factors, such as the increasing number of user devices capable of executing always more advanced applications, and the simplified malware diffusion processes through cloud services and social networks [1]. Although the possibility of exploiting such big data, on one hand, represents a great opportunity, on the other hand, enormous challenges on the scalability and performance of the malware detection system are to be faced.

Following such an idea, in this paper, we propose the adoption of a cloud-based system which exploits a big data infrastructure to gather and manage the huge amount of file to be analyzed, to drive their classification and to exploit them in a continuous learning process.

Malware detection methods, proposed in the literature, can be roughly classified in two main categories depending on whether the analysis is statically or dynamically performed [2]. The former category typically analyses only the information contained in the potentially infected file, and does not require any execution. On the contrary, the dynamic analysis observes the execution behavior by exploiting a protected environment, e.g., a "sandbox", in which the malware can be executed, without threatening a real working system. Static methods are generally faster and less resource hungry, even if they can exploit less information and are vulnerable to malicious code obfuscation. On the other hand, the detailed analysis enabled by dynamic approaches often require higher computing time and resource consumption.

Our system overcomes such limitations by combining static and dynamic analyses to obtain a good trade-off between response time, required resources, and detection accuracy. The main idea is to perform the static analysis by default and to turn to the dynamic analysis only when the former does not yield a sufficiently accurate classification. Moreover, the results produced by the dynamic analysis, together with the executable files submitted to be analyzed, contribute to update the system through a continuous learning process. In the static analysis, we adopt a lightweight pre-processing of executable files and exploit deep learning techniques to automatically extract relevant features directly from raw information.

The current static and dynamics analyses are tailored to detect Windows malware, but it is worth noticing that the proposed architecture is general and applicable to other platforms.

The remainder of this paper is structured as follow. Section II presents an overview of the related work. Section III describes the architecture of the proposed system, details the static and dynamic analysis subsystems, and the continuous learning process. Section IV reports the preliminary experimental evaluation. Finally Section V presents our conclusions and future work.

II. Related Work

A. Deep learning for malware detection

Several works which adopt deep learning to perform malware detection have been presented in the recent literature, based both on dynamic and static analysis [2].

The system proposed in [3] performs a dynamic analysis, by adopting deep learning to automatically generate the signature which represents such behavior. The authors use a deep belief network implemented through stacked denoising autoencoders, which processes the text file containing the transcription of all the events occurred during a file run, adequately converted in a binary form. The automatically-generated signature is then processed by a SVM to perform the effective classification. In [4], the sequence of system calls recorded while the

executable file is running is exploited by a Deep Neural Network (DNN) which combines convolutional layers and recurrent layers, using Long Short Term Memory (LSTM) cells to increase malware detection capabilities.

In order to overcome the limitation of dealing with less information than those obtained by dynamic analysis, some works based on static analysis perform deep learning on high-level features extracted according to a well-designed process. The authors of [5] propose to build a large vector of features, which is reduced through a random projection process. The resulting vector is then analyzed by a deep classifier which is pre-trained through a Restricted Bolzmann Machines (RBM). The reduction of the feature vectors through random projection is also adopted in [6], where JavaScript sources are processed through a 5–layer deep neural network implemented with stacked denoising autoencoders. In [7] four different sets of static features, converted in a 1024-length binary vectors and classified by a classic DNN, are used. A more complex set of features is propose in [8], where data is obtained by merging information coming from static and dynamic analysis.

Even though malware detection systems which combine deep learning and high-level features provide good accuracy values, some authors emphasize the convenience to adopt features as simple as possible so as to design light and efficient malware-detection systems [9]. We follow such direction in order to design a lightweight system capable of performing malware detection with short response time. The potentiality of applying deep learning on simple raw data, as we propose here, is confirmed by some works recently presented in the literature, such as [10], where an android malware detection system which applies a deep convolutional neural network to the raw sequences of opcode extracted from disassembled programs is presented.

B. Hybrid malware detection systems

The idea of combining static analysis and dynamic analysis to design a hybrid system has been proposed by some works in the literature. The common approach is to merge static and dynamic features in a single feature vector, which is then analyzed by a unique classification algorithm.

The authors of [11] present a comparative analysis of different classification algorithms applied to static and dynamic features, often obtained through a sandbox tool, such as in [12] and [13]. Even the authors of [14] propose the adoption of a single feature vector with static and dynamic information. Static features are obtained from the Printable Strings Information (PSI) contained in the executable file. The dynamic analysis is performed through the cuckoo sandbox [15] and produces the sequence of system calls represented as n-grams, using 3-API-call-grams and 4-API-call-grams. A similar system is proposed in [16], which uses the Op-codes of the disassembled executable file as static features, and the list of API calls with their parameters, and raised exceptions as dynamic features. The same information are exploited by the system proposed in [17], which uses Markov chain graphs of dynamic instructions and system calls.

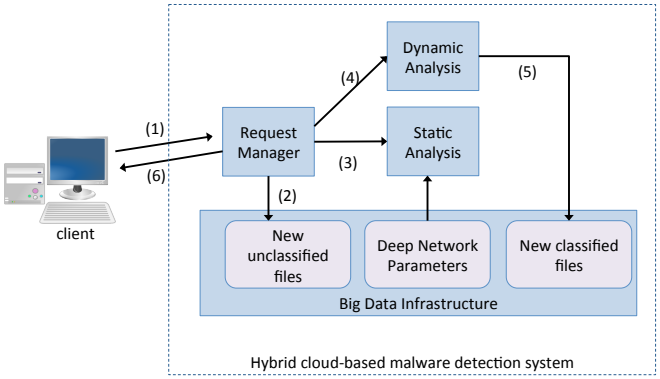


Fig. 1: System Architectures

A different approach is adopted by the authors of [18], which propose a hybrid system for detecting Android ransomware, composed by a static detection method which is applied to determine whether to allow to run the software, and a dynamic detection method to monitor software during its execution. The authors of [19] propose a similar approach, but where the dynamic analysis is performed only for suspicious files.

Similarly, we propose a hybrid system in which the dynamic analysis is performed only when the static analysis produces a high degree of uncertainty. Such design choice is justified by the need of guaranteeing a small computational burden in a cloud-based system which has to exhibit reduced response time even when a request peak occurs.

Moreover, to the best of our knowledge, our work is the first which adopts dynamic analysis to produce new sets of labeled data in order to perform a continuous learning process which refines the static analysis. The continuous learning of malware detection systems has been proposed by some other works in the literature in order to guarantee a model always up-to-date with respect to new threats. However, in such works, new labeled dataset are obtained by the intervention of human experts which annotate potential malware over time [20]–[22].

III. System Architecture

In this paper, we propose a cloud-based malware detection system which combines static and dynamic analyses in order to provide fast classification of executable files on very large scale. Our system is based on a big data management infrastructure capable of ingesting data in real time and supporting a continuous learning process which exploits such data to refine the underlying models.

Figure 1 shows the architecture of our system and the sequence of steps performed during the interaction with the client agents. Clients send the executable files to be analyzed to the Request Manager (RM) (1), a cloud agent responsible to drive the whole malware detection process. Request Manager, by applying simple hashing filters, discriminates between new files and copies of already received. Each previously unseen executable file is stored as new sample of unclassified data,

and will be used in successive learning phases (2). Malware detection is performed by the Static Analysis subsystem (SA) (3), which is based on a deep network capable of obtaining a good classification accuracy by processing a small part of the file through a lightweight process. Such subsystem provides also the probability that the analyzed file is a malware, thus giving a measure of its uncertainty degree. If the uncertainty exceeds a given threshold, the request manager activates the Dynamic Analysis (DA) (4), computationally more expensive both in terms of CPU and memory waste, and consequently exhibiting higher response times in comparison to the static analysis. Such a hybrid approach allows to statistically provide short response times in large scale systems, since it introduces temporal delays only when necessary to guarantee a high accuracy degree. The uncertainty threshold which triggers the dynamic analysis can be also dynamically tuned at run-time, in order to perform more extensive analyses when the incoming load decreases. The classification results obtained by dynamic analysis are then associated by the Big Data Infrastructure (BDI) to the analyzed files (5), thus building a new set of labeled files for the continuous learning process. Finally, classification results are delivered to the clients (6).

In the following subsections, we describe the static and the dynamic analyses. Moreover, we describe the learning process which is responsible of training the system before its activation, performing the continuous learning by exploiting new unclassified files provided by clients, with the associated results of the dynamic analysis.

A. Static Analysis

The Static Analysis subsystem we propose is based on a two-phase training deep neural network, where a first unsupervised pre-training with stacked denoising autoencoders [23] is followed by a supervised fine-tuning through back-propagation.

Features extraction is performed by accessing *DOS Header*, *File Header*, *Optional Header* and *Section Table* of the PE packaging. Each header contains a number of different fields, for each of which three values are read: data contained in the field and two offsets. In order to obtain a compact representation of fields data, simple values are handled as unsigned integers, whilst other data, e.g., timestamps, arrays, strings, are processed by means of a hashing function. Offset values are needed to preserve any spatial information related to the fields. In particular, the *local offset* specifies the position of a certain field within the header, whilst the *global offset* represents the displacement from the beginning of the file.

Since a file can have a variabile number of sections and the classifier is designed to process fixed-size data, we limit the number of *Section Tables* to be processed. To this aim, some experiments were performed on real data showing that a reasonable threshold on the number of sections is 13. If a file contains less sections than the threshold then the elements of the feature vector related to the missing sections are set to zero; otherwise, if the number of sections is higher than the threshold, the extra sections will be ignored. Thus, 19 fields of the DOS header, 7 of the file header, 30 of the optional header, 12 of the section section header (for each of the 13 sections) were analysed, obtaining feature vectors of 636 elements in the range $[0, 1]$. Since data are not extracted from the entire file but only from its headers, the feature extraction process is very fast and independent of the file size.

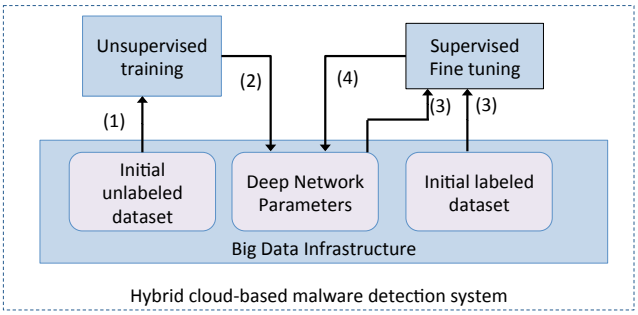


Fig. 2: Two-phase training of the deep network, core of the static-analysis subsystem.

The deep network proposed here consists of five layers. The input layer has 636 elements, the three hidden layers contain 256, 64, and 16 nodes respectively, whilst the output layer consists of a single node. For all nodes, sigmoid activation functions are used. In this neural network, the total amount of trainable hyper-parameters is of $180,577$ elements.

The training of the deep network is performed in two steps (see Figure 2). At the first step, rather than randomly initializing the model parameters, an unsupervised pre-training is performed on the unlabeled dataset in order to obtain a first estimation of weights and biases of the hidden layers. This first learning phase is implemented through stacked denoising autoencoders. According to such model, each hidden layer is pre-trained individually by means of a support network consisting of an input layer, a corrupted layer, an encoder, and a decoder layer (see Figure 3), where the encoder layer corresponds to the hidden layer to be pre-trained. On the contrary, the input, the corrupted, and the decoder layers have the same number of nodes, which is equal to the number of inputs of the hidden layer to be pre-trained.

This support network is trained through a back-propagation algorithm and processes each input vector by i) corrupting

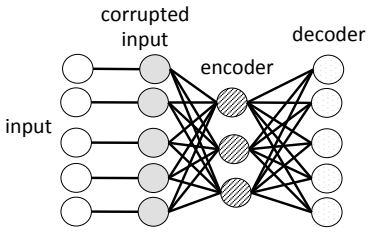
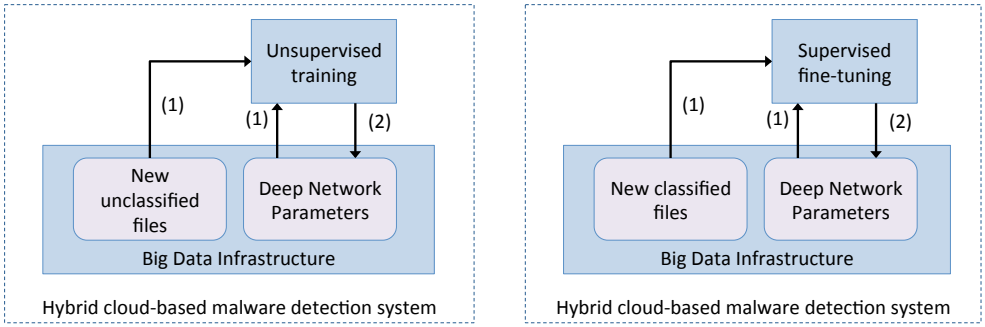


Fig. 3: Structure of the denoising autoencoder.



(a) (b)

Fig. 4: Continuous learning of the deep network through (a) unsupervised learning and (b) supervised fine-tuning.

it with some kind of noise, ii) encoding the resulting noisy signal, and then iii) reconstructing the encoded signal. The aim of the corruption layer, which adds isotropic Gaussian noise to the input signals, is to force the encoding layer to learn the most useful information from input vectors, by automatically neglecting noise from the corrupted input. The iteration of the pre-training for all the hidden layers produces a deep neural network in which each layer is able to extract and represent features at a higher level of abstraction than its predecessors. The pre-training of the first hidden layer is performed using the original dataset, while the other hidden layers are pre-trained through an encoded version of the dataset, obtained by exploiting the current trained denoising autoencoders to build a temporary network which extracts the output of the encoding layer that precedes the layer to be trained.

The second training step is the fine-tuning of the network, which is implemented through a supervised back-propagation algorithm with the Adam stochastic optimization, applied by considering the binary cross entropy as objective function. During this stage, weights and bias of all hidden layers are initialized with the values produced by the pre-training.

B. Dynamic Analysis

The modularity of our system allows an easy integration of different dynamic analysis algorithms without affecting the system architecture and its functioning logic. In its current version, the dynamic analysis subsystem is based on the cuckoo sandbox environment [15], installed under Windows 10 with the VirtualBox virtual machine, in order to create a secure evaluation environment. The cuckoo sandbox allows the real execution of the potentially infected codes producing as result a set of log files which summarize the behavior of the executable file during its execution. The tool can be tuned in order to produce different types of information, such as the list of API calls, each with its parameters, register alterations, heap memory addresses, process addresses, network usage, and so on. As proposed in [14] we use the sequence of API calls represented through the 3-grams and 4-grams method, as dynamic features. Other works, such as [11], suggest to enrich this feature vector with other information, such as the frequency of API calls, but the authors of [14] proved that even the simple adoption of API-n-grams allows to obtain an accuracy of 97.16%. Such features are analyzed through the random forest algorithm, which obtains good results for malware classification in several environments [14].

C. Continuous Learning

One of the most relevant features of our system is its capability of updating the internal model through a continuous learning process triggered by external input data.

In this process, the first input consists of big data flow generated by the client agents which submit classification requests. Such dataset continuously evolves following the natural advancement of application software and malware. The second input considered is constituted by the labeled data samples produced by the Dynamic Analysis subsystem when activated to classify the most uncertain files.

An off-line training phase precedes the system activation. During this step, as previously described, the deep network representing the heart of the Static Analysis subsystem is trained in two phases. In the first phase, an unlabeled dataset is used to train the network, in order to automatically identify the relevant information contained in the feature vector, regardless of whether the file is a malware or not. The second phase performs a fine-tuning of the network parameters using a labeled dataset, in order to learn how the relevant information corresponds to the two file classes (malware or not).

During all the normal life cycle, our cloud-based system is in charge of classifying huge amounts of data. In our approach, we propose to exploit such data in order to update the deep network parameters using an auxiliary network of stacked denoising autoencoders, as in the initial learning phase (see Figure 4a). This way, starting from a set of already refined parameters, the system accomplishes those internal adjustments which allow to capture relevant features in new malware samples.

It is worth noticing that such unlabeled dataset has been already analyzed by the deep network whenever the clients submit the classification requests. After the static classification, the Big Data Infrastructure characterizes each file by its

Classifier	Loss	Accuracy (%)	Precision (%)	TPR (%)	FPR (%)	AUC (%)
Deep model (600)	0.075	97.39	97.48	**97.33**	2.55	97.39
Deep model (800)	**0.074**	**97.48**	**98.09**	96.88	**1.91**	97.48
Deep model (1000)	**0.074**	**97.49**	97.92	97.08	2.09	**97.49**
Deep model (*fine tuning* only)	0.087	96.73	96.69	96.82	3.37	96.73
Classic-623-256	0.095	96.48	95.98	97.10	4.15	96.47

TABLE I: Average values Loss, Accuracy, Precision, TPR (True Positive Rate), FPR (False Positive Rate), and AUC (Area Under Curve) using different classifiers. The number of pre-training epochs is reported in parentheses.

probability to be a malware. Only the most uncertain files are further analyzed by the Dynamic Analysis subsystem. Such dynamic analysis is completely independent from the static one, and is statistically more accurate. Its results are thus utilized as new descriptions for those further analyzed files, in order to create a new labeled dataset. When the amount of new annotated data exceeds a given threshold, or at regular time intervals, a new fine-tuning process through back-propagation is started, in order to further refine the network parameters (see Figure 4b).

Such an approach allows to overcome one of the most relevant issues in deep network design, that is the availability of a big amount of data to be exploited for training. Furthermore, the exploitation of dataset collected in different time intervals, allows the network to dynamically follow the continuous evolution of the software.

IV. Experimental Evaluation

The experimental evaluation presented here aims to evaluate the effectiveness of the proposed approach and the impact of the design choices we made.

The dataset used for the experimental evaluation was obtained by merging 12.000 samples of malware obtained from VirusShare [1] and 11.874 samples of certified software obtained from a clean Windows 10 installation.

Firstly, we wanted to investigate whether there is a connection between the number of epochs used to train the deep network and the performance of the whole system. For this purpose we compared the performance obtained by varying the maximum training epoch threshold (i.e., 600, 800, and 1000) for the pre-training phase. The second evaluation aimed to assess the effectiveness of the pre-training phase, by comparing our system with a deep network characterized by the same topology, but trained only through the second training phase. Finally, we intended to verify the impact of the hierarchical stratification of our deep classifier on the overall performances of the malware detection system. To this aim, we compared our system with a classic neural network obtained by removing the last two hidden layers from our classifier, so obtaining three layer of 636, 256 and 1 nodes respectively.

The evaluation is performed through a K-fold cross validation with a stratified sampling, in order to preserve the percentage of samples per class, with $K = 5$.

The classification performance are evaluated by analyzing the trend of the ROC curve (Receiver Operating Characteristic) with respect to the training epochs. Furthermore we computed the final loss, the final accuracy, as well as several other metrics, i.e., TPR (True Positive Rate), FPR (False Positive Rate), Precision, and AUC (Area Under Curve) of ROC curve. This last metrics allows to evaluate the classification performance independently of the threshold adopted for the last layer of the deep network.

In Table I, the average values of the considered performance metrics are shown. We can observe that higher accuracy values are obtained by using the pre-trained classifiers, and this confirms the effectiveness of the two-phase training. However, increasing the number of pre-training epochs (e.g., from 600 to 800 or 1000) does not significantly improve the accuracy, nor reduce the loss value.

The performances of the model that uses fine tuning only are worse than the pre-trained models on all metrics. Furthermore, the lowest values are achieved by using the classic model, suggesting that the hierarchical stratification is effective, but only when combined with the pre-training phase.

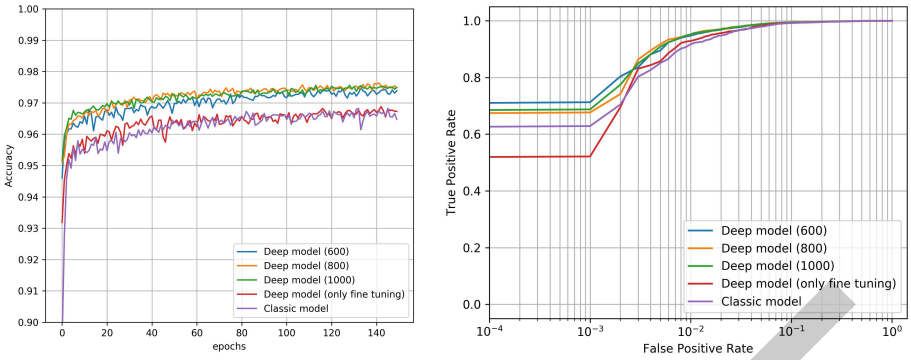
A more in-depth analysis of the average accuracy achieved using different models is shown in Figure 5a, which uses results from the second training phase. Results confirm that the pre-trained deep models exhibit a better trend along all the time line. Figure 5b shows the average ROC curves from which AUC score was calculated. Even in this case, pre-trained models outperform the others, with no significative differences when using 800 or 1000 pre-training epochs.

V. Conclusions and Future Work

In this work, we propose a cloud-based malware detection system, capable of facing big data amount produced on the network. Our system performs a static analysis based on deep networks to provide a fast classification of potential malware files, and a more burdensome dynamic analysis only when the detection uncertainty exceeds a given threshold. The continuous flow of data produced by client agents and the classification results of the dynamic analysis are exploited to refine the deep network in a continuous learning loop. Such mechanism makes the system always up-to-date with respect to new malware versions.

The preliminary experimental evaluation presented here suggests the effectiveness of the proposed approach based on an unsupervised learning which understand how to extract relevant information from raw data, and a supervised learning which understands how correlate such information to the malicious or benign nature of the analyzed file.

[1] https://virusshare.com



(a) (b)

Fig. 5: Average accuracy (a) and ROC curves (b) of different models. Pre-train epochs are reported in parentheses.

As future work, we plan to perform an exhaustive experimental evaluation to assess the effectiveness of the continuous learning loop with malware samples collected in different time periods. Moreover, we will evaluate the possibility of adopting a deep network as core of the dynamic analysis, built as an extension of the deep network used for the static analysis.

Finally, we will address some issues related to the cloud infrastructure, such as the potentially huge amount of traffic generated from the clients. We plan to overcome such problem by designing a lightweight version of the static analysis to be performed on mobile devices, thus triggering the communication toward the remote server only to perform the dynamic analysis. Such improvement will allow clients to analyze executable files even during offline periods.

REFERENCES

[1] J. H. Abawajy, A. Kelarev, and M. Chowdhury, "Large iterative multitier ensemble classifiers for security of big data," *IEEE Trans. on Emerging Topics in Computing*, vol. 2, no. 3, pp. 352–363, 2014.

[2] K. Mathur and S. Hiranwal, "A survey on techniques in detection and analyzing malware executables," *Int. J. of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, 2013.

[3] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *Proc. of the 2015 Int. Joint Conf. on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[4] B. Kolosnjaji, A. Zarras, G. D. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proc. of the Australasian Conf. on Artificial Intelligence*, ser. Lecture Notes in Computer Science, vol. 9992. Springer, 2016, pp. 137–149.

[5] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Proc. of the 2013 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 3422–3426.

[6] Y. Wang, W.-D. Cai, and P. Wei, "A deep learning approach for detecting malicious javascript code," *Security and Communication Networks*, vol. 9, no. 11, pp. 1520–1534, 2016.

[7] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. of the 2015 10th Int. Conf. on Malicious and Unwanted Software*, 2015, pp. 11–20.

[8] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "Hadm: Hybrid analysis for detection of malware," in *Proc. of the SAI Intelligent Systems Conf.*, 2016, pp. 1037–1047.

[9] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. of the Sixth ACM Conf. on Data and Application Security and Privacy*, 2016, pp. 183–194.

[10] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, and G. Joon Ahn, "Deep android malware detection," in *Proc. of the Seventh ACM on Conf. on Data and Application Security and Privacy*, 2017, pp. 301–308.

[11] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[12] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proc. of the 4th ACM Work. on Security and Artificial Intelligence*, 2011, pp. 21–30.

[13] K. Schütt, M. Kloft, A. Bikadorov, and K. Rieck, "Early detection of malicious behavior in javascript code," in *Proc. of the 5th ACM Work. on Security and artificial intelligence*, 2012, pp. 15–24.

[14] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[15] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "The cuckoo sandbox," last accessed 03 Jan. 2018. [Online]. Available: https://cuckoosandbox.org

[16] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *Int. Joint Conf. CISIS12-ICEUTE 12-SOCO 12 Special Sessions*, 2013, pp. 271–280.

[17] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap," in *Proc. of the 5th ACM Work. on Security and artificial intelligence*, 2012, pp. 3–14.

[18] A. Ferrante, M. Malek, F. Martinelli, F. Mercaldo, and J. Milosevic, "Extinguishing ransomware - a hybrid approach to android ransomware detection," in *The 10th Int. Symp. on Foundations Practice of Security*, 2017, pp. 1–16.

[19] A. Gharib and A. Ghorbani, "Dna-droid: A real-time android ransomware detection framework," in *Int. Conf. on Network and System Security*, 2017, pp. 184–198.

[20] B. Miller, A. Kantchelian, S. Afroz, R. Bachwani, E. Dauber, L. Huang, M. C. Tschantz, A. D. Joseph, and J. D. Tygar, "Adversarial active learning," in *Proc. of the 2014 Work. on Artificial Intelligent and Security*, 2014, pp. 3–14.

[21] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, "AI^2: training a big data machine to defend," in *Proc. of the IEEE Int. Conf. on High Performance and Smart Computing, and IEEE Int. Conf. on Intelligent Data and Security, 2016 IEEE 2nd Int. Conf. on Big Data Security on Cloud*, 2016, pp. 49–54.

[22] A. Beaugnon, P. Chifflier, and F. Bach, "Ilab: An interactive labelling strategy for intrusion detection," in *Int. Symp. on Research in Attacks, Intrusions, and Defenses*, 2017, pp. 120–140.

[23] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.