Neurlux: Dynamic Malware Analysis Without Feature Engineering

Chani Jindal
University of California, Santa Barbara
Appfolio
chanijindal@ucsb.edu

Christopher Salls
University of California, Santa Barbara
salls@cs.ucsb.edu

Hojjat Aghakhani
University of California, Santa Barbara
hojjat@cs.ucsb.edu

Keith Long
University of California, Santa Barbara
klong@ucsb.edu

Christopher Kruegel
University of California, Santa Barbara
Lastline
chris@cs.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
Lastline
vigna@cs.ucsb.edu

ABSTRACT

Malware detection plays a vital role in computer security. Modern machine learning approaches have been centered around domain knowledge for extracting malicious features. However, many potential features can be used, and it is time consuming and difficult to manually identify the best features, especially given the diverse nature of malware.

In this paper, we propose Neurlux, a neural network for malware detection. Neurlux does not rely on any feature engineering, rather it learns automatically from dynamic analysis reports that detail behavioral information. Our model borrows ideas from the field of document classification, using word sequences present in the reports to predict if a report is from a malicious binary or not. We investigate the learned features of our model and show which components of the reports it tends to give the highest importance. Then, we evaluate our approach on two different datasets and report formats, showing that Neurlux improves on the state of the art and can effectively learn from the dynamic analysis reports. Furthermore, we show that our approach is portable to other malware analysis environments and generalizes to different datasets.

CCS CONCEPTS

• **Security and privacy → Software and application security**;
• **Computing methodologies → Neural networks**.

KEYWORDS

Dynamic malware analysis, Machine learning, deep learning

ACM Reference Format:
Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. 2019. Neurlux: Dynamic Malware Analysis Without Feature Engineering. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3359789.3359835

ACSAC '19, December 9–13, 2019, San Juan, PR, USA

ACM ISBN 978-1-4503-7628-0/19/12…$15.00
https://doi.org/10.1145/3359789.3359835

1 INTRODUCTION

As malware becomes more sophisticated, malware analysis needs to evolve as well. Traditionally, most anti-malware software uses signature-based detection, which cross-references executable files with a list of known malware signatures. However, this approach has limitations, since any changes to malware can change the signature, so new releases of the same malware can often evade signature-based detection by encrypting, obfuscating, packing, or recompiling the original sample. VirusTotal reports that over 680,000 new samples are analyzed per day [40], of which potentially a significant number of samples are just re-packed versions of previously seen samples, as Brosch et al. [3] observed more than 50% of new malware are simply re-packed versions of existing malware.

In recent years, the need for techniques that generalize to previously unseen malware samples has led to detection approaches that utilize machine learning techniques [25, 26, 38]. Malware analysis can be broadly divided into two categories: code (static) analysis and behavioral (dynamic) analysis. Both static and dynamic analysis have their advantages and disadvantages. Although dynamic analysis provides a clear picture of the executable behavior, it faces some problems in practice. For example, dynamic analysis of untrusted code requires a virtual machine that replicates the target host, which requires a substantial amount of computing resources. Besides, malware may not exhibit its malicious behavior, or the virtualized environment may not reflect the environment targeted by the malware [5, 15, 27, 30].

To avoid such limitations, some related work relies only on features extracted from static analysis to achieve rapid detection for a large number of malware samples. However, various encryption and obfuscation techniques can be employed to hinder static analysis [19, 24]. This becomes a more severe problem for static malware detectors, since packing is also in widespread use in benign samples today. samples [28]. Although dynamic analysis is shown to be susceptible to evasion techniques, run-time behavior is hard to obfuscate. Dynamically analyzing a binary gives the ability to unpack and record its interactions with the OS which it an attractive choice for malware analysis.

Regardless of the use of static analysis or dynamic analysis, most machine-learning based malware detectors rely heavily on relevant

domain knowledge [12, 13, 34]. These approaches often rely on features that are investigated manually by malware experts, which requires a vast amount of feature engineering. For example, Kolbitsch et al. [13] captured the behavior graphs of PE executables in specific features designed for this purpose. Malware is continually being created, updated, and changed, which can make the original well-designed features not applicable to newer malware or different malware families. In this case, the costly feature engineering work has to be refined continuously. Hence, it is crucial to find a way toreduce the cost of artificial feature engineering to extract usefulinformation from raw data.

There has recently been some work on deep learning based malware classification which does not require feature engineering. However, existing deep learning approaches do not leverage the information from already-available dynamic analysis systems, instead tending to pick one type of dynamic feature [14] or use static features [6]. These solutions miss out on the complete information concerning what actions are taken by each sample.

In this paper we propose Neurlux, a system that uses neural networks to analyze dynamic analysis reports. Services such as Cuckoo [21] provide a detailed dynamic analysis of an executable by tracing it in a sandbox. This analysis contains information, such as network activity, changes to the registry, file actions, and more. We use such reports as the basis for our analysis. That is, given a dynamic analysis report, we want to be able to predict whether or not the report is for a malware sample or a benign executable.

Our intuition is that we can treat these reports as documents. With this intuition, we present Neurlux, a neural network which learns and operates on the (cleaned) dynamic analysis report without needing any feature engineering. Neurlux borrows concepts from the field of document classification, treating the report as a sequence of words, which make sequences of sentences, to create a useful model. Neurlux intends to replace expensive hand-crafted heuristics with a neural network that learns these behavioral artifacts or heuristics.

To check if our method is biased to a particular report format (i.e., sandbox), we included in our evaluation two different sandboxes, the Cuckoo sandbox [21], *CuckooSandbox* and a commercial anti-malware vendor's sandbox, which we will refer to as *VendorSandbox*. In addition, we used two different datasets, one provided by the commercial anti-malware vendor, *VendorDataset* along with the labeled benchmark dataset EMBER [2], *EmberDataset*.

To show that Neurlux does better than feature engineering approaches, we implement and compare against three such techniques which are discussed later. Furthermore, we implement and compare against MalDy, a model proposed by Karbab et al. [11], as a baseline. MalDy formalizes the behavioral (dynamic) report into a bag of words (BoW) where the features are words from the report.

In summary, we make the following contributions:

- We propose Neurlux, an approach which leverages document classification concepts to detect malware based on the behavioral (dynamic) report generated by a sandbox without the need for feature engineering. The only preprocessing step is cleaning the reports to extract words, upon which our model learns relevant sequences of words which can aid its prediction. Neurlux shows high accuracy achieving 96.8% testing accuracy, in our K-fold validation.
- We create and test several approches for malware classification on dynamic analysis reports, including novel methods such as, a Stacking Ensemble for Integrated Features, and a Feature Counts model. We compared with these, showing Neurlux outperforms approaches with feature engineering.
- We assess the generalization ability of Neurlux by testing it against a new dataset and also a new report format, i.e., generated by a new sandbox and show that it generalizes better than the methods we evaluated against.
- The source code and dataset of executables will be released on github.

2 BACKGROUND

Some related work adopted Natural Language Processing techniques for malware classification such as MalDy [11], which formalizes the behavioral report of a sample into a bag of words. In this section, we explain such techniques that we exploited to build Neurlux and other models as a baseline for comparison to Neurlux.

2.1 Word Embeddings

Word embeddings are translations from words to vectors that aim to give words with similar meaning corresponding vectors that are close in the feature space. Word embeddings are frequently found as the first data processing layer in a deep learning model that processes words [10]. This is because grouping vectors by meaning gives a deep learning model an initial correlation of words. These embeddings are frequently pre-trained or based on another model such as word2vec [18]. However, in our case, we do not use a pre-trained embedding as the similarities of "words" (i.e., file paths, mutexes, etc.) differ from ordinary English.

2.2 Embedding Visualization

Dimensionality reduction methods are used to convert high dimensional data into lower dimensional data. We can use dimensionality reduction to convert the data into two dimensions, allowing us to show the distribution of the data in a scatter plot. To do this, we choose to use t-Distributed Stochastic Neighbor Embedding (t-SNE) [17], a technique for visualization of similarity data. t-SNE preserves the local structure of the data and some global structure, such as clusters, while reducing the dimensionality.

2.3 CNN for text classification

Convolutional neural networks (CNN) have recently shown to be very useful in text classification. A typical model for this represents each input as a series of n sequences, where each sequence is a d-dimensional vector; thus input is a feature map of dimensions $n \times d$. The model starts by mapping words to vectors, as discussed in Section 2.1. Then, convolutional layers are used for representation learning from sliding $k - grams$.

To extract higher level features from input vectors, a CNN applies filters of $R^{k \times d}$ on an input of length n, $\{x_1, x_2, x_3, x_4, x_i ....x_n\}$. After applying a filter of size k we have, $\{x_{1:k}, x_{2:k+1}, x_{3:k+2}, ...x_{n-k+1:n}\}$. Embeddings for x_i, $i < 1$ or $i > n$, are zero padded. For each window, $x_{i:i+k-1}$, a feature p_i is generated which is then fed into ReLU

non-linearity.

$$p_i = f(W.k^T + b) \tag{1}$$

Where $b \in R$ is a bias term, f is a non-linear activation function, such as the hyperbolic tangent, and $W.k$ is the weights for filter k. Applying filter k to all windows results in the feature map.

Max pooling sub-samples the input by applying a max operation on each sample. It extracts the most salient n-gram features across the sentence in a translation-invariant manner. The extracted feature can be added anywhere in the final sentence representation.

In practice we use multiple window sizes and multiple convolutional layers in parallel. A combination of convolution layers followed by max pooling is often used to create deep CNN networks. Sequential convolutions can improve the sentence mining process by capturing an abstract representation which is also semantically rich.

2.4 LSTM/ BiLSTM

Recursive Neural Networks (RNN) have gained popularity with text classification due to their ability to preserve sequence information over time. LSTM networks [9] overcome the vanishing gradient problem of RNN [8]. LSTM networks use an adaptive gating, which regulates the flow of information from the previous state and the extracted features of the current data input. For an input sequence with n entries: $x_1, x_2, ..., x_n$, an LSTM network processes it word by word. Then, it uses the following equations to update the memory p_t and hidden state h_t at time-step t:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ q_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} W.\left[h_{t-1}, x_t\right] \tag{2}$$

$$p_t = f_t * p_{t-1} + i_t * q_t \tag{3}$$

$$h_t = o_t * tanh(p_t) \tag{4}$$

where x_t is the input at time-step t, i_t is the input gate activation, f_t is the forget gate activation, and o_t is the output gate activation. All gates are generated by a sigmoid function, σ over the ensemble of input x_t and the preceding hidden state h_{t-1}.

A BiLSTM network extends the unidirectional LSTM by initiating a second hidden layer. In this layer, the hidden-to-hidden connections can flow in reverse temporal order. Therefore, the model holds information from both the past and the future. The output of j^{th} word can be represented as:

$$h_j = \left[\overrightarrow{h_j} \bigoplus \overleftarrow{h_j}\right] \tag{5}$$

2.5 Attention

It is evident that not all words contribute equally to the malicious or benign attributes of dynamic behavior. Hence, at the word level, an attention mechanism [39] can be used to extract malicious features/words that are important to the behavior classification. Finally, we aggregate the representations of those malicious features to form the sentence representation.

Let $H \in R^{d \times n}$ be a matrix of hidden vectors $[h_1, h_2, ..., h_n]$ that the LSTM network produced, where d is the size of the hidden layers, and n is the number of words in a sentence. Let $h_{it} \in H$ represent a hidden state. The first step is to feed h_{it} through a single-layer Perceptron network to get u_{it} as its hidden representation:

$$u_{it} = tanh(W_w h_{it} + b_w). \tag{6}$$

The second step is to initialize the context vector u_w randomly. Then, the importance of the word as the similarity of u_{it} with a word-level context vector u_w is measured. This gives a normalized importance weight vector α through a softmax function. The Context vector u_w is learned during the training process. α measures the importance of i^{th} word for malicious behavior.

$$\alpha_{it} = \frac{exp(u_{it}^\top u_w)}{\sum_t exp(u_{it}^\top u_w)} \tag{7}$$

In the end the sentence can be represented as the weighted hidden vector r:

$$r = H\alpha^T. \tag{8}$$

3 APPROACH

In this section, we describe our proposed method, Neurlux, a model which treats the dynamic behavior classification problem in a way similar to a document classification problem. The steps of this approach are as follows:

- **Data cleaning:** We want to treat the reports as a document classification task, so the first step is to clean the JSON formatted reports so that it is structured less like a JSON document and more like sequences of words, which makes up sequences of sentences. To do this, we first remove special characters, such as the brackets which are part of the JSON structure. Then the document is tokenized to extract words, of which the top 10,000 most common words are converted into numerical sequences.
- **Data Formatting:** The naïve method of converting words to vectors assigns each word with a one-hot vector. This vector would be all zeros except one unique index for each word. This kind of word representation can lead to substantial data sparsity and usually means that we may need more data to train statistical models successfully. This can be fixed by continuous vector space representation of words. To be more specific, we want semantically similar words to be mapped to nearby points, thus making the representation encode useful information about the words' actual meaning.
  Therefore, we use trainable word embeddings in Neurlux, which can have the property that similar words have similar vectors as described in Section 2.1. This way, the model can cluster words based on their usage patterns and they can provide more meaningful inputs to the later layers in the model.
- **Model:** We use the combination of CNN, BiLSTM networks, and Attention networks to create a model that understands the hidden lexical patterns in the malicious and benign reports. This model is designed using concepts from document classification. For example, an important idea is that not all words in a sentence are equally important, so it uses the attention mechanism to recognize and extract important words [41]. Another aspect is that context is important for

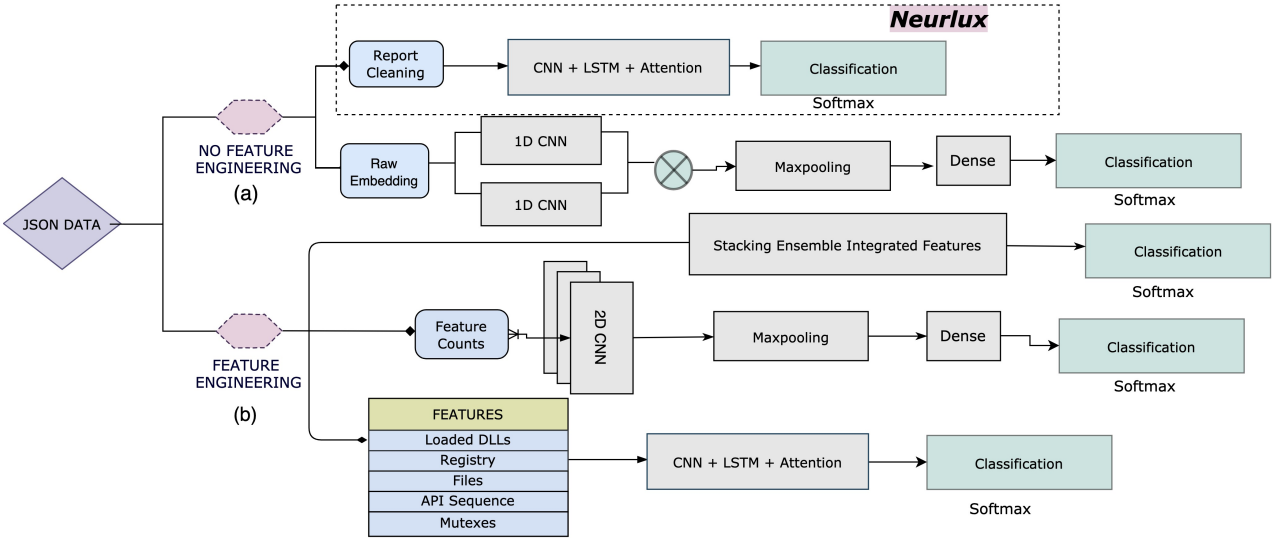

Figure 1: Overview of Models

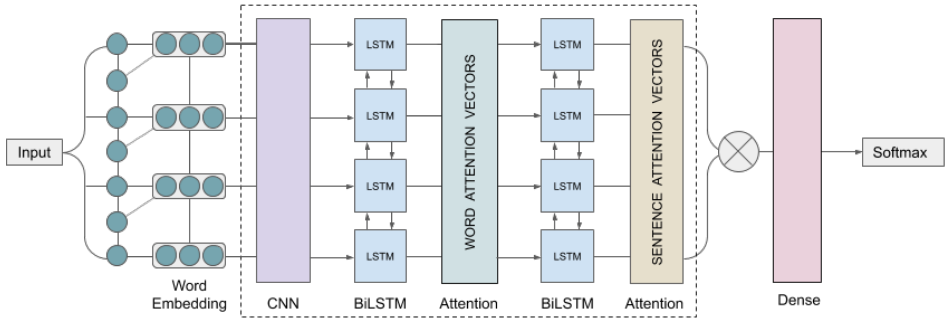

Figure 2: CNN-BiLSTM-Attention model for document classification

understanding the meaning of words, therefore we use BiLSTM to give context to the model. This model is described in detail in the next section (3.0.1).

3.0.1 ***CNN + BiLSTM + Attention***. Inspired by the previous document classification methods, we create the model illustrated in Figure 2, which consists of a convolutional neural network layer (CNN), and two pairs of bi-directional long short term memory network (BiLSTM) and attention layers. Convolutional neural networks (CNN) extract local and deep features from the input text. Then we obtain the high-level representation from the bidirectional LSTM network by using the hidden units from both the forward and backward LSTM. The CNN and LSTM combination is useful to extract local higher level features, which the LSTM can then find temporal relationships between [45]. The two pairs of BiLSTM and Attention are inspired by hierarchical attention networks [41]. The input is a trainable word embedding of dimension 256 to allow the model to cluster similar words as it learns. Each of these components is described thoroughly in the Section 2.

As described before, different parts of the report have different importance for determining the overall malicious behavior of a binary. For example, some parts of a registry key can be decisive, while others are irrelevant. Suppose we have the sentence attention score A_i for each sentence $s_i \in x$, and the word attention score $a_{i,j}$, for each word $w_{i,j} \in s_i$; both scores are normalized which satisfy the following equations,

$$\sum_i A_i = 1 \quad and \quad \sum_j a_{i,j} = 1. \tag{9}$$

The sentence attention measures which sentence is more important for the overall behavior while the word attention captures behavior signals such as the behavior words in each sentence. Therefore, the document representation r for document x is calculated as

Table 1: Overview of the models that we create and compare against Neurlux.

Model	Feature Engineering	Description
Counts Model	Yes	Counts of each feature
Individual Model	Yes	Document classification on individual extracted features
Ensemble Model	Yes	Ensemble of individual features
MalDy	No	State of the art model for report classification from [11]
Raw Model	No	Model trained on raw bytes
Neurlux	**No**	**Document classification on whole report**

follows,

$$r = \sum_i \left[A_i . \sum_j \left(a_{i,j} . h_{i,j} \right) \right]. \tag{10}$$

Finally, Neurlux outputs a classification decision as a score from 0-1 where 0 is benign, and 1 is malicious.

4 COMPARISON METHODS

In this section, we describe various models with which we compare. We compare with a previous state of the art method and with a couple approaches which involve feature engineering to check if we can actually do better than feature engineering approaches. An overview of the approaches we compare against are shown in Table 1.

4.1 Comparison With a State-of-the-Art Model

We used the method described in *MalDy* [11], as a model for comparison. Their approach is to preprocess sandbox reports with standard Natural Language Processing (NLP) techniques and then create an ensemble supervised machine learning (ML) model from a multitude of different ML algorithms. They attempt to formalize the behavioral reports in a way agnostic to the execution environment. This is done on both Win32 and Android. They argue that the key to their success is using their bag of words (BOW) model with Common N-Grams (CNG). CNG effectively computes the contiguous sequences of n items where n is an adjustable hyper-parameter. Instead of using single words (1-grams), using n-grams aids in finding distinct features. Once the reports are in a list of n-gram strings, they carry out two different vectorization approaches: TF-IDF and Feature Hashing. Feature Hashing creates fixed length feature vectors from sparse input n-grams. A hash is taken of each n-gram, and if the value is found within the table, it is incremented; otherwise, a value of 1 is added to the table. This process creates probabilistically unique vectors, given that the hash bucket size is sufficiently large. These vectors are subsequently fed into the ML models. We implement and use their best performing model as a comparison. In our evaluation (Section 6), this model reaches a plateau with 89.23% accuracy and an F-1 score of 88.5%.

A weakness of the BOW approach used in *MalDy* is that it does not take into account the context, just the frequencies with which words appear [22].

4.2 Raw JSON Data

A more basic deep learning approach is to learn from raw bytes, treating it as an image classification problem. Although the structure of the input data is defined, the placement of different string objects within the file is not ordered. To best capture such high-level location invariance, we choose to use a convolution network architecture. Combining the convolutional activations with a global max-pooling, followed by fully-connected layers allows this model to produce its activation regardless of the location of the detected features.

The *Raw Model* was inspired by an earlier approach on byte classification [26]. First, we clean the document, removing special characters. Then the bytes are extracted as integer values then padded to fix length to form a vector x of d elements. This ensures that regardless of the length of the input file, the input vector provided to the network has a fixed dimensionality. Each byte x_j is then embedded as a vector $z_j = \phi(x_j)$ of eight elements (the network learns a fixed mapping during training). This amounts to encoding x as a matrix $Z \epsilon R^{[d \times 8]}$. Figure 1($a$) shows an outline of the model used for raw JSON data binary classification. Then, it goes through the convolutional layers to eventually produce a classification between 0 and 1.

4.3 Features for Engineering Approaches

For the feature engineering approach comparisons, we begin by categorizing the six main categories of features available in the reports. These features are described in more detail below.

- **API Sequence Calls.** The reports typically include all system calls and their arguments stored in a representation tailored explicitly to behavior-based analysis. Much of the past work on behavior analysis has focused on using API call sequences for malware classification [23, 32, 35].
- **Mutexes.** Mutexes control the simultaneous access of the system resources. They are used by malware creators to avoid infecting a system more than once, and coordinate between processes [42].
- **File System Changes.** The interaction of a malware sample with the host file system might be a good indicator to determine malicious behavior. We consider all the important file operations such as create, read, write, modify, delete, etc.
- **Registry Changes.** The registry is a core part of Windows and contains a plethora of raw data. Registry keys can reveal much information about the system, but the true challenge is in unraveling which modifications to the registry are malicious and which are legitimate. The registry also represents a fundamental tool to hook into the operating system to gain persistence. Discovering what keys are queried, created, deleted, and modified can shed light on many significant characteristics of a sample.
- **Loaded DLLs.** The reports contain the shared library code loaded by a program. Nearly every executable program imports DLLs during execution. These DLLs can give insights into the types of APIs used by the program.

Figure 4 is an example of a *CuckooSandbox* report for a malicious sample that shows the various behavioral features cuckoo identifies. We obtain 28 such different features from *CuckooSandbox* and 43

features from *VendorSandbox*. These features are mapped based on semantic similarities and divided into 6 main behavioral groups as described above. The following sections give an exhaustive description of the various feature engineering techniques used. They are shown in Figure 1(a).

4.4 Feature Counts Model

In this section, we discuss an approach to use a neural network on shallow numerical features. Numerical features here are simply the counts of each event that was recorded, e.g., number of registry reads, number of file writes, etc. The first step is to parse the reports and extract all the available features. The number of features extracted differs due to the structural differences in dynamic analysis reports collected from *CuckooSandbox* and *VendorSandbox*. Each report lists features according to the parent process and child processes (any process that was either spawned by or tampered with by the primary process). Each process has its own set of individual features. Since each executable can contain one or more processes, the final representation of input features per sample will be:

$$\mathrm{S} = processes \times features$$

which expands to

$$\mathrm{S} = \begin{array}{c} \begin{matrix} reg\_read & file\_write & \cdots & mutex \end{matrix} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \begin{matrix} 1 \\ 2 \\ \vdots \\ n \end{matrix} \end{array} \tag{11}$$

(Columns are features, rows are processes)

The data representation is similar to that of a gray-scale image; therefore, a 2D CNN can be used for training on this dataset. We use an 8-layer deep CNN model inspired by Simonyan et al. [36]. The model consists of 8 convolution layers and 2 fully-connected layers. Every convolutional filter has a kernel size of 3, 4, or 5 with a stride of 1 and pooling region of 3x3 without overlap. A pooling function is applied to each feature map to induce a fixed-length vector. These fixed-length, top-level feature vectors generated from filter maps are then fed through a softmax function to generate the final classification. Figure 1 gives an overview of this model.

4.5 Text-Based Individual Feature Model

Each analysis report is a collection of statements, and each statement is a sequence of words. We believe that these sequences can give a more granular description of the actual events, compared to the features count method discussed in the previous section. The assumption for the text classification approach described in this section is that the difference between malicious and benign behavior of binaries could be translated into sequences present in the reports. In other words, the sequence of actions better represents if a binary is malicious or not than merely the number of actions. Additionally, we were looking for a feature representation (sequences of words) that uses an automatic feature extraction without the intervention of a security expert.

The input generation process can be divided into four steps. This process is performed iteratively for all six feature groups.

- **Feature Selection:** Different features are selected from the feature pool based on the top six behavioral groups discussed earlier in this section.
- **Data cleaning:** Similar to our method for Neurlux, we need to remove special characters and perform tokenization to extract numerical sequences.
- **Data Formatting:** As discussed previously, we want to have a continuous vector space representation of words, with semantically similar words mapped to a nearby point. So once again we use a trainable word embedding.
- **Model training:** We use the combination of CNN, BiLSTM, and Attention networks to create a model that understands the hidden lexical patterns in the malicious and benign reports. This model is described in detail in Section 3.0.1

4.6 Integrated Features using Ensemble

When the neural network, described in Section 3.0.1, is trained on individual behavioral feature types (such as mutexes or api calls), it exhibits a high variance depending on the feature. This variance can be attributed to the importance and contribution of each feature extracted from the reports. Therefore, in this section, we describe a way to use ensemble learning to combine multiple models to get a low variance and better predictive performance than any single constituent algorithm alone. Specifically, we use a stacking ensemble which uses one large multi-headed neural network and learns how to best integrate predictions from each input sub-network.

We start with five separately trained models from the previous section. These models are trained on the five most important behavior traits observed in the reports; namely, API Sequences, Mutex Operations, File Operations, Registry Operations, and DLLs Loaded. Each model uses the text-based feature classification method explained in Section 4.5. The outputs from all the sub-models are concatenated, and provided to a fully connected layer that acts as a meta-learner, and makes its probabilistic predictions. The sub-models are not trainable; therefore, their weights are not changed during the training, and only the weights of new hidden and output layers are updated.

Merging the outputs from multiple neural networks adds a bias that in turn counters the variance of a single trained neural network model. As a result, the final predictions are less sensitive to the specifics of training data and are more generalized.

5 DATASETS

We have employed two different datasets for evaluating our research work. Dataset 1 (*VendorDataset*) is a set of 27,540 Windows x86 binaries with 13,760 benign and 13,760 malicious files. This private dataset was randomly selected from an original pool that was analyzed by the anti-malware vendor's sandbox in the US during the period from 2017-05-15 to 2017-09-19. This security vendor provides a sandbox that runs executables, and collects full analysis results that outline what the sample does while running inside an isolated operating system. Dataset 2 (*EmberDataset*) is a subset of the publicly available Ember dataset[2]. It consists of 42,000 Windows x86 binaries with 21,000 benign and 21,000 malicious samples.

Malware Sample 1

windows	nt	terminal	services	maxdiscon...	microsoft
windows	defender	spynet	spynetreporting	microsoft	windows
defender	exclusions	paths	c	users	administrator
appdata	roaming	alfsvwjb	policies	microsoft	windows
nt	terminal	services	maxidletime	microsoft	windows
currentversion	explorer	advanced	showsuperhidden		

Table 2: Attention for registry keys written shown for a malware sample that the model classified correctly as malicious. The cells are colored by how much attention each word received, with colors: veryhigh, high, medium, low, and verylow.

#	Malware Family
1047	Win32.Virtob.Gen.12.Dam
346	Trojan.VIZ.Gen.1
285	Gen:Heur.PonyStealer.2
220	NSIS:Solimba-H [PUP]
201	Win32.Trojan.WisdomEyes.16070401.9500.9991
189	Gen:Variant.Zusy.189695
176	Gen:Trojan.Brresmon.Gen.1
168	Gen:Variant.Application.Bundler.DownloadGuide.48
162	Trojan.Win32.Generic!BT
146	Win32:Malware-gen
144	Trojan.Ransom.WannaCryptor.H
132	Gen:Variant.Adware.Kazy.3692
128	Win32.Virlock.Gen.8
111	ELF:Androot-I [PUP]
105	Gen:Heur.ZOF.2
104	Gen:Variant.Symmi.582
103	Trojan/Avanzado
97	Win32.Virtob.Gen.12
89	Gen:Variant.FakeAlert.93
87	Trojan.Generic.8995937
84	Gen:Variant.Symmi.439
81	Adware.Downware.1125
70	Gen:Variant.FakeAlert.88
66	Trojan[Packed]/Win32.PolyCrypt
66	MemScan:Trojan.Agent.BYFH

Figure 3: Top 25 Malware Families in *VendorDataset* and count(#)

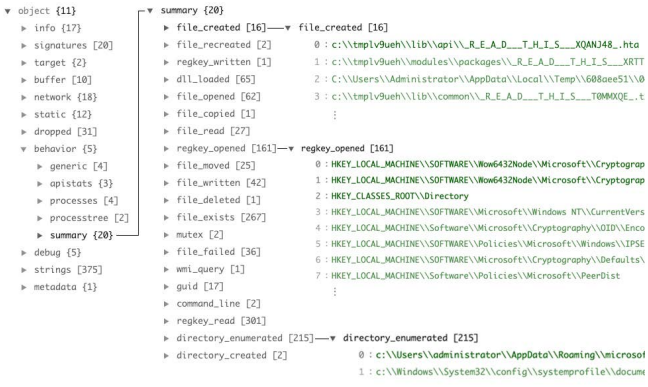

Figure 4: Cuckoo Sandbox Report format example

VendorDataset has over a 1000 malware families, top 25 of which are shown in the Figure 3.

Each binary is accompanied by two versions of detailed behavioral analysis reports in a JSON format. One behavioral report, *CuckooSandbox*, is collected using the Cuckoo sandbox [1]. Cuckoo is a publicly available sandbox which can be used to trace execution in a virtualized environment. It generates a JSON report of the actions taken by the binary during runtime. The other report, *VendorSandbox*, is collected using the sandbox from the security vendor, which also traces execution and collects information about the runtime execution of the executable. This results in two datasets with two different report formats for a total of four different combinations of datasets and reports.

5.1 Comparison of Reports

We ran an initial analysis of the two reports to understand their structure and the features they contain, which can be used for dynamic malware analysis. The format of one JSON report is shown in Figure 4. When examining the reports, we noticed that parts of the reports could be quite different for identical executables. The number of registry actions, file actions, and even the actual paths tended to differ. Differences show up because of differences in the sandbox and execution environment; even how long the executable is run influences the data in the reports.

The feature names do not match up exactly, so we try to draw parallels in features from the respective sandboxes. For example, *VendorSandbox* references "loaded_libraries", whereas *CuckooSandbox* uses "dll_loaded", but semantically they are the same. Due to subtle differences between the sandbox environments and formats, strings to do not match exactly. For example, when looking at registry keys, in the *VendorSandbox* we observe a key starting `hkcu\software\microsoft\windows`, while in Cuckoo it shows as `hkey_current_user\software\ microsoft\ windows`.

This shows that the reports are similar but not identical, and thus, our model needs to be robust enough to handle the discrepancies. We will later evaluate how robust various approaches are to these differences between the two sandbox reports.

6 EVALUATION

In this section, we evaluate Neurlux, which we described in Section 3. We compare Neurlux to approaches with feature engineering, which are described in Section 4.3. We also compare Neurlux with MalDy [11]. In particular, we attempt to answer the following research questions:

RQ1: Can deep learning methods without feature engineering identify malware from dynamic analysis reports as effectively as methods with feature engineering?

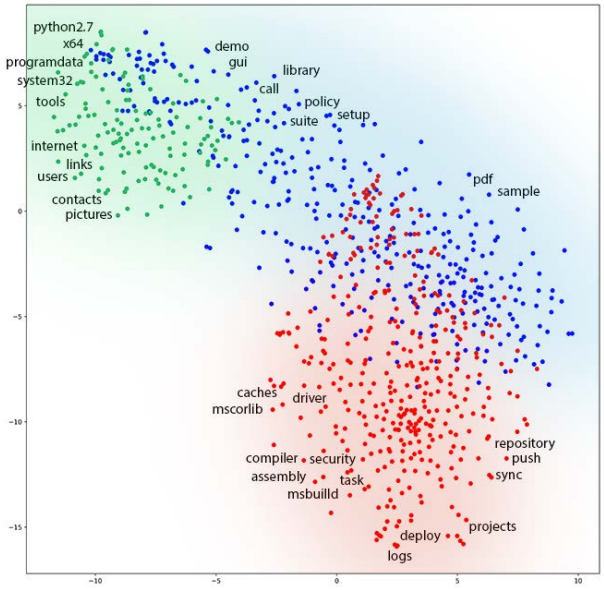

Figure 5: T-SNE visualization of file operations. This shows how the files used in file operations were clustered by the trained embedding layer. Blue shows files common to benign files, red shows files common to malicious, and green shows files common to both.

RQ2: Does the application of advanced NLP techniques improve the results of malware detection on dynamic analysis reports compared to other deep learning models?

RQ3: How robust are the various approaches to being applied to different datasets and sandbox/reports?

RQ4: Which parts of the report does Neurlux learn to use in detecting malware?

The first two research questions will help us to evaluate Neurlux's malware detection capabilities, and compare our approach to other deep learning models. The third question is useful in order to understand if our approach is learning robust features that apply to other environments. Finally, the last question is chosen to explore which parts of the report are used by Neurlux. This can help in determining if the approach is learning useful features.

6.1 Experiment Design

Here we will evaluate the performance of Neurlux and each of the comparison models described in the previous section. All the evaluated models are listed in Table 1 with a brief recap on their approach. We trained each model on reports from the *VendorSandbox* on executables from the *EmberDataset*. We chose the *EmberDataset* for our training because it is the larger dataset and would provide more samples for training. We performed the classical k-fold cross validation (where $k = 10$) to test the models. That is, we divide the dataset D into $D_1, D_2, ..., D_k$. We spare D_i for $1 < i < k$ for testing, and use the remaining $k - 1$ folds for training. This process is repeated k times to get accurate validation results. We trained all our models by minimizing the cross-entropy error.

After this, each model is then tested on the *VendorDataset* to evaluate its ability to generalize to a new dataset. Each model is also tested using the *CuckooSandbox* on the same samples in the *EmberDataset* to evaluate the robustness of the model to a different report from a different sandbox. These tests should show if the model is learning features that generalize well and are not specific to the particular report or dataset.

Additionally, we compared against *MalDy* [11]. This is a state of the art model using XG-Boost and feature hashing. We implemented it for our tests as we could not obtain the source code.

6.2 Results

Table 3 shows the results of each of the models we trained. Looking at the results, we see that Neurlux performs very well, showing the best accuracy when applied on both a new dataset and on a new sandbox, getting 87.0% and 86.7% respectively. The ensemble classifier outperformed it slightly in terms of validation accuracy, but it wasn't nearly as robust to differences in datasets or sandboxes. This result allows us to answer our first research question.

> **Answer for RQ1**: Neurlux, a deep learning method without feature engineering performs about as well in validation accuracy as our best model with feature engineering. It also showed better results than the feature engineered models when tested on another dataset or report format.

6.3 Individual Features

We also trained and tested each of the five individual features we extracted using the same CNN+BiLSTM+Attention model design that we use in Neurlux. These individual models are also used to compose the *Ensemble Model*. The results of each model are shown in Table 4. We observed that file actions perform the best of any of our features. It also generalized fairly, showing good results on the other dataset. Note that APIs were represented significantly differently between the two sandboxes, which explains its low score on *CuckooSandbox*.

We visualized the trained word embeddings for file actions to see if the word embeddings are creating good clusters. For this we use a T-SNE plot [17], which is shown in Figure 5. In the T-SNE we see clusters of similar files that the model learned. We also see that the two clusters for files seen in benign and malicious files only had a bit of overlap.

6.4 Attention

When we look at the results in Table 3, we find that the *Raw Model* had a lower validation accuracy than Neurlux, and performed much worse when tested on a different dataset. This implies that the *Raw Model* might not be using very general features, whereas Neurlux appears to be learning features that generalize better. In this section, we explore what the two models appear to be paying attention to, or in other words, which parts of the reports contribute most to the classification decisions by the models. This can be used to both understand the model better, and it can also be used by security researchers to identify possible features that they can use in other analyses.

Firstly, we examine the *Raw Model*. To do this, we use a concept called saliency, introduced by Simonyan et al. [37]. Saliency uses the

Model	Accuracy	Precision	Recall	F1-Score	Accuracy on *VendorDataset*	Accuracy on *CuckooSandbox*
MalDy [11]	.8923	.850	.830	.885	.55	.40
Counts Model	.935	.891	.941	.92	.778	–
Ensemble Model	.980	.986	.975	.980	.843	.780
Raw Model	.914	.948	.906	.927	.698	.627
Neurlux	.968	.967	.966	.968	.870	.867

Table 3: This table shows how the models performed. The models were all trained on reports from *VendorSandbox* on the *EmberDataset*. They are tested using K-Fold validation and tested on the other dataset and report format. The first row, *MalDy*, is a state of the art model from [11] that we compare with. The next two rows are feature-engineering based approaches.

Individual Feature	Accuracy	Accuracy on *VendorDataset*	Accuracy on *CuckooSandbox*
Registry	.944	.801	.767
File	.978	.842	.771
API	.865	.776	.510
Mutex	.808	.733	.677
DLLs	.960	.810	.701

Table 4: This table shows how the model performed on each of the five individual features. This table shows the validation accuracy as well as the accuracy when given a different dataset and different report format.

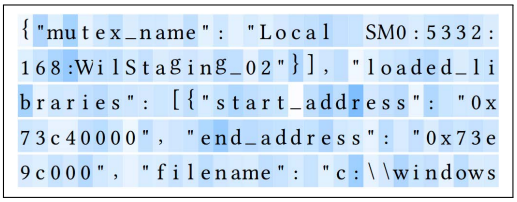

Figure 6: The most salient area of sample 234 highlighted by the *Raw Model* at the byte level showing the *Raw Model* giving a lot of importance to seemingly random parts such as part of "address" and "end". Sample 234 was misclassified by the *Raw Model* as benign when it is malicious.

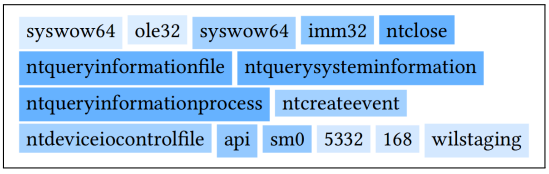

Figure 7: The highest attention areas of sample 234 highlighted by Neurlux. Neurlux paid the most attention to the api calls in this example. Neurlux was able to correctly classify it as malicious.

gradient of the output with respect to the input to determine which areas of input will most affect the output of a neural net. This will allow us to highlight, for a particular sample, the regions of bytes that contribute most to its classification. When applying saliency on the *Raw Model*, we see that the model frequently pays the most attention to hashes (SHA-1 hashes, MD5 hashes), DLLs, file names, and library addresses. An example of the most highlighted area of one sample is shown in Figure 6. Although SHA-1 hashes are frequently important to the model's classifications results, they are not a feature that generalizes well. Any byte changed can cause these features to be wrong. Additionally, intuitively, we know that library addresses are likely to change, and not the best feature either.

Then, we examine our approach, Neurlux, which uses NLP techniques to learn on whole words rather than bytes. For this model, we have an Attention layer, explained in the Section 2.5, which allows the model to learn what it should focus on, and allows us to interpret which words/phrases received the most attention directly. By examining the attention activations, we can also see what the model is paying attention. An example is shown in Figure 7, which shows our method giving the most attention to a few API calls. Overall, we found that Neurlux focused on parts of the document that seemed much more general. For example, in the first couple of samples it focused on words such as "ntqueryinformationprocess", "ntreadvirtualmemory", "virustotal", "programs", "startup". These intuitively make sense as valuable features. The first couple indicates that the process is attempting to interact with other running processes. Then, of course, "startup programs" shows that the executable might be trying to set a new process to autostart.

Our approach (Neurlux) was able to pick features that look better intuitively, and it showed that its results generalized well to other datasets. This seems to be a result of applying document classification techniques from NLP to our model. Our approach looks at whole words rather than bytes, and its model learns better which words and phrases are indicative of malicious behavior when compared with the raw model, which focused on more arbitrary things such as sequences of bytes in SHA1 or MD5 hashes.

We also counted which features show up most frequently with the highest attention score. More specifically, for a subset of malicious samples we took the words with the most attention and determined to which feature they applied. These results are shown in Table 5. We see that file operations were most commonly the most highly paid attention to, followed by API calls, then network and DLL loads. Interestingly, file operations were also the best performing individual feature. This means that the feature it paid the most attention to was also the best performing on its own.

Feature	Number of times in top attention
File	831
API	305
Network	107
DLL Loads	95
Mutex	72
Registry	20

Table 5: This table shows how many times each feature appeared as one of the 10 most important words according to the attention score for 100 samples.

Answer for RQ2: NLP techniques for document classification can be effectively applied on reports to perform malware detection and show much better results than our *Raw Model* neural network.

Answer for RQ4: Neurlux appears to be learning to use the best combinations of features. Specifically, it pays more attention to the file operations performed by the malware, as well as the API calls.

6.5 Robustness

We found that all the approaches had lower accuracies when tested on a dataset or sandbox report that they were not trained on. The report format has many differences that should account for the drop in accuracy. However, the lower accuracy on *VendorDataset* implies that we are learning features that do not generalize to all executables. This could be a problem due to deficiencies in the training set (not having as wide a breadth of samples as we need). Also, it could be that the model is still learning some specific features that do not generalize as well. Neurlux showed the best robustness to the report formats. Also, the other NLP-based deep learning approaches (the individual models and *Ensemble Model*) showed decent robustness, implying that the NLP techniques give us more general features than our raw bytes approach.

Answer for RQ3: Neurlux is the most robust of the models we tried, showing the highest accuracy on another dataset and on another report format. On the other hand, our raw bytes model was poor at classification across datasets and reports. This implies that the features learned using text classification approaches were more general.

6.6 Unseen Malware Family

Another experiment we performed was to remove one malware family, train on the remaining data, and then evaluate Neurlux's classification accuracy when tested on that family. We removed all samples that were identified as the family Trojan.Viz.Gen.1, using VirusTotal. During evaluation Neurlux still correctly classified all 346 samples of that family as malware.

6.7 Performance

We run our experiments on Nvidia Titan RTX and Xeon Gold 6252 processor. Our training process took 19.47 *milliseconds/sample* and detection process took 8.21 *milliseconds/sample*. The data-cleaning/preprocessing runs at a rate of 6.16 *milliseconds/sample*.

7 DISCUSSION AND FUTURE WORK

One limitation of Neurlux is that it performs classification based on behavior seen in dynamic analyses. This means that it is not as effective as a preventative measure. However, its results still are useful for identifying malware to generate signatures or catch that an infection has occurred. Also, Neurlux showed that it was able to detect a previously unseen family, indicating that it can be used even on malware that has not been analyzed before. This result could be further explored to understand the correlations between different malware families.

Also, Neurlux relies on accurate and broad training data. Future work would be to try and make it more resilient to the quality of the training data, as well as exploring different execution environments to discover if one provides better results for it. Other directions for future work include exploring different models to improve the results. For example, image recognition [16], has recently shown promising results. These models are based on ones that were known to perform well on image recognition tasks, and can also sometimes use transfer learning, using the training already done on images.

7.1 Adversarial Learning

Some recent works try to evade the detection of machine learning based malware classifiers by adversarial learning. Their experiments show that it is possible to generate adversarial samples based on a trained machine learning classifier. The core of adversarial sample crafting is to find a small perturbation to feature vectors X of the original malware sample to change the classification results F to benign. Formally, they compute the gradient of F with respect to X to estimate the direction in which a perturbation in X would maximally change $F's$ output. The earliest work of this topic came from Nguyen et al. [20] who found that a slight change in the image could trick the image classifier, and then it has been introduced into computer security in recent years to attack security systems that rely on machine learning models.

Robustness against adversarial attacks provided is an essential design characteristic. Our future work will include making our proposed model more robust against such attacks.

8 RELATED WORK

There has been much work on using machine learning and NLP models to classify malware, all bringing new strengths and trade-offs to the table. There were initially many machine learning models being used, such as Support Vector Machines (SVM), Decision Trees (DT), Random Forest, and K-Nearest Neighbor (KNN) amongst others. Recently, neural networks have been prevalent for detecting and classifying malware.

Saxe et al. [33] proposed a method to distinguish malware from benign software based on a neural network, deep learning approach. Their system uses four different types of complementary static features from benign and malicious binaries. These features are entropy histogram features, PE import features, string 2D histogram features, and PE metadata features. However, this work primarily focused on feature engineering and static analysis. Fan et al. [4]

created a sequence mining algorithm that discovers consecutive malicious patterns using All-Nearest-Neighbor classifier. They used feature engineering to extract instruction sequences from the PE files as the preliminary features. This approach used static features and required domain knowledge for feature extraction.

Zheng et al. [43] creates a behavior chain that aids in its detection method. The method monitors behavior points based on API calls and then uses the respective calling sequence at runtime to construct a behavior chain. The system uses long short-term memory (LSTM) to detect maliciousness from the created chains. Salehi et al. [31] used a monitored environment where the arguments and return values of every API call are recorded as (API, variables, return value) tri-tuples. The authors found a selective and discriminative set of these features and used SVM algorithm for classification. Other methods also focused on using API sequence calls for dynamic malware classification [23, 32, 35]. However, restricting to only one type of feature limits the vast feature space that can be used to represent different types of malware behaviors.

MalInsight [7] takes a different approach, and profiles malware based on basic structure, low-level behavior, and high-level behavior. A feature space is built on three core profiles, namely, the structural features, how the binary interacts with the OS, and operations on files, registry, and network. Han et al. pick select features from the Cuckoo sandbox and train on those. Unlike our experiments in Section 6.3, they do not use neural networks and they do not use the whole report as Neurlux does, instead requiring feature selection.

Another approach is based on early stage recognition [29]. They implemented a recurrent neural network that takes as input a short feature set of file activity. They can predict whether or not a file is malicious using the first few seconds of file execution. Their intuition is that malicious activity surfaces rapidly once the malicious file begins execution. Their approach differs from ours in that it only uses file activity, and aims to detect malware in real time. We use entire dynamic analysis reports which provide a more complete picture but requires analyzing it in a sandbox.

Zhong et al. [44] suggested that a single deep learning model was insufficient and created a Multi-Level Deep Learning System. They first partition the data using static and dynamic features, then create a convolutional model which learns to classify each cluster, and combine the clusters to create their final model. They use a feature extraction phase to extract static and dynamic features, whereas Neurlux simply learns on the report of dynamic behavior and not extracted features.

Another approach that performs classification on behavior reports is MalDy, which uses a bag of words approach, with models combined in an ensemble [11]. A limitation of a bag of words approach is that it does not take into account the context, just the frequencies with which words appear [22]. Additionally, BoW produces feature vectors which are fairly sparse. On the other hand, our system, which uses word embeddings does not have this limitation, and produces more dense, low dimensional feature vectors. In our evaluation (results in Table 3), we show that Neurlux gives a higher validation accuracy and a much stronger ability to generalize, both to other datasets and to other report formats than MalDy.

9 CONCLUSIONS

This paper introduced Neurlux, a robust malware detection tool that can successfully identify malicious files based on their run-time behavior without feature engineering. It is based on techniques borrowed from the field of document classification and applied to dynamic analysis reports. Based on our evaluation results, we conclude that Neurlux outperforms similar approaches for malware classification. Our work not only focused on eliminating the need for feature engineering, but also explains the relation of the classification process with respect to different auto-detected features. The fact that Neurlux can retain a high detection accuracy when tested on samples from another dataset and on an unknown report format shows that our model promises robust real-world applicability.

ACKNOWLEDGMENTS

This research is based on research sponsored by a gift from Intel for the investigation of machine learning for malware analysis, by the National Science Foundation grant #CNS-1704253, and by DARPA under agreement number #FA8750-19-C-0003. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

We would also like to thank Lastline for providing data that made this research possible.

REFERENCES

[1] Cuckoo, automated malware analysis. https://cuckoosandbox.org/.
[2] H. S. Anderson and P. Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.
[3] T. Brosch and M. Morgenstern. Runtime Packers: The Hidden Problem? *Black Hat USA*, 2006.
[4] Y. Fan, Y. Ye, and L. Chen. Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications*, 52:16–25, 2016.
[5] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin. Compatibility is not transparency: Vmm detection myths and realities. In *HotOS*, 2007.
[6] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer, 2017.
[7] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong. Malinsight: A systematic profiling based malware detection framework. *Journal of Network and Computer Applications*, 125:236–250, 2019.
[8] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.
[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[10] Kanchan Sarkar. Recent trends in natural language processing using deep learning, 2017. https://medium.com/@kanchansarkar/recent-trends-in-natural-language-processing-using-deep-learning-a1469fbd2ef.
[11] E. B. Karbab and M. Debbabi. Maldy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports. *Digital Investigation*, 28:S77–S87, 2019.
[12] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda. {UNVEIL}: A large-scale, automated approach to detecting ransomware. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 757–772, 2016.
[13] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *USENIX security symposium*, volume 4, pages 351–366, 2009.
[14] 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*, pages 137–149. Springer, 2016.
[15] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti. Detecting environment-sensitive malware. In *International Workshop on Recent Advances in Intrusion Detection*,

pages 338–357. Springer, 2011.
[16] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.
[17] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
[19] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE, 2007.
[20] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.
[21] D. Oktavianto and I. Muhardianto. *Cuckoo malware analysis*. Packt Publishing Ltd, 2013.
[22] G. Paltoglou and M. Thelwall. More than bag-of-words: Sentence-based document representation for sentiment analysis. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 546–552, 2013.
[23] N. Peiravian and X. Zhu. Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th international conference on tools with artificial intelligence*, pages 300–305. IEEE, 2013.
[24] R. Perdisci, A. Lanzi, and W. Lee. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 301–310. IEEE, 2008.
[25] E. Raff, J. Sylvester, and C. Nicholas. Learning the pe header, malware detection with minimal domain knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 121–132. ACM, 2017.
[26] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[27] T. Raffetseder, C. Kruegel, and E. Kirda. Detecting system emulators. In *International Conference on Information Security*, pages 1–18. Springer, 2007.
[28] B. Rahbarinia, M. Balduzzi, and R. Perdisci. Exploring the long tail of (malicious) software downloads. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 391–402. IEEE, 2017.
[29] M. Rhode, P. Burnap, and K. Jones. Early-stage malware prediction using recurrent neural networks. *computers & security*, 77:578–594, 2018.
[30] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *2012 IEEE Symposium on Security and Privacy*, pages 65–79. IEEE, 2012.
[31] Z. Salehi, A. Sami, and M. Ghiasi. Maar: Robust features to detect malicious activity based on api calls, their arguments and return values. *Engineering Applications of Artificial Intelligence*, 59:93–102, 2017.
[32] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze. Malware detection based on mining api calls. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 1020–1025. ACM, 2010.
[33] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*, pages 11–20. IEEE, 2015.
[34] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. *arXiv preprint arXiv:1609.03020*, 2016.
[35] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala. Malware detection using assembly and api call sequences. *Journal in computer virology*, 7 (2):107–119, 2011.
[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[37] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
[38] D. Ucci, L. Aniello, and R. Baldoni. Survey on the usage of machine learning techniques for malware analysis. *arXiv preprint arXiv:1710.08189*, 2017.
[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
[40] VirusTotal. Av comparative analyses. https://blog.virustotal.com/2012/08/av-comparative-analyses-marketing-and.html. (Accessed: 2019-3-31).
[41] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.
[42] L. Zeltser. How malware generates mutex names to evade detection. https://isc.sans.edu/diary/How+Malware+Generates+Mutex+Names+to+Evade+Detection/19429/. (Accessed: 2019-5-31).
[43] H. Zhang, W. Zhang, Z. Lv, A. K. Sangaiah, T. Huang, and N. Chilamkurti. Maldc: a depth detection method for malware based on behavior chains. *World Wide Web*, pages 1–20, 2019.
[44] W. Zhong and F. Gu. A multi-level deep learning system for malware detection. *Expert Systems with Applications*, 2019.
[45] C. Zhou, C. Sun, Z. Liu, and F. Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.