Deep Learning for Classification of Malware System Call Sequences

Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert

Technical University of Munich
`[kolosnjaji,zarras,webstergd,eckert]@sec.in.tum.de`

Abstract. The increase in number and variety of malware samples amplifies the need for improvement in automatic detection and classification of the malware variants. Machine learning is a natural choice to cope with this increase, because it addresses the need of discovering underlying patterns in large-scale datasets. Nowadays, neural network methodology has been grown to the state that can surpass limitations of previous machine learning methods, such as Hidden Markov Models and Support Vector Machines. As a consequence, neural networks can now offer superior classification accuracy in many domains, such as computer vision or natural language processing. This improvement comes from the possibility of constructing neural networks with a higher number of potentially diverse layers and is known as *Deep Learning*.
In this paper, we attempt to transfer these performance improvements to model the malware system call sequences for the purpose of malware classification. We construct a neural network based on convolutional and recurrent network layers in order to obtain the best features for classification. This way we get a hierarchical feature extraction architecture that combines convolution of n-grams with full sequential modeling. Our evaluation results demonstrate that our approach outperforms previously used methods in malware classification, being able to achieve an average of 85.6% on precision and 89.4% on recall using this combined neural network architecture.

1 Introduction

An increasing problem in large-scale malware detection and analysis is the high number of new malware samples. This number has exponentially increased throughout the years, which creates difficulty for malware analysts, as they need to extract information out of this large-scale data. Recent reports from McAfee reveal that tens of thousands of new distinct samples are being submitted on a daily basis [9]. Furthermore, statistics page of VirusTotal shows that, in just a single day, over a million newly retrieved samples had to be analyzed [29]. This surge of samples makes reverse engineering a challenging task. Although there exist efforts to automate the reverse engineering and malware analysis process, manual signature-based or heuristics-based detection and analysis procedures are still very prominent. Apart from the problem of sheer number of samples, these samples are of increased variety, which is usually caused by advances in malware development that utilize polymorphic and metamorphic algorithms to generate different versions of the same malware. This makes it extremely difficult for signature-based systems to correctly classify and analyze these samples.

To aid malware analysts in retrieving useful information from such a large amount of samples, we need to solve the problem of automatic classification under the existing statistical variance on a large scale. Existing signature-based malware detection systems cannot cope with this variance as they do not take statistical noise into account and thus this kind of systems can be easily evaded. Therefore, we need a robust alternative that can abstract away the noise and capture the essential information from static or behavioral malware properties.

Static analysis tools, such as PEInfo [1], offer extraction of different properties or metadata (e.g., entropy, histograms, section length) from malware code. This data can be very useful for characterizing malware samples. However, miscreants can easily obfuscate the malware code to the point where it is impossible to retrieve any useful information from static analysis. On the other hand, behavioral analysis tools are less sensitive to obfuscation, as they only record traces of activity retrieved from the execution of malware samples.

The most important traceable events for determining malware behavior are system calls. In order to execute malicious actions, malware needs to use the services from the operating system. For any meaningful action, such as opening a file, running a thread, writing to the registry, or opening a network connection, interaction with the operating system (OS) is necessary. This interaction is done through the system call API of the target OS. Therefore, in order to characterize the malware behavior, it is important to track the sequence of system call events during the execution of malware. Different malware families have different execution goals, which should be revealed by inspecting these traces.

Towards this direction, machine learning-based systems have been developed as a solution to the problems of large number and variety of malware samples. For instance, researchers utilized Hidden Markov Models in an attempt to model system call sequences [4]. Others used Support Vector Machines (SVM) with String Kernels to detect malware based on the executed system calls [21]. Apart from behavioral data, static code properties have also been used as data sources for statistical analysis [25]. Many other papers of similar content have tried to deal with this issue [5, 6, 22]. The problem with these approaches is that many times machine learning methods use simplifying assumptions. For example, some of these works utilize Markov assumption of memoryless processes in Hidden Markov Models or different kernel definitions that define similarity measures between samples. One exception is the work of Pascanu et al. [19], where they use recurrent networks for modeling system call sequences, in order to construct a "language model" for malware. They test Long Short-Term Memory and Gated Recurrent Units and report good classification performance. However, they do not test deep learning approaches. This may simplify the modeling, but on the other hand it can result in reduced classification accuracy.

In this paper, we focus on investigating the utilization of neural networks to improve the classification of newly retrieved malware samples into a predefined set of malware families. As a matter of fact, recent years have brought a significant development in the area of neural networks, mostly under a paradigm of *deep learning*. This paradigm encompasses a movement towards creating neural networks with a high number of layers to model complex functions of input data. Using modern hardware technology, such as General Purpose GPUs and novel algorithms developed in recent years, deep networks can be trained efficiently using high dimensional datasets on a large scale.

We construct and analyze two types of neural network layers for modeling system call sequences: *convolutional* and *recurrent* layers. These two types of layers use different types of approach in modeling sequential data. On the one hand, convolutional networks use sequences in a form of a set of n-grams, where we do not explicitly model the sequential position of system calls and only count presence and relation of n-grams in a behavioral trace. While this approach simplifies sequence modeling, it also potentially causes loss of information fidelity. On the other hand, recurrent networks train a stateful model by using full sequential information: the model contains dependency of certain system call appearance from the sequence of previous system calls. Since this model is more complex, it is more difficult to train. However, if trained properly, it could potentially offer better accuracy on sequential data, as it is able to capture more information about the training set. By combining those two layers in a hierarchical fashion, we can increase our malware detection capabilities. This is enabled by more robust automatic feature extraction, where we convolve n-grams of system calls and, furthermore, create sequential model out of convolution results. We stack layers in the neural network in accordance to the principles of *deep learning* [7]. In deep learning, one can construct deep neural networks in order to extract a hierarchy of features for classification. We use this technique to achieve improvement in capturing the relation between n-grams in system call sequences. In essence, our approach enables us to get average accuracy, precision, and recall of over 90% for most malware families, which brings significant improvement over previously used methods and thus can help analysts to classify malware more accurately.

In summary, we make the following main contributions:

- We construct deep neural networks and apply them to analyze system call sequences.
- We combine convolutional and recurrent approaches to deep learning for optimizing malware classification.
- We investigate neural unit activation patterns and explain the performance improvement of our models by illustrating the inner workings of our neural network.

2 Methodology

In this section, we describe the methodology we use to perform the task of malware classification. We first provide information regarding our set up environment and then proceed with our deployed techniques.

2.1 System Description

Our malware classification process is displayed in Figure 1. It begins with a malware zoo, where the executable files are acquired and input data is retrieved by executing malware in a protected environment. The results of these executions are preprocessed in order to get numerical feature vectors. These vectors are then forwarded as inputs for neural networks, which in turn classify the malware into one of the predefined malware families.

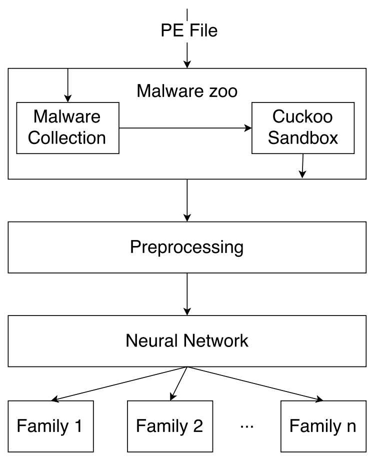

Figure 1. Overview of our malware classification system.

We use the *Tensorflow* [3] and the *Theano* [8] frameworks to construct and train the neural networks. These frameworks enable us to design neural network architectures and precompile the training algorithms for execution on graphical processors. Since GPUs are designed for fast execution of linear algebra operations, such as matrix multiplications, we utilize them to speed up our neural network training. This is a very popular approach in neural network applications, since training deep networks can be a very resource-intensive task. In our set up, we use the `NVIDIA TITAN X GPU` and the `NVIDIA CUDA 7` software platform to accelerate the training algorithms.

2.2 Dataset

We collect malware samples and trace malware behavior using a *malware zoo* [32]. Our malware collection consists of samples gathered from three primary sources: Virus Share [23], Maltrieve [18] and private collections. We select these sources to provide a large and diverse volume of samples for evaluation.

Since malware authors can use code obfuscation and packers in order to subvert static analysis, we use dynamic malware analysis to gather data about malware behavior. Towards this direction, there exist multiple tools that enable tracing the execution of malware and gathering of logs of execution sequences [13,17]. We choose the Cuckoo sandbox which is open source, widely-used, and provides a controlled environment for executing malware. During the execution of malware samples we record calls to the kernel API that we later use to characterize these malware samples. For each malware sample we obtain a sequence of API calls (system calls) and employ this sequence for behavioral modeling.

We use kernel API call sequences (system calls) as features, but as we are doing supervised learning, we also need labels for training. These labels are obtained using services of VirusTotal [2]. In particular, for each malware sample we use, we extract antivirus labels from the VirusTotal web service. This service is used by uploading MD5 hashes of malware executables to VirusTotal and retrieving results from a large number of antivirus engines through the VirusTotal API. These engines compare the hash of the malware file to the data already contained in their own database. We leverage the VirusTotal services in order to access malware analysis results and particular signatures for the samples in our malware zoo. We are interested in the antivirus signatures, out of which we want to retrieve labels for supervised learning. Each antivirus that provides signatures through VirusTotal has its own methodology of labeling malware, which makes it difficult for machine learning classification tests, since we need to extract one numerical label per unique sample. Our approach was to execute clustering on the signatures from different antivirus programs and obtain ground truth classes from the resulting clusters. An ideal solution for labeling is to use reverse engineering and expert knowledge. However, on a larger scale that is not a realistic scenario. Therefore we postprocess the VirusTotal results in order to pull maximum information without manual inspection of malware samples.

2.3 Signature Clustering

We attempt to extract maximum amount of information from the VirusTotal signatures by performing a simplified version of signature clustering method introduced in VAMO [20]. To each malware sample we attach a boolean vector that contains information about presence or absence of different antivirus signatures. Using a variant of *cosine distance* and DBSCAN [12] algorithm we cluster signature vectors in order to detect regions of high similarity. We select the ten most populated clusters and use them to create classes for the evaluation of our methodology. These ten clusters contain 4753 malware samples in total and cover most of our sample set. The rest of the samples we consider as outliers. Usually a malware analyst is interested in extracting samples belonging to certain families and tries to differentiate them from the other executables in the dataset.

2.4 Feature Preprocessing

Before using the API call sequences as inputs to neural networks, we need to remove redundant data and convert the data to sequences of numerical feature vectors. First, we preprocess sequences by removing subsequences where one API call is repeated more than two times in a row. For example, this happens if a process tries to create a file repeatedly in a loop. We cut these subsequences by using maximum two consequent identical kernel API call instances in the

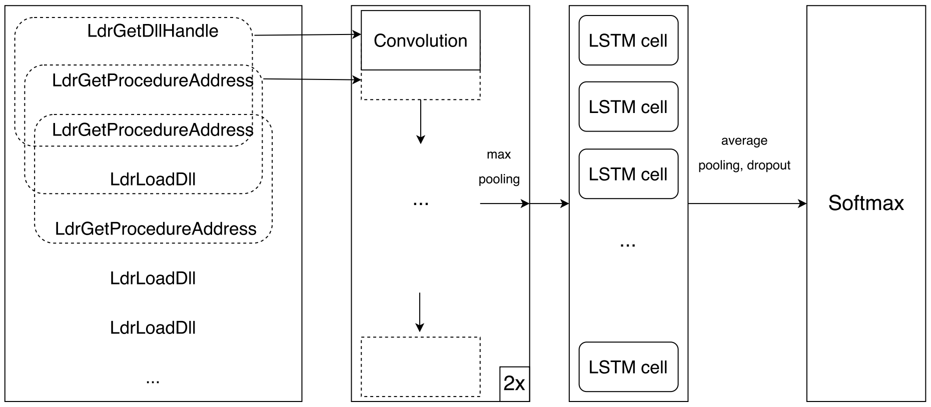

Figure 2. Deep Neural Network architecture

resulting sequence. Furthermore, we use *one-hot encoding* to find a unique binary vector for every API call present in the dataset. This means that we create zero feature vectors of length equal to the number of distinct API calls and toggle one bit in a position unique for a particular kernel API call. This way, we have sequences of binary vectors instead of sequences of API call names provided by Cuckoo sandbox. Since our dictionary consists of only 60 distinct system calls, we do not face any challenge with the size of feature vectors.

2.5 Deep Neural Network

In order to maximize the utilization of the possibilities given by neural network methodology, we combine convolutional and recurrent layers in one neural network. Figure 2 depicts our neural network architecture. The convolutional part consists of convolution and a pooling layers. On the one hand, the convolutional layer serves for feature extraction out of raw one-hot vectors. Convolution captures the correlation between neighboring input vectors and produces new features. We use two convolution filters of size 3×60, which corresponds to 3-grams of instructions. As the results of convolution we take feature vectors of size 10 and 20 for the first and second convolution layer, for every input feature. After each convolutional layer we use max-pooling to reduce the dimensionality of data by a factor of two. Outputs of the convolutional part of our neural network are connected to the recurrent part. We forward each output of the convolutional filters as one vector. The resulting sequence is modeled using the LSTM cells. We use LSTM cells, as they are flexible in terms of training, even though the maximal sequence length was limited to 100 vectors. Using the recurrent layer we are able to explicitly model the sequential dependencies in the kernel API traces. Mean-pooling is used to extract features of highest importance from the LSTM output and reduce the complexity of further data processing. Furthermore, we use Dropout [26] in order to prevent overfitting and a softmax layer to output the label probabilities.

Family	**Results of various neural network architectures (%)**								
	Feedforward network			**Convolutional Network**			**ConvNet+LSTM**		
	ACC	PR	RC	ACC	PR	RC	ACC	PR	RC
Multiplug	99.6	100.0	99.3	98.9	98.9	98.9	98.9	99.8	99.0
Kazy	100.0	71.7	100.0	100.0	100.0	100.0	100.0	99.9	100.0
Morstar	0.0	0.0	0.0	100.0	100.0	100.0	100.0	99.9	100.0
Zusy	9.1	57.2	68.5	100.0	56.8	100.0	100.0	57.5	100.0
SoftPulse	100.0	100.0	98.2	100.0	99.6	100.0	100.0	99.1	100.0
Somoto	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Mikey	89.1	37.1	28.8	0.0	0.0	0.0	0.0	0.0	0.0
Amonetize	100.0	98.4	100.0	99.1	100.0	99.1	99.1	100.0	99.6
Eldorado	100.0	100.0	96.4	99.2	100.0	99.2	99.4	100.0	99.5
Kryptik	100.0	100.0	100.0	95.0	100.0	95.1	96.6	100.0	96.2
Average	79.8	76.4	79.1	89.2	85.6	89.2	89.4	85.6	89.4

Table 1. Evaluation of deep hybrid network.

3 Evaluation

In this section, we describe the outcomes of our experiments executed using the data from our malware zoo and the neural network architectures that we constructed. For our evaluation measurements we noted the performance for different families in terms of accuracy (ACC), precision (PR) and recall (RC).

In our evaluation we used a type of 3-fold crossvalidation. In each test, we chose two thirds of samples as training set, while one third of samples is assigned to the test set. As a matter of fact, to obtain a reliable performance estimate, we averaged the results of ten crossvalidation tests, executed each time with a new random dataset permutation.

In our first experiment, we investigated the performance of our neural network architecture with respect to the simpler, previously used architectures, such as feedforward neural network, or convolutional network. Table 1 displays the overall results in percentage. These results show that our configuration brings improvements in the classification performance. It also shows that for one family, *Mikey*, we do not get good accuracy. This family gets very often confused for the family *Zusy*, because of very similar behavioral traces.

Next, we wanted to measure the performance improvement of our methodology. Therefore, we compared the performance of deep hybrid architecture with Hidden Markov Models and Support Vector Machines (SVM), since these methods have been heavily used in previous works. Table 2 illustrates the results. This table shows high improvement in case of using neural networks as classifiers. The improvement is statistically significant, with a 5x2cv t-statistics value [11] in the 95% confidence level in comparison with the SVM.

In our last experiment we attempted to illustrate the inner working of our neural network. Thus, we visualized intermediate results of the convolutional layer in the neural network trained on our dataset. Table 3 displays the correlation of 5-gram similarity and similarity of results from convolutional filters. We got the best classification results for using convolutional filters of width 3, but

Family	**Deep Neural Network**			**Hidden Markov Model**			**Support Vector Machine**		
	ACC	PR	RC	ACC	PR	RC	ACC	PR	RC
Multiplug	98.9	99.8	99.0	91.5	74.5	91.5	99.3	99.9	99.3
Kazy	100.0	99.9	100.0	73.1	95.1	73.1	96.6	93.1	96.6
Morstar	100.0	99.9	100.0	80.0	63.7	80.0	82.3	91.0	82.3
Zusy	100.0	57.5	100.0	65.4	45.1	65.4	100.0	58.4	100.0
SoftPulse	100.0	99.1	100.0	51.1	100.0	51.1	99.9	99.6	99.9
Somoto	100.0	100.0	100.0	50.0	37.6	50.0	99.8	100.0	99.8
Mikey	0.0	0.0	0.0	5.7	20.0	5.7	0.0	0.0	0.0
Amonetize	99.1	100.0	99.6	29.4	100.0	29.4	99.3	100.0	99.3
Eldorado	99.4	100.0	99.5	20.0	80.4	20.0	100.0	100.0	100.0
Kryptik	96.6	100.0	96.2	10.0	100.0	10.0	97.1	100.0	97.1
Average	89.4	85.6	89.4	47.5	71.6	47.6	87.4	84.2	87.4

Table 2. Comparison with previously used methods.

API Call Array	**Neuron Activation Value**
LdrGetDllHandle, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetDllHandle, LdrLoadDll	64.40814948
LdrGetDllHandle, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrLoadDll, LdrGetProcedureAddress	72.43323427
LdrGetDllHandle, DeviceIoControl, DeviceIoControl, LdrGetDllHandle, LdrLoadDll	-59.49672516
LdrGetDllHandle, LdrGetProcedureAddress, LdrGetProcedureAddress, RegOpenKeyExW, RegOpenKeyExW	-60.89228849
ExitThread, LdrLoadDll, LdrGetProcedureAddress, LdrGetProcedureAddress, LdrGetDllHandle	-60.64636472
NtOpenFile, NtCreateSection, ZwMapViewOfSection, NtOpenFile, LdrLoadDll	-12.56477188

Table 3. Comparison of neuron activation

we use 5-grams here for visualization, as it is more illustrative to show longer n-grams. The activation values are normally vectors, but we reduced them to scalars using the t-SNE transformation [28], for better visualization. It is noticeable that the neuron activation values follow the similarity between n-grams of kernel API calls. The function that governs how this similarity is followed is optimized during the training of our neural network.

As the input values are propagated through the layers of the neural network, activations are trained in order to separate different malware classes. In the last layer the activation values should be able to totally separate between different classes, based on features learned in previous layers. Figure 3 shows the results of clustering neuron activation values before the softmax layer, when using malware data of different families as inputs. Again, the neuron activation vectors are reduced to two-dimensional representation using the t-SNE transformation.

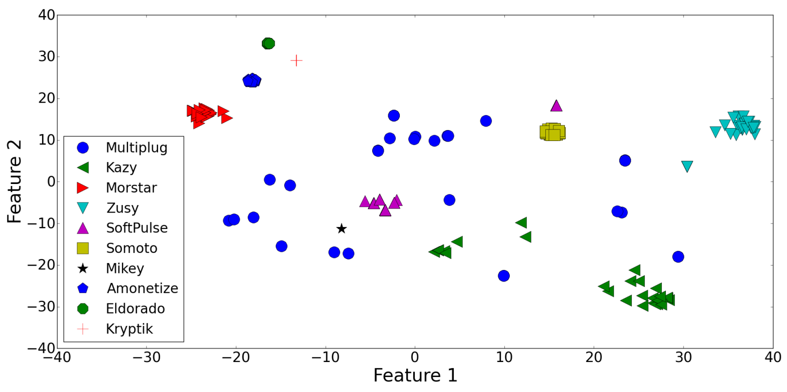

Figure 3. T-SNE transform of neuron activations in the last layer of our deep hybrid network.

4 Discussion

Our results show that deep learning indeed brings improvements in classification of malware system call traces. Although there exist classes where the classification accuracy is under 90%, on average we bring an improvement of performance with respect to Hidden Markov Models and SVM. We combine convolutional and recurrent layers to achieve this improvement. This combination helps us obtain slightly better results than with simpler architectures with only feedforward or only convolutional layers. Using only LSTM recurrent network also does not achieve accuracy as high as we get with our architecture, which can be explained with the relatively short length of the malware execution traces.

While dynamic analysis information is very important, in this work we only look at system call traces, which contain information of only the path taken by malware on execution. This path could also depend on input data, which we ignore in this work. By joining static and dynamic analysis, we could unify the two approaches in looking at malware features. Future work will be dedicated to this goal. One other limitation of our approach is that we do not consider evasion of malware detectors by inserting noise in the system call sequences. This issue has been raised before by Wagner and Soto [30]. We plan to investigate the mitigation of this issue in the future.

Although training time for neural network ranges from three to ten hours, on test time the classification is instantaneous. Therefore the neural network approach is very good in case where there is no common need to retrain the model. We have also experimented with extending the neural network further with an even higher number of layers. However, we did not get any more performance improvement. Further improvement could exist if the dataset was larger and if the malware sample set was more diverse. Our plan is to investigate this direction as well.

5 Related work

This section contains the description of the research efforts that precede our paper. These efforts are mostly divided into research dedicated to (*i*) application of machine learning methods in malware analysis and (*ii*) using specifically neural networks for malware detection and classification. In the following we explain the evolution and current state of those two groups of methods separately.

5.1 Machine Learning Methods for Malware Detection

Machine learning-based malware detection and classification are topics of multiple research efforts. These efforts use various behavioral features of malware as input data for statistical models. The features are obtained by analyzing code or tracing events such as system calls [31], registry accesses [14], and network traffic [27]. This kind of event sequences are analyzed using unsupervised (e.g., clustering), semi-supervised, or supervised learning (classification) methods. Although most of papers investigate malware clustering [5, 6], supervised learning is also a prominent topic.

There also papers in recent years that try to use advanced machine learning methods and extract more information from malware datasets. An example of this is given by recent application of statistical topic modeling approaches to the classification of system call sequences [34], which was further extended with a nonparametric methodology [16]. This approach was extended by taking system call arguments as additional information and including memory allocation patterns and other traceable operations [33]. Support vector machines with string kernels represent an another interesting sequence-aware approach [21].

5.2 Neural Networks for Malware Detection and Classification

There are also noticeable efforts to use neural networks for malware detection and analysis. For example, Dahl et al. [10] try to classify malware on a large scale using random projections and neural networks. However, they do not report advantages of increasing the number of neural network layers. Saxe et al. [24] used feedforward neural networks to classify static analysis results. However, they do not consider dynamic analysis results in their research. In case of obfuscated binaries, the static analysis may not give satisfactory inputs for classification. Huang et al. [15], on the other hand, use up to four hidden layers of feedforward neural network, but focus on evaluating multi-task learning ideas. Pascanu et al. [19] use recurrent networks for modeling system call sequences, in order to construct a "language model" for malware. They test Long Short-Term Memory and Gated Recurrent Units and report good classification performance. However, they do not test deep learning approaches.

These papers give motivating conclusions that enhance the reputation of neural networks being applicable for malware datasets. However, these papers do not report any advantages from diversifying deep architectures in case of behavioral modeling of malware activity and they do not do extensive research in that direction.

6 Conclusion

In this paper, we construct deep neural networks to improve modeling and classification of system call sequences. By combining convolutional and recurrent layers in one neural network architecture we obtain optimal classification results. Using a hybrid neural network containing two convolutional layers and one recurrent layer we get a novel approach to malware classification. Our neural network outperforms not only other simpler neural architectures, but also previously widely-used Hidden Markov Models and Support Vector Machines. Overall, our approach exhibits better performance results when compared to previous malware classification approaches.

References

1. PEInfo Service. `https://github.com/crits/crits_services/tree/master/peinfo_service`.
2. VirusTotal. `http://www.virustotal.com`, May 2015.
3. M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. *arXiv preprint arXiv:1603.04467*, 2015.
4. S. Attaluri, S. McGhee, and M. Stamp. Profile Hidden Markov Models and Metamorphic Virus Detection. *Journal in computer virology*, 5(2):151–169, 2009.
5. M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2007.
6. U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2009.
7. Y. Bengio. Learning Deep Architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.
8. J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A CPU and GPU math expression compiler. In *Python for Scientific Computing Conference (SciPy)*, 2010.
9. Z. Bu et al. McAfee Threats Report: Second Quarter 2012. 2012.
10. G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-Scale Malware Classification Using Random Projections and Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.
11. T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.
12. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases With Noise. In *Kdd*, 1996.
13. C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser. The Cuckoo Sandbox, 2012.
14. K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo. One Class Support Vector Machines for Detecting Anomalous Windows Registry Accesses. In *Workshop on Data Mining for Computer Security (DMSEC)*, 2003.
15. W. Huang and J. W. Stokes. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

16. B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert. Adaptive Semantics-Aware Malware Classification. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.
17. T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias. Scalability, Fidelity and Stealth in the Drakvuf Dynamic Malware Analysis System. In *Annual Computer Security Applications Conference (ACSAC)*, 2014.
18. K. Maxwell. Maltrieve. `https://github.com/krmaxwell/maltrieve`, Apr 2015.
19. R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware Classification With Recurrent Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.
20. R. Perdisci and M. C. U. VAMO: Towards a Fully Automated Malware Clustering Validity Analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2012.
21. J. Pfoh, C. Schneider, and C. Eckert. Leveraging String Kernels for Malware Detection. In *Network and System Security*. 2013.
22. K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and Classification of Malware Behavior. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2008.
23. J.-M. Roberts. Virus Share. `https://virusshare.com/`, Nov 2015.
24. J. Saxe and K. Berlin. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. *arXiv preprint arXiv:1508.03096*, 2015.
25. M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. In *IEEE Symposium on Security and Privacy*, 2001.
26. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks From Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
27. F. Tegeler, X. Fu, G. Vigna, and C. Kruegel. Botfinder: Finding Bots in Network Traffic Without Deep Packet Inspection. In *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2012.
28. L. Van der Maaten and G. Hinton. Visualizing Data Using T-Sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
29. VirusTotal. File Statistics. `https://www.virustotal.com/en/statistics/`, Nov 2015.
30. D. Wagner and P. Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Conference on Computer and Communications Security (CCS)*, 2002.
31. C. Warrender, S. Forrest, and B. Pearlmutter. Detecting Intrusions Using System Calls: Alternative Data Models. In *IEEE Symposium on Security and Privacy*, 1999.
32. G. Webster, Z. Hanif, A. Ludwig, T. Lengyel, A. Zarras, and C. Eckert. SKALD: A Scalable Architecture for Feature Extraction, Multi-User Analysis, and Real-Time Information Sharing. In *International Conference on Information Security*, 2016.
33. H. Xiao and C. Eckert. Efficient Online Sequence Prediction With Side Information. In *IEEE International Conference on Data Mining (ICDM)*, 2013.
34. H. Xiao and T. Stibor. A Supervised Topic Transition Model for Detecting Malicious System Call Sequences. In *Workshop on Knowledge Discovery, Modeling and Simulation*, 2011.