FeatAll: An All Feature Malware Detection Technique

Saurabh Gupta
Indraprastha Institute of Information Technology - Delhi
saurabhg@iiitd.ac.in

***Abstract*—Malware, i.e., malicious software, represents one of the main cyber security threats today. Over the last decade malware has been evolving in terms of the complexity of malicious software and the diversity of attack vectors. As a result modern malware is characterized by sophisticated obfuscation techniques, which hinder the classical analysis approaches. The tools that perform malware detection and classification are limited by the knowledge they have. Plenty of them are available with a plethora of techniques ranging from supervised shallow models to deep learning to unsupervised deep belief networks. However, due to availability of several feature selection methods, feature importance is always overlooked upon. In this work, we use a state-of-the-art Deep Belief Network implementation with different sets of data features. We perform an ablation study with all combinations of this data to see what features give a better representation. Specifically we collect four kinds of datasets: static features, abstract text features, behavioral traces and program behaviour. We observe through experiments that the combination of static features, abstract text features and behavioral traces yields the best results and therefore are more important that program behaviour data.**

***Index Terms*—malware, classification, detection**

I. Introduction

Malware, i.e., malicious software, represents one of the main cyber security threats today. Various attempts have been made in the past to detect and classify malware from benign programs. These methods range from the early-day signature-based detection to the more modern Machine Learning and Deep Learning based detection. Predictive technologies are already effective at detecting and blocking malware at first sight.

One of the major challenges that anti-malware software faces today are the vast amounts of data which needs to be evaluated for potential malicious intent. For example, Microsoft's real-time anti-malware detection products executes on over 600M computers worldwide. This generates tens of millions of daily data points to be analyzed as potential malware. One of the main reasons for these high volumes of different files is that in order to evade detection, malware authors introduce polymorphism to the malicious components. This means that malicious files belonging to the same malware "family", with the same forms of malicious behavior, are constantly modified and/or obfuscated using various tactics, so that they appear to be many different files.

Several methods have been proposed for automatic malware signature generation, e.g., signatures based on specific vulnerabilities, payloads, honeypots, etc. A major problem associated with these methods is that they target specific aspects of the malware, thus allowing the malware developers to create a new undetected variant by modifying small parts of their software. For example, a malware spreading through the use of a specific vulnerability found in Windows operating system, can use another vulnerability in the system to spread, thus evading vulnerability-based signatures.

Our Contributions. Malware detection represents binary classification problem with two classes, namely "malware" and "cleanware" or "benign". The assumption is that a behavioral difference is present for the two classes, making it possible to discriminate between malicious and benign behavior. We extract four kinds of features for the experimentation. First is the static features for malwares icollected using "theZoo"[1]. "theZoo" is a project created to make the possibility of malware analysis open and available to the public. Second, behavioral traces of malware and cleanware samples collected using a scalable and distributed malware analysis setup based on Cuckoo Sandbox. Cuckoo Sandbox represents a powerful malware analysis systems that is able to trace various client-level forensics by executing malware within Virtual Machines (VMs). Third, the behavioral traces obtained from Cuckoo Sandbox are used to generate abstract level text features. Finally, Behavior of programs (and specifically malware) is recorded by running the programs in a sandbox. A sandbox is a special environment which allows for logging the behavior of programs (e.g., the API function calls, their parameters, files created or deleted, websites and ports accessed, etc.) The results are saved in a file (typically a text file). We use a Deep Belief Network as the classifier/detector, and perform an ablation study by taking these datasets individually, and all possible combinations of the four datasets.

II. Related Works

In this section we present the state-of-the-art approaches that use static and dynamic analysis to perform either malware detection or family classification. We focus on methods that rely on machine learning as the tool of classifying malware. We do so as many existing work rely on machine learning for discriminating between malware families, based on the analysis of static and behavioral data.

[1] uses random forest for malware classification and detection. The authors use behavioral traces of 270,000 mal-

[1]https://github.com/ytisf/theZoo

ware samples, and 837 benign samples. They collect these behavioral traces using Cuckoo Sandbox environment to run the malware files and collect logs [2]. They analyzed the samples for up to 200 seconds, which is sufficient time to provide with enough behavioral data while maintaining efficiency of the setup. Malware evasion techniques were not considered in this project.

[2] Recent researches mainly use machine learning based methods heavily relying on domain knowledge for manually extracting malicious features. Authors propose MalNet, a novel malware detection method that learns features automatically from the raw data. First, they generate a grayscale image from malware file, meanwhile extracting its opcode sequences with the decompilation tool IDA. MalNet uses CNN and LSTM networks to learn from grayscale image and opcode sequence, respectively and takes a stacking ensemble for malware classification. MalNet achieves 99.88% validation accuracy for malware detection. In a malware classification task, MalNet outperforms most of the related works with 99.36% detection accuracy and achieves a considerable speed-up on detecting efficiency comparing with two state-of-the-art results on Microsoft malware dataset. Limited to .exe files

[3] The authors applied this approach to an end-to-end convolutional neural malware detector and present a high evasion rate. They introduce a novel approach for generating adversarial examples for discrete input sets, such as binaries. They modify malicious binaries by injecting a small sequence of bytes to the binary file. The modified files are detected as benign while preserving their malicious functionality. Basically, the authors proposed a generative adversarial network for Malware dataset. In this work, they analyze malware detectors based on Convolutional Neural Networks (CNNs). Such detectors are not based on handcrafted features, but on the entire raw binary sequence. Results suggest that E2E deep learning based models are susceptible to AE attacks.

[4] In this paper, the authors present a neural network based representation for addressing the open set recognition problem. In this representation instances from the same class are close to each other while instances from different classes are further apart. “open category learning”, terms such as “open world recognition” and “open set recognition” have been used in past literature. In this paper, they used the term “open set recognition”. Training data usually is incomplete because of novel malware families/classes that emerge regularly. As a result, malware classification systems operate in an open set scenario. The authors propose an approach for learning a representation that facilitates open set recognition, a loss function that enables us to use the same distance function both when training and when computing an outlier score, proposed approaches achieve statistically significant improvement compare to previous research work on three datasets. When the proposed approach is used for open set recognition, the final prediction is a class label, which can be with one of the K known class labels if the test instances has an outlier score less than a threshold value or it can be an “unknown” label if the instance has an outlier score greater than the threshold.

[2]https://cuckoosandbox.org/download

[5] The authors propose a CNN-based architecture to classify malware samples. They convert malware binaries to grayscale images and subsequently train a CNN for classification. This was the first research work to use CNNs for the task. Previous approaches used shallow models. Experimented on Malimg and MS Malware dataset. The method achieves better than state-of-the-art performance. The proposed method achieves 98.52% and 99.97% accuracy on the Malimg and Microsoft datasets respectively. The proposed approach is data independent and learns the discriminative representation from the data itself rather than depending on hand-crafted feature descriptors.

[6] learning a sequence regression function, i.e., a mapping from sequential observations to a numeric score. The resulting linear regression model provides the user with a list of the most predictive features selected during the learning stage, adding to the interpretability of the method. Other examples - DNA sequences (classify each sequence into sub-families). Methods aimed at solving such problems typically employ Hidden Markov Models (HMM), kernel Support Vector Machines (SVM) or more recently, Convolutional Neural Networks (CNN). While the accuracy of such techniques is promising, their efficiency and interpretability are still critical challenges. In this work, the authors propose a regression approach that can use the entire space of k-mers, of unlimited length, by learning a linear model using an iterative branch-and-bound strategy. The empirical study shows that we can achieve results comparable to the state-of-the-art, with a simple linear regression model, while employing little to no domain knowledge or pre-processing.

[4] Malware classification systems need to identify instances from unknown classes addition to discriminating between known classes. In this paper,the authors present a neural network based representation for addressing the open set recognition problem. In this representation instances from the same class are close to each other while instances from different classes are further apart, resulting in statistically significant improvement when compared to other approaches on three datasets from two different domains.

In [7], the authors use a deep neural network (DNN) as a classifier for nation-state APT attribution. They record the dynamic behavior of the APT when run in a sandbox and use it as raw input for the neural network, allowing the DNN to learn high level feature abstractions of the APTs itself. They also use the same raw features for APT family classification. Finally, they use the feature abstractions learned by the APT family classifier to solve the attribution problem. Using a test set of 1000 Chinese and Russian developed APTs, the model achieved an accuracy rate of 98.6%.

In [8],the authors investigate a Deep Learning based system for malware detection. In the investigation, they experiment with different combination of Deep Learning architectures including Auto-Encoders, and Deep Neural Networks with varying layers over Malicia malware dataset on which earlier studies have obtained an accuracy of (98%) with an acceptable False Positive Rates (1.07%). But these results were done using extensive man-made custom domain features and investing corresponding feature engineering and design efforts. In their proposed approach, besides improving the previous best results (99.21% accuracy and an False Positive Rate of 0.19%) indicates that Deep Learning based systems could deliver an effective defense against malware. Since it is good in automatically extracting higher conceptual features from the data, Deep Learning based systems could provide an effective, general and scalable mechanism for detection of existing and unknown malware.

In [9], the authors attempt to transfer performance improvements to model the malware system call sequences for the purpose of malware classification. They construct a neural network based on convolutional and recurrent network layers in order to obtain the best features for classification. This way they get a hierarchical feature extraction architecture that combines convolution of n-grams with full sequential modeling. The evaluation results demonstrate that the 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.

Table I shows a comparison among the current malware approaches and the data they user versus the proposed and the data it uses.

III. Dataset

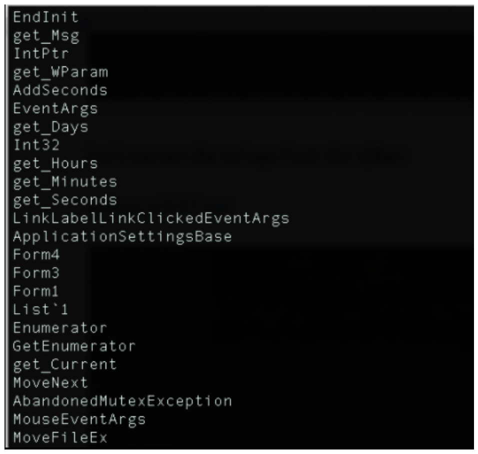
After going through an extensive literature survey on malware analysis, we picked up the one of the best classifiers available, and different kinds of datasets available to perform a combined and ablation study and see what features are most informative. For that, we collect the following datasets:

A. Static Features of Malware

The static features for malwares is collected using "theZoo"[3]. "theZoo" is a project created to make the possibility of malware analysis open and available to the public. They have found out that almost all versions of malware are very hard to come by in a way which will allow analysis, they have decided to gather all of them for you in an accessible and safe way. Static analysis gives us two things:

- Strings: Extracting strings from malicious software will give us many additional pieces of information about it and about its functionalities. Some useful pieces of information are: IP Addresses, Bitcoin wallet addresses, Error messages, comments and so on. Figure 1 shows an example.
- PE headers: The Portable Executable (PE) format is a file format for executables, object code, DLLs and others used in 32-bit and 64-bit versions of Windows operating systems. Inspecting PE headers will help us get more information about the malware including where the binary needs to be loaded into memory and so on. Figure 2 shows the complete structure of PE headers.
- Sequence: The sequence is the order in which API calls, including their arguments, are called on the VM. Two different representations have been made for sequence features. In Representation 1, the first 200 APIs are listed in the order in which they are called, where each API and its input argument are combined into one feature. In Representation 2, the API and its input argument are separated, such that 200 extra features are introduced. The input arguments used in this representation are only the ones that are connected with calls to new functions or APIs, else a "0" will be introduced.
- Frequency: The frequency is the number of occurrences of API calls during the analysis on the VM. For both feature representations, the same frequency techniques are used. For each API call a frequency is assigned in the representation. Furthermore, feature construction technique using the addition operator, produces 14 different behavioral API bins, where frequency for each API in the corresponding bin are summed. Finally, frequencies of the 25 most used DLL files are used.

[3]https://github.com/ytisf/theZoo



```
EndInit
get_Msg
IntPtr
get_WParam
AddSeconds
EventArgs
get_Days
Int32
get_Hours
get_Minutes
get_Seconds
LinkLabelLinkClickedEventArgs
ApplicationSettingsBase
Form4
Form3
Form1
List`1
Enumerator
GetEnumerator
get_Current
MoveNext
AbandonedMutexException
MouseEventArgs
MoveFileEx
```

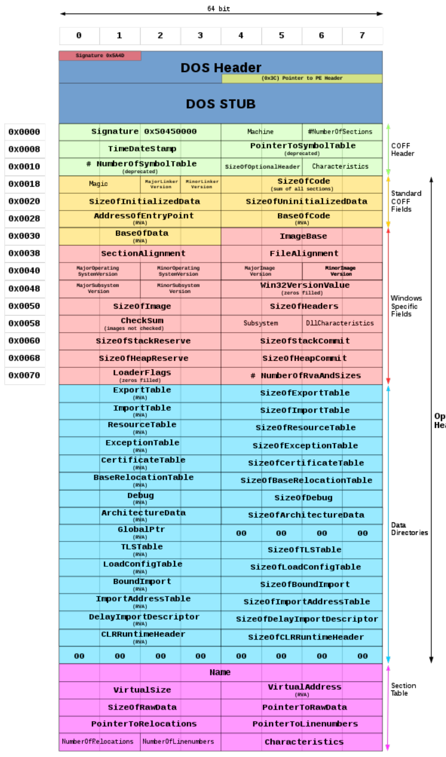
Fig. 1. Sample String Data from a Malware file

B. Behavioral Traces from Cuckoo Sandbox

In order to obtain behavioral traces of malware and cleanware samples we have developed a scalable and distributed malware analysis setup based on Cuckoo Sandbox. Cuckoo Sandbox represents a powerful malware analysis systems that is able to trace various client-level forensics by executing malware within Virtual Machines (VMs). In order to analyze a large amount of samples efficiently, we modified Cuckoo to work in a scalable and distributed manner. This was done such that commands to control the VMs, were send over a closed network using Secure Shell (SSH), making it possible to connect multiple machines as VM controllers and distribute the analysis. Furthermore, we deployed a server to collect all the analysis data from the VMs and store it in json files. Finally, the samples were analyzed for up to 200

TABLE I
COMPARISON WITH RELATED WORKS

Approach	Cuckoo + Feature Selection	Cuckoo text features	API call flow	Opcodes
An approach for detection and family classification of malware based on behavioral analysis [1]	Yes	No	No	No
End-to-End Deep Neural Networks and Transfer Learning for Automatic Analysis of Nation-State Malware [7]	No	Yes	No	No
Deep Learning for Classification of Malware System Call Sequences [9]	No	No	Yes	No
An investigation of a deep learning based malware detection system [8]	No	No	No	Yes
FeatAll: An All Feature Malware Detection Technique	Yes	Yes	Yes	Yes



64 bit
0 1 2 3 4 5 6 7
Signature 0x5A4D
DOS Header
(0x3C) Pointer to PE Header
DOS STUB
0x0000 Signature 0x50450000 Machine #NumberOfSections
0x0008 TimeDateStamp PointerToSymbolTable (deprecated)
0x0010 # NumberOfSymbolTable (deprecated) SizeOfOptionalHeader Characteristics
0x0018 Magic MajorLinker Version MinorLinker Version SizeOfCode (sum of all sections)
0x0020 SizeOfInitializedData SizeOfUninitializedData
0x0028 AddressOfEntryPoint (RVA) BaseOfCode (RVA)
0x0030 BaseOfData (RVA) ImageBase
0x0038 SectionAlignment FileAlignment
0x0040 MajorOperating SystemVersion MinorOperating SystemVersion MajorImage Version MinorImage Version
0x0048 MajorSubsystem Version MinorSubsystem Version Win32VersionValue (zeros filled)
0x0050 SizeOfImage SizeOfHeaders
0x0058 CheckSum (images not checked) Subsystem DllCharacteristics
0x0060 SizeOfStackReserve SizeOfStackCommit
0x0068 SizeOfHeapReserve SizeOfHeapCommit
0x0070 LoaderFlags (zeros filled) # NumberOfRvaAndSizes
ExportTable (RVA) SizeOfExportTable
ImportTable (RVA) SizeOfImportTable
ResourceTable (RVA) SizeOfResourceTable
ExceptionTable (RVA) SizeOfExceptionTable
CertificateTable (RVA) SizeOfCertificateTable
BaseRelocationTable (RVA) SizeOfBaseRelocationTable
Debug (RVA) SizeOfDebug
ArchitectureData (RVA) SizeOfArchitectureData
GlobalPtr (RVA) 00 00 00 00
TLSTable (RVA) SizeOfTLSTable
LoadConfigTable (RVA) SizeOfLoadConfigTable
BoundImport (RVA) SizeOfBoundImport
ImportAddressTable (RVA) SizeOfImportAddressTable
DelayImportDescriptor (RVA) SizeOfDelayImportDescriptor
CLRRuntimeHeader (RVA) SizeOfCLRRuntimeHeader
00 00 00 00 00 00 00 00
Name
VirtualSize VirtualAddress (RVA)
SizeOfRawData PointerToRawData
PointerToRelocations PointerToLinenumbers
NumberOfRelocations NumberOfLinenumbers Characteristics
COFF Header
Standard COFF Fields
Windows Specific Fields
Optional Header
Data Directories
Section Table

Fig. 2. PE header information in a malware file

seconds, which we believe is sufficient time to provide us with enough behavioral data while maintaining efficiency of the setup. Malware evasion techniques were not considered in this project.

C. Abstract Text Features

The behavioral traces obtained from Cuckoo Sandbox are used to generate abstract level text features. Specifically, we followed the following simple steps to convert sandbox files into fixed size inputs to the neural network: 1. Select as features the top 50,000 words with highest frequency in all Cuckoo reports, after removing the words which appear in all files. The rationale is that words which appear in all files, and words which are very uncommon do not contain lots of useful information. 2. Convert each sandbox file to a 50,000-sized bit string by checking whether each of the 50,000 words appear in it. That is, for each analyzed Cuckoo report, feature[i] = 1 if the i-th most common word appears in that cuckoo report, or 0 otherwise (word vectors).

D. Program Behaviour as Binary Vector

Behavior of programs (and specifically malware) is typically recorded by running the programs in a sandbox. A sandbox is a special environment which allows for logging the behavior of programs (e.g., the API function calls, their parameters, files created or deleted, websites and ports accessed, etc.) The results are saved in a file (typically a text file). Figure 1 shows a snippet of logs recorded by a sandbox. Sandbox records are usually analyzed manually, trying to learn information that would assist in creating a signature for the malware. Figure 3 shows a sample program call.

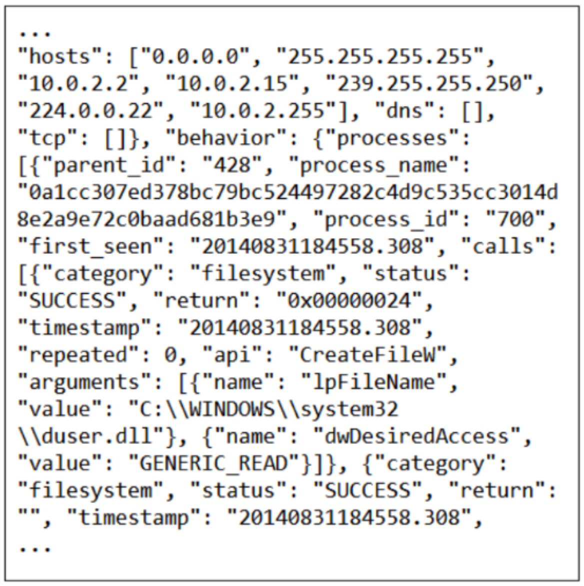

```
...
"hosts": ["0.0.0.0", "255.255.255.255",
"10.0.2.2", "10.0.2.15", "239.255.255.250",
"224.0.0.22", "10.0.2.255"], "dns": [],
"tcp": []}, "behavior": {"processes":
[{"parent_id": "428", "process_name":
"0a1cc307ed378bc79bc524497282c4d9c535cc3014d
8e2a9e72c0baad681b3e9", "process_id": "700",
"first_seen": "20140831184558.308", "calls":
[{"category": "filesystem", "status":
"SUCCESS", "return": "0x00000024",
"timestamp": "20140831184558.308",
"repeated": 0, "api": "CreateFileW",
"arguments": [{"name": "lpFileName",
"value": "C:\\WINDOWS\\system32
\\duser.dll"}, {"name": "dwDesiredAccess",
"value": "GENERIC_READ"}]}, {"category":
"filesystem", "status": "SUCCESS", "return":
"", "timestamp": "20140831184558.308",
...
```

Fig. 3. PE header information in a malware file

The simplest method for converting the sandbox generated text file to a fixed size string is using one of the methods common in natural language processing (NLP). Of these methods, the simplest yet is unigram (i-gram) extraction. For example, given a dataset of text samples, find the 5,000 most frequent words in the text (these words would comprise the dictionary), and then for each text sample check which of these 5,000 words are present. Thus, each text sample is represented as a 5,000 sized bit-string. Unlike language text files, sandbox files contain a variety of information, and require several preprocessing stages to extract the useful content (e.g., string after ”api” tag contains the name of function call, etc.). However, in order to remain as domain agnostic as possible, we propose to treat the sandbox file as a simple text file, and extract unigrams without any preprocessing. That is, all the markup and tagged part of the files are extracted as well (e.g., given ”api”: ”CreateFileW”, the terms extracted are ”api”: and ”CreateFileW”, completely ignoring what each part means). While this may sounds absurd (intentionally adding useless noise where it can be easily removed), this should not pose a problem, since the learning system (described below) should easily learn to ignore these irrelevant parts. Specifically, our method follows the following simple steps to convert sandbox files to fixed size inputs to the neural network:

- For each sandbox file in the dataset, extract all unigrams,
- Remove the unigrams which appear in all files (contain no information),
- For each unigram count the number of files in which it appears,
- Select top 20,000 with highest frequency, and
- convert each sandbox file to a 20,000 sized bit string, by checking whether each of the 20,000 unigrams appeared in it.

In other words, we first define which words (unigrams) participate in our dictionary (analogous to the dictionaries used in NLP, which usually consist of the most frequent words in a language), and then for each sample we check it against the dictionary for the presence of each word and thus produce a binary vector.

IV. Methodology

Malware Detection: Malware detection represents binary classification problem with two classes, namely “malware” and “cleanware”. The assumption is that a behavioral difference is present for the two classes, making it possible to discriminate between malicious and benign behavior. For each malware and clean ware sample a set of discriminative features is extracted.

The previous subsection described our data collection process for different sets. As we discussed previously, most malware variants make small changes in their code (i.e., small changes in behavior), which is sufficient to evade the classical signature generation methods. We would like to generate a signature for each program which is resilient to these small changes (an invariant representation, similar to those used for computer vision). In order to achieve this goal, we create a deep belief network (DBN) by training a deep stack of denoising autoencoders.

An autoencoder is an unsupervised neural network which sets the target values (of the output layer) to be equal to the inputs, i.e., the number of neurons at the input and output layers is equal, and the optimization goal for output neuron i is set to equal Xi, which is the value of the input neuron i. A hidden layer of neurons is used between the input and output layers, and the number of neurons in the hidden layer is usually set to fewer than those in the input and output layers, thus creating a bottleneck, with the intention of forcing the network to learn a higher level representation of the input. That is, for each input X, it is first mapped to a hidden layer y, and the output layer tried to reconstruct x. The weights of the encoder layer (W) and the weights of the decoder layer (W’) can be tied (i.e., defining W’ = WT). Autoencoders are typically trained using backpropagation with stochastic gradient descent.

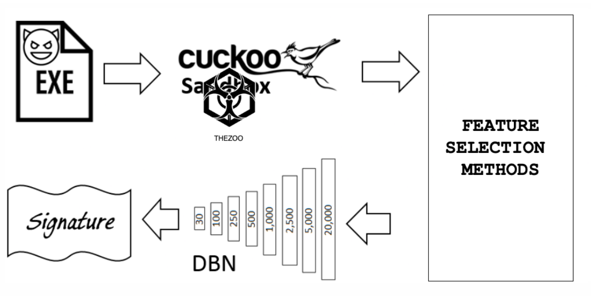


Fig. 4. Methodology followed for the study.

Recently it has been demonstrated that denoising autoencoders generalize much better than basic autoencoders in many tasks. In denoising autoencoders each time a sample is given to the network, a small portion (usually a ratio of about 0.1 to 0.2) of it is corrupted by adding noise (or more often by zeroing the values). That is, given an input X, first it is corrupted to x and then given to the input layer of the network. The objective function of the network in the output layer remains generating X, i.e., the uncorrupted version of the input. This approach usually works better than basic autoencoders due to diminishing the overfitting in the network. By having to recreate the uncorrupted version of the input, the network is forced to generalize better, and determine more high level patterns. Additionally, since the network rarely receives the same input pattern more than once (each time sees a corrupted version only), there is a diminished risk of overfitting (though it still takes place). Finally, using denoising autoencoders the hidden layer need not necessarily be smaller than the input layer (in basic autoencoder such a larger hidden layer may result in simply learning the identity function). Note that the noise is added only during training. In prediction time the network is given the uncorrupted input (i.e., similar to basic autoencoder).

When an autoencoder’s training is complete, we can discard the decoder layer, fix the values of the encoder layer (so the layer can no longer be modified), and treat the output. One layer of denoising autoencoder during training. the hidden layer as the input to a new autoencoder added on top of the previous autoencoder. This new autoencoder can be trained similarly. Using such layer-wise unsupervised training, deep

TABLE II
ABLATION STUDY PERFORMED BY RUNNING THE CHOSEN MODEL WITH/WITHOUT ONE OR MORE DATASETS TO STUDY THE IMPACT OF DIFFERENT FEATURES ON THE ACCURACY VALUES.

Dataset	Size of feature vector	Accuracy
Static Features of Malware (A)	5000	87.2
Behavioral Traces from Cuckoo Sandbox (B)	55,000	89.9
Abstract Text Features (C)	50,000	86.45
Program Behaviour as Binary Vector (D)	20,000	91.9
A + B	60,000	92.2
B + C	105,000	92.5
C + D	70,000	92.9
A + D	25,000	93.6
A + B + C	110,000	93.6
B + C + D	125,000	89.9
A + C + D	75,000	88.7
A + B + D	80,000	88.1
A + B + C + D	130,000	90.1

stacks of autoencoders can be assembled to create deep neural networks consisting of several hidden layers (forming a deep belief network). Given an input, it will be passed through this deep network, resulting in high level outputs. In a typical implementation, the outputs may then be used for supervised classification if required, serving as a compact higher level representation of the data.

We collected data from Cuckoo and theZoo, form features from the data as explained in the dataset section, then use the Deep Belief Network for experimentation. To show the importance of each dataset we performed an ablation study to show the accuracies on each kinds of data.

V. EXPERIMENTS

In our approach we train a deep denoising autoencoder consisting of eight layer: X-5,000-2,500-1,000-500-250- 100-30. X represents the size of vector depending on the chosen dataset. At each step only one layer is trained, then the weights are "frozen", and the subsequent layer is trained, etc. At the end of this training phase, we have a deep network which is capable of converting the X input vector into 30 floating point values. We regard these 30-sized vector as the "signature" of the program. Note that the network is trained only using the samples in the training set, and for all future samples it will be run in prediction mode, i.e., receiving the X-sized vector it will produce 30 output values, without modifying the weights. We have trained the model on 40 million packets (20 million from each malware and benign class). After training we predicted and found more bias for benign and hence ran for 20 million more packets in attack.

VI. DISCUSSION

The tools that perform malware detection and classification are limited by the knowledge they have. Plenty of them are available with a plethora of techniques ranging from supervised shallow models to deep learning to unsupervised deep belief networks. However, due to availability of several feature selection methods, feature importance is always overlooked upon. In this work, we use a state-of-the-art Deep Belief Network implementation with different sets of data features. We perform an ablation study with all combinations of this data to see what features give a better representation. Specifically we collect four kinds of datasets: static features, abstract text features, behavioral traces and program behaviour. We observe through experiments that the combination of static features, abstract text features and behavioral traces yields the best results and therefore are more important that program behaviour data.

REFERENCES

[1] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," in *2016 International conference on computing, networking and communications (ICNC)*. IEEE, 2016, pp. 1–5.

[2] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, vol. 2018, 2018.

[3] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.

[4] M. Hassen and P. K. Chan, "Learning a neural-network-based representation for open set recognition," in *Proceedings of the 2020 SIAM International Conference on Data Mining*. SIAM, 2020, pp. 154–162.

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

[6] S. Gsponer, B. Smyth, and G. Ifrim, "Efficient sequence regression by learning linear models in all-subsequence space," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017, pp. 37–52.

[7] I. Rosenberg, G. Sicard, and E. O. David, "End-to-end deep neural networks and transfer learning for automatic analysis of nation-state malware," *Entropy*, vol. 20, no. 5, p. 390, 2018.

[8] M. Sewak, 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*, 2018, pp. 1–5.

[9] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.