Malware detection on Windows Audit Logs using LSTMs

Markus Ring[a,*], Daniel Schlör[b], Sarah Wunderlich[a], Dieter Landes[a], Andreas Hotho[b]

[a]*Faculty of Electrical Engineering and Informatics, Coburg University of Applied Sciences, 96450 Coburg, Germany*
[b]*Data Mining and Information Retrieval Group, University of Würzburg, 97074 Würzburg, Germany*

Abstract

Malware is a constant threat and is continuously evolving. Security systems try to keep up with the constant change. One challenge that arises is the large amount of logs generated on an operating system and the need to clarify which information contributes to the detection of possible malware. This work aims at the detection of malware using neural networks based on Windows audit log events. Neural networks can only process continuous data, but Windows audit logs are sequential and textual data. To address these challenges, we extract features out of the audit log events and use LSTMs to capture sequential effects. We create different subsets of features and analyze the effects of additional information. Features describe for example the action-type of windows audit log events, process names or target files that are accessed. Textual features are represented either as one-hot encoding or embedding representation, for which we compare three different approaches for representation learning. Effects of different feature subsets and representations are evaluated on a publicly available data set. Results indicate that using additional information improves the performance of the LSTM-model. While different representations lead to sim-

*Corresponding author
Email addresses: markus.ring@hs-coburg.de (Markus Ring), daniel.schloer@informatik.uni-wuerzburg.de (Daniel Schlör), sarah.wunderlich@hs-coburg.de (Sarah Wunderlich), dieter.landes@hs-coburg.de (Dieter Landes), hotho@informatik.uni-wuerzburg.de (Andreas Hotho)

ilar classification results, analysis of the latent space shows differences more precisely where FastText seems to be the most promising representation.
Keywords: Malware, LSTM, embeddings, Windows audit logs

---

1. Introduction

The idea of classifying security-related data into normal and malicious using machine learning algorithms is followed by the community over years. Various surveys report about the recent state of using machine learning for cyber security [1], host-based intrusion detection [2], or on available data sources [3]. This work focuses on a specific task within that setting, namely the classification of windows audit log events into two classes, normal and malicious.

Problem. Audit logs are comprehensive and record detailed information about the operating system and user activities. All activities on Windows operating systems generate sequences of events in this kind of logging. That is why Windows audit log events are a promising source for detecting unwanted activities from malware, ransomware, trojan horses and so on. Further, it may even allow to distinguish between normal and malicious user behaviour. However, Windows audit logs are textual and sequential data which complicates their analysis using state-of-art machine learning algorithms and it is not clear whether it is beneficial to use all information available of Windows audit log events or whether certain information can be discarded. Therefore, manual analyses or static thresholds are often used to analyze Windows audit log events. This work tackles the problem of developing an appropriate approach for analyzing Windows audit log events using machine learning algorithms with respect to malware detection.

Objective. We aim to transform Windows audit logs to meaningful vectors such that they can be processed by typical machine learning algorithms like deep neural networks. Thereby, this work follows two main objectives. The first objective measures the impact of different representations for the used features. The second objective investigates the influence of considering different feature

subsets of Windows audit log events.

Approach and Contributions. The basic challenges, textual and sequential data, show parallels to natural language processing (NLP). One-hot encodings are often used to represent words in NLP. Other methods such as BERT [4], ELMo [5], GloVe [6] or Word2Vec [7, 8] use a text corpus as input and learn real-valued vector representations for words. Due to the similar structure of the data, we transfer one-hot encodings and three embedding approaches (FastText, GloVe and Word2Vec) to our security-related data. First, we extract several features from each event of Windows audit logs. Then, we use one-hot encoding and different embedding representations to transform these values to continuous vectors. Finally, this work explores Long Short Term Memory Networks (LSTMs) for classifying the preprocessed Windows audit log events into two classes (normal and malicious). The LSTMs are evaluated on different feature subsets and different representations. Thereby, this work analyzes if the presence of additional information helps to improve the classification accuracy. We experimentally evaluate our models on a publicly available data set from Berlin et al. [9]. The experiments indicate an improvement of the results when using additional information whereas different representations only have a marginal influence on the classification performance. The source code will be published on github.

Structure. The remainder of the paper is organized as follows. Section 2 discusses related work about machine learning algorithms for malware detection. The required foundations including Windows audit logs and encoding approaches are discussed in Section 3. Section 4 proposes our feature generation approach and neuronal network architecture. Experiments are presented in Section 5 and are discussed in Section 6. The last section summarizes the paper.

2. Related Work

This section reviews related work about malware detection using machine learning algorithms. A comprehensive survey about malware detection approaches using data mining techniques is given by Souri and Hosseini [10]. Souri and Hosseini categorize existing approaches into signature-based or behavior-based, and explicitly address the specific challenges for the application of data mining approaches. Ye et al. [11] give a brief overview of the malware and anti-malware industry. Then, the authors provide a detailed overview of data mining methods for malware detection and categorize them into classification-based and clustering-based approaches. Further, this survey discusses the topics features extraction and feature selection in great detail.

A broad range of research regarding host-based intrusion detection using sequences of system calls can be found. System calls are similar to Windows audit logs in that they also have a type and various parameters, but system calls contain even more detailed information as they record every access to the kernel. Over the years, Hidden Markov Models [12], Support Vector Machines [13] and Neural Networks [14] have been used to analyze system calls. Athiwaratkun and Stokes [15] execute normal and malware binaries and record the generated system calls. Then, the authors use a recurrent neural network to detect malware based on the recorded system calls. Wunderlich et al. [16] classify systems calls using an LSTM network and study the influence of different embedding representations for system calls.

Another popular approach is the analysis of binaries. Saxe and Berlin [17] extract features like byte and entropy histograms from benign and malicious binaries. Then, the authors use a deep neural network to classify binaries into two classes, normal and malicious. Moskovitch et al. [18] evaluate four different classifiers for malware detection based on binary files. In a preprocessing step, the authors parse the binary files and extract n-grams. Features are built using term frequency (TF) and inverse document frequency (TF-IDF) based on the extracted n-grams. In order to reduce the huge number of features, only the

50.000 most frequent features are considered. Another data source for identifying malware is used by Sami et al. [19]. The authors use classification methods such as Naïve Bayes or Random Forest to classify binaries based on their API calls. Further, Boukhtouta et al. [20] evaluate various machine learning algorithms for malware classification based on network data. These works also try to detect malware, but use other data sources like (binary) file content, API calls, network traffic or build their own datasets by collecting data from sources like VirusShare (e.g. [21] or [22]) which makes them difficult to compare. Often these data sets are not available or only in preprocessed formats, making it impossible to subsequently identify the used binary files and calculating different features.

The most similar work to ours is given by Berlin et al. [9]. Berlin et al. apply a logistic regression classifier to classify four minute time frames of Windows audit log events into the classes normal and malicious. The authors create features by counting the presence of specific action and target values within the Windows audit log events. Temporal aspects are considered through the creation of q-grams. This feature generation process results in about seven million features. Due to the large number of features, the correlation coefficient to the class label is used to extract the 50.000 most relevant features. The authors achieve a detection rate of 83 percent. In contrast to Berlin et al., we use LSTMs which capture temporal aspects inherently and do not model them manually through q-grams. In contrast to Berlin et al., we learn different representations for available features in Windows audit log events and use LSTMs which capture temporal aspects inherently and do not model them manually through q-grams.

Other works which use the Windows audit log data set from Berlin et al. [9] are presented by Wang et al. [23] and Guo et al. [24]. Although they use the same data set, they pursue a different objective than this work. E.g. Wang et al. propose a method for protecting deep neuronal networks against adversarial examples on the application domain malware detection. The method is based on randomly nullifying features and evaluated on the Windows audit log data

set.

3. Foundations

3.1. Windows Audit Logs

While a variety of data sets exist for the analysis of binaries (e.g. [25] and [26]), there are only few publicly available data sets for Windows systems. The only recent and labeled data set which contains Windows audit logs is made available by Berlin et al. [9], which we choose for our experiments. The structure of the data is shown in Figure 1.

```
{
  "start_time": "2014-09-10T20:22:10.000038223",
  "end_time": "2014-09-10T20:22:25.000038738",
  "size": 467,
  "Processes": [
    {
      "pid": 3548,
      "name": "[python]\\pythonw.exe",
      "events": [
        {
          "time": "2014-09-10T20:22:10.000038223",
          "event_id": 4663,
          "ignored": false,
          "string_id": "WINDOWS_FILE:Execute:[temp]\\971ab",
          "action": "Execute",
          "target": "[temp]\\971ab",
          "abstraction": ""
        },
        …
     ]}
     …
}
```

Figure 1: Excerpt of the Windows audit log files from Berlin et al. [9]

The data set consists of separate files where each file contains around four minutes of Windows audit log events. Each log file contains some meta information like the recording time and the number of events. A file has 1 to m processes. For each process, there are the process name *name*, process id *pid*

and the list of corresponding events. Events are the smallest units in the log files and contain the following features: *time*, *event_id*, *string_id*, *action*, *target* and *abstraction*. The feature *abstraction* is often not given and the *string_id* can be interpreted as a summary of the features *action* and *target*. Berlin et al. [9] used Cuckoo Sandbox to record the data set. Cuckoo Sandbox (cuckoo-box)[1] is an open source automated analysis system which among other things is able to analyze malicious files and trace their API calls. To avoid special characteristics of cuckoo-box, the authors introduced a feature *ignored*, which indicates if it is a special cuckoo-box event or not.

In reality, Windows audit log events are available as a continuous data stream which is not divided into separate files. Therefore, we use the division into files of the given data set only for labeling the data and evaluate each sequence of events separately. As a result, our work is more similar to a real-world scenario.

3.2. Representations

As mentioned in the introduction, there are several methods for learning embeddings. For this study we use FastText [27], GloVe [6] and Word2Vec [7] which have been successfully used in other security-related domains like IP addresses [28] and system calls [16] and evaluate their suitability for learning meaningful representations from Windows audit logs.

We either use one-hot vectors or word embeddings to transform the features *action*, *name* and *target* to continuous values. Both approaches are explained in the following.

3.2.1. One-hot vector

One-hot vectors consider the different values of a feature. Assume, the feature *action* has 10 different values: *close*, *create*, *delete*, *execute*, *modify*, *permissions*, *read*, *spawn*, *write* and *write_and_createReg*. In that case, the one-hot vector contains 10 components where each component represents a possible value of the feature *action*. The components are listed in a predefined

[1] https://cuckoosandbox.org/

order, e.g. the order of the listed values of the feature *action* given above. For one-hot vectors, the component which represents the current value is 1, while all other components are 0. Consequently, the value *close* is represented by the vector $\vec{a}_{close} = (1,0,0,0,0,0,0,0,0,0)^T$, the value *create* by the vector $\vec{a}_{create} = (0,1,0,0,0,0,0,0,0,0)^T$ and so on.

3.2.2. Word Embeddings

The three approaches (FastText, GloVe and Word2Vec) use a text corpus as input and create semantically meaningful vector representations for words. In that context, vector representations are called embeddings. The basic approaches and our adaptation for Windows audit logs are described in the following.

Word2vec. Methodically, Word2Vec [7] is based on a neural network where the hidden layer contains less neurons than the input and output layer. Using the skip-gram approach of Word2Vec, the neural network is trained with an input word and should predict the surrounding words. After the training, the weights of the hidden layer are used as vector representations for words. Words appearing in similar context tend to have similar vector representations while vector representations differ for words that do not appear in similar contexts.

FastText. FastText [27] builds upon the basic idea of Word2Vec but represents each word additionally as bag of character n-grams. The vector representation of a word is then the sum of the representation of all character n-grams. This approach allows sharing of sub-word representations between words, which improves the representations of infrequent words.

GloVe. GloVe [6] is a co-occurrence based approach, which uses local context window information like Word2Vec but also incorporates global information from matrix factorization to obtain meaningful word representations. In contrast to Word2Vec and FastText, GloVe is a counting-based rather than a prediction-based model. It benefits from training on non-zero entries in the

co-occurrence matrix to learn latent word representations in such a way that they follow the probability of co-occurrence of the respective words.

Adaptation. In the following, we exemplarily use the feature *action* and explain how we represent our data as word embeddings. The same procedure is done for the features *name* and *target*.

To be able to train the models, we need an analogy to sentences. Therefore, we consider the sequence of events as a sentence where the *action* represents the words. Each file of the data set represents one sentence consisting of all events in the order of occurrence. Consequently, we extract sentences like "*read execute spawn modify write*" from Windows audit logs. Table 1 illustrates the generation of training samples based on this sentence for Word2Vec.

Table 1: Sample generation for the parameter *action* for word2vec.

#							**input word**	**context word**
1	read	execute	spawn	modify	write	→	read	execute
							read	spawn
2	read	execute	spawn	modify	write	→	execute	read
							execute	spawn
							execute	modify
3	read	execute	spawn	modify	write	→	spawn	read
							spawn	execute
							spawn	modify
							spawn	write
4	read	execute	spawn	modify	write	→	modify	execute
							modify	spawn
							modify	write
5	read	execute	spawn	modify	write	→	write	spawn
							write	modify

At first, Word2Vec selects a so called *input word* from the training sentence. Then, words from the surrounding window (we refer to them as *context words*) are used to build training samples. Table 1 uses a window size of 2. Training

is done as follows. The neural network is fed with the *input word* and tries to predict a *context word*. For each training sample, the expected output value for the corresponding *context word* is 1 and 0 for all other words. Let's consider the first sample where *read* is the *input word* (see Table 1). In that case, we have the words *execute* and *spawn* in the context which should be predicted by the network for the input *read*. After the training phase, the weights of the hidden layer are used as vector representations for the values of the feature *action*.

FastText expands this approach by also incorporating sub-strings, i.e. *execute* is represented by the character n-grams <*ex, exe, xec, ecu, cut, ute, te*> and <*execute*>, with < and > representing word boundary tokens. The benefit of this approach becomes more visible for the features *name* and *target*, where sub-strings of for example the file name carry certain semantics, such as the prefix [system] for the Windows system root folder or a suffix such as .exe indicating an executable file, even if the full path name was unique and never seen during training.

Other than FastText and Word2Vec, GloVe formulates a weighted least-squares regression problem to infer word embeddings from co-occurrence counts. Therefore a co-occurrence matrix is constructed based on the extracted sentences from Windows audit logs, which indicates how often a certain word appears together with each other word. GloVe then uses the ratio of co-occurrence probabilities to optimize word vectors, such that their dot product follows the log-probability of their corresponding words.

For all approaches, the representations are created beforehand so they can be used as input values when analyzing the influence of different features with a LSTM-model. In our experiments, we use the FastText and Word2Vec implementations from the gensim 3.8.3 library [29]. For GloVe, we use the implementation of the glove-python-binary 0.2.0 library.

Table 2: Feature generation. Derived features are represented by → feature name.

level	**feature**	**raw**
1	start time	2014-09-10T20:22:10.38223
1	end time	2014-09-10T20:24:13.38733
1	size	451
2	process id	358
2	name	[python]\\python.exe
2	→ process path	[python]
2	→ process ending	exe
3	time	2014-09-10T20:22:10.38223
3	event id	4663
3	ignored	false
3	string id	WINDOWS_FILE:Execute: [system]\\cmd.exe
3	action	Execute
3	target	[system]\\cmd.exe
3	→ target path	[system]
3	→ target ending	exe
3	abstraction	""

4. Approach

4.1. Feature Generation

Table 2 provides an overview of the features and preprocessing rules. The column level indicates the hierarchy of the files. A file consists of 1 to m processes and each process contains 1 to n events. Level 1 features are the same for all events within the file, level 2 features are the same for all events which are caused by the same process and level 3 features are different for each event. The column feature describes the name of the feature and the column raw shows example values.

All features from level 1 are artificially created during data set creation and therefore not used in this setting. The feature *process id* is an identifier from the operating system and is also ignored in this work.

We apply the same preprocessing for the features *name* (level 2) and *target* (level 3). Some processes and targets are often used and should be given special attention. These frequent values often indicate typical normal activities or even malware properties. Therefore, we introduce threshold values (min_proc and min_tar) to distinguish between frequent and non-frequent values for these features. All values which do not exceed the predefined thresholds are treated as a default value *other*. The use of threshold values has two advantages. First, we can also handle previously unseen feature values by assigning them the value *other* if necessary. Second, we do not learn behavior for certain feature values that are not representative in our training data. However, the threshold values play an important role and their influence is investigated in the experiments. After identifying relevant values, we either create one-hot vectors or learn embedding representations for all frequent values including the default value *other*. The transformation approaches are illustrated for the feature *action* in Section 3.2.1 and 3.2.2.

The file type and file path may provide useful information. Therefore, we extract the file type *process ending* from the feature *name*. The file type is defined as the part of the value after the last dot (.). Further, we extract

the file path *process path* from the feature *name*. The file path is defined as the first part of the value until the first two backslashes (\\). We identify frequent and non-frequent values for *process ending* and *process path* using the threshold *min_proc*. Next, we either create one-hot vectors or learn embedding representations. The same procedure is applied to the features *target ending* and *target path* based on the feature *target* using threshold *min_tar*.

The first feature on level 3 is *time*. Generally, the time of events could be a good identifier for normal and malicious behavior. Modifying large numbers of files would be normal for daytime, but suspicious at night in a company network. However, we refrain from considering timestamps since the underlying data set does not reflect such characteristics.

Since there are only four different values for the feature *event id* in the underlying data set, we ignore this feature as well. We also ignore the features *string_id* and *abstraction*. The feature *abstraction* is a free text field which is often empty and the feature *string_id* is an aggregation of the parameters *action* and *target*.

The feature *ignored* is used as a selector. We ignore all events which contain the value *true* for the feature *ignored*, since these events constitute artifacts of the data set creation (see Section 3.1).

For the feature *action*, we either create one-hot representations or learn embedding representations as described in the previous section.

Overall, we intend to train our models with different amounts of available information. Therefore, we create five different feature subsets which are shown in Table 3. The first column in Table 3 assigns each feature subset a unique identifier. Subset I contains only the most basic feature *action*. Subset II contains the feature *action* and the event-related feature *target* whereas subset III contains the feature *action* and the process-related feature *name*. Subset IV contains three features: *action*, *target* and *name*. The last subset V contains all relevant features, namely *action*, *target*, *name*, *process path*, *process ending*, *target path* and *target ending*.

Table 3: Feature subsets generated from Table 2.

ID	included features
I	action
II	action, target
III	action, name
IV	action, target, name
V	action, taget, name, process path, process ending, target path, target ending

4.2. LSTM Model

Sak et al. [30] show that stacking LSTM-layers together improves the handling of long-term dependencies in sequences. Due to the long-term dependencies in Windows audit logs, we decide to stack several LSTM layers together. A preliminary study shows that stacking three LSTM layers together works very well for our data. The LSTM layers are followed by two fully connected layers, as shown in Figure 2.

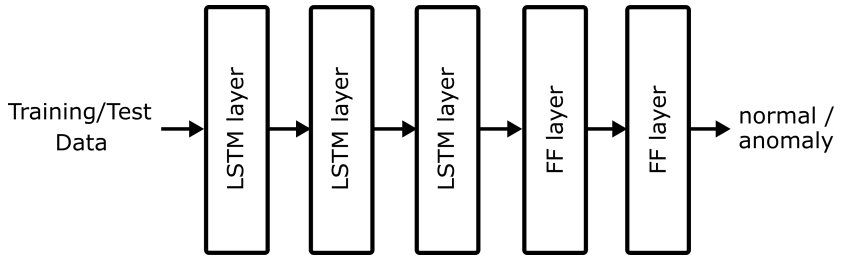

Figure 2: LSTM model.

The data transformation approach leads to a varying number of features which depend on the considered features and used representations. Therefore, we adapt the number of neurons per layer depending on the number of input features n as follows: The first layer uses $min(max(\frac{n}{2}, 4), 128)$ neurons, the second LSTM layer $min(max(\frac{n}{4}, 3), 64)$, the third LSTM layer $min(max(\frac{n}{8}, 2), 32)$, the first fully connected layer 4 and the last layer 1 neuron. For one-hot encoding, the number of input features increases fast for small threshold values min_tar and min_proc. Therefore, we limit the maximum number of neurons in the first

layer to 128, in second layer to 64 and in the third layer to 32 in order to avoid too large network architectures. The network is trained to predict a value of 0 for normal and 1 for malicious sequences. Training is done using the ADAM optimizer with a learning rate of 0.0001 for 10 epochs and the batch size is fixed to 64. The sequence length of Windows audit logs is fixed to 32 events. A preliminary study showed that a sequence length of 32 achieved better results than smaller sequence lengths like 8 or 16. The data set consists of files where each file contains the Windows audit log events of a four minute time window. We do not discard files with less than 32 events, instead we fill the stream with zeros at the beginning. The complete processing pipeline is shown as pseudo code in Algorithm 1.

Algorithm 1: Processing pipeline of Windows audit log events.

Input: dataset, feature_subset, representation_type, thresholds

Output: TPR, FPR, ACC

```
foreach subset in crossValidationSplits(dataset) do
    // Preprocess data
    train = getTrainingData(subset, feature_subset)
    replaceInfrequentValues(train, thresholds)
    rep = learnRepresentations(train, representation_type)
    // Transform training and test data
    train_preprocessed = transform(train, rep)
    test = getTestData(subset, feature_subset)
    test_preprocessed = transform(test, rep)
    // Train and evaluate the model
    trainModel(train_preprocessed)
    results = results ∪ evaluateModel(test_preprocessed)
TPR, FPR, ACC = mean(results)
```

5. Experiments

5.1. Evaluation Methodology

5.1.1. Data Set

We use the anonymized Windows audit log data set from Berlin et al. [9] which can be found on github[2]. The structure of the data set is discussed in Section 3. The available version consists of 5.440 files recorded in a real-world environment exclusively representing normal behavior as well as 20.362 files recorded in a cuckoo-box from which 5.683 files represent normal behavior and 14.679 files represent malicious behavior.

In our experimental evaluation, we use both parts of the data set and apply a 5-fold cross-validation over the complete data set. In that kind of evaluation, the data set is split into 5 parts where 4 parts are used to train the model and 1 part is used to evaluate the model. This procedure is repeated five times such that each part is used once for evaluation. Reported results are the mean and standard deviation over the 5-fold cross-validation.

5.1.2. Evaluation Methodology

In the area of intrusion detection, the number of detected attacks and false alarms are the most important evaluation measures. Therefore, we use the true positive rate (TPR), false positive rate (FPR) and classification accuracy (ACC) as evaluation measures. The TPR sets the number of correctly identified attack sequences in relation to all attack sequences within the data set. FPR calculates the ratio between the number of false alarms and the total number of non-attack sequences. ACC divides the number of correct predictions by all predictions.

Contrarily to the structure of the data set in which Windows audit logs are aggregated in separate files each representing time Windows up to 4 minutes, in a real world application Windows audit log events are a continuous data stream. In such a stream, subsequences represent malicious behavior, which an

[2]https://github.com/konstantinberlin/malware-windows-audit-log-detection

intrusion detection system ideally is able to differentiate from normal behavior. For our evaluation methodology, we follow this realistic setting and choose a subsequence-based classification. The data set, however, only reports labels for each file and, as a consequence, a sequence of audit log events labeled as malicious might contain malicious and non-malicious behavior. As more fine-grained labels are not available, some subsequences might be erroneously marked as malicious which may lead to a higher number of false alarms through our analysis methods.

5.1.3. Definition of a Baseline

The baseline approaches evaluate Windows audit log events only based on the feature *action* which reflects the feature subset I from Table 3. To improve comparability, the baselines are based on the same LSTM architecture (see Figure 2). All baselines use the feature *action* as it is event-related and events are the smallest units in Windows audit logs. Further, the feature *action* has a set of predefined values which is not the case for other event-related features like *target*. This allows the baselines to extract the essential information from Windows audit logs.

5.2. Experiment 1

The thresholds *min_proc* and *min_tar* are used to identify frequent and non-frequent values and thereby limit the number of possible manifestations for the features *process* and *target* in the data set. E.g., the threshold value 0.01% leads to the following amount of different values: 143 *names*, 22 *process paths*, 3 *process endings*, 465 *targets*, 22 *target paths* and 91 *target endings*. These numbers are calculated for one subset and may vary slightly over the different subsets of the 5-fold cross-validation.

Experiment 1 analyzes the influence of these threshold values. Since the thresholds affect the features *taget*, *name*, *process path*, *process ending*, *target path* and *target ending*, we choose feature subset V (see Table 3), since it contains all mentioned features. The smaller the values for the thresholds, the more

Table 4: Results of experiment 1.

Encoding	min_proc	min_tar	TPR	FPR	ACC
one-hot	3%	3%	0.9908 ± 0.0035	0.1634 ± 0.0086	0.8869 ± 0.0046
one-hot	1%	1%	0.9772 ± 0.0086	0.1527 ± 0.0087	0.8897 ± 0.0061
one-hot	0.5%	0.5%	0.9941 ± 0.0012	0.1349 ± 0.0102	0.9073 ± 0.0051
one-hot	0.1%	0.1%	***0.9944 ± 0.0013***	***0.1292 ± 0.0064***	***0.9112 ± 0.0031***
one-hot	0.01%	0.01%	0.9921 ± 0.0031	0.1361 ± 0.0049	0.9049 ± 0.0028
FastText	3%	3%	0.9814 ± 0.0079	0.1658 ± 0.0089	0.8823 ± 0.0076
FastText	1%	1%	0.9684 ± 0.0201	0.1557 ± 0.0102	0.8848 ± 0.0108
FastText	0.5%	0.5%	0.9913 ± 0.0034	0.1453 ± 0.0134	0.8993 ± 0.007
FastText	0.1%	0.1%	0.9928 ± 0.0012	0.1374 ± 0.0097	0.9052 ± 0.0053
FastText	0.01%	0.01%	0.9903 ± 0.0045	*0.1314 ± 0.0083*	*0.9084 ± 0.0054*
FastText	0.001%	0.001%	0.9911 ± 0.0042	0.1334 ± 0.0104	0.9073 ± 0.0068
FastText	0%	0%	*0.9930 ± 0.0033*	0.1398 ± 0.0113	0.9036 ± 0.0059
Word2Vec	3%	3%	0.9751 ± 0.0102	0.1707 ± 0.0102	0.8769 ± 0.0088
Word2Vec	1%	1%	0.9796 ± 0.0132	0.1556 ± 0.0129	0.8886 ± 0.0081
Word2Vec	0.5%	0.5%	***0.9935 ± 0.0017***	0.1438 ± 0.0118	0.9011 ± 0.0064
Word2Vec	0.1%	0.1%	0.9925 ± 0.0022	0.1354 ± 0.0097	0.9064 ± 0.0046
Word2Vec	0.01%	0.01%	0.9900 ± 0.0050	*0.1314 ± 0.0087*	*0.9082 ± 0.0060*
Word2Vec	0.001%	0.001%	0.9893 ± 0.0054	0.1314 ± 0.0097	0.9080 ± 0.0065
GloVe	3%	3%	0.9500 ± 0.0172	0.2156 ± 0.0216	0.8385 ± 0.0150
GloVe	1%	1%	0.9641 ± 0.0207	0.2125 ± 0.0203	0.8452 ± 0.0154
GloVe	0.5%	0.5%	0.9839 ± 0.0108	0.1768 ± 0.0202	0.8757 ± 0.0114
GloVe	0.1%	0.1%	0.9820 ± 0.0088	0.1626 ± 0.0211	0.8846 ± 0.0122
GloVe	0.01%	0.01%	*0.9925 ± 0.0036*	0.1591 ± 0.0232	0.8904 ± 0.0145
GloVe	0.001%	0.001%	0.9889 ± 0.0058	*0.1361 ± 0.0100*	*0.9047 ± 0.0068*

different values are considered.

Results. Table 4 show the results of experiment 1. The best results for each encoding are shown in italics, while the overall best results are in bold print. The columns *min_proc* and *min_tar* provide the used threshold values in percent. The results show no major differences for one-hot encodings and embedding-based representations (FastText, GloVe and Word2Vec).

One-hot encodings achieve their best results for medium-high thresholds like $min\_proc = 0.1\%$ and $min\_tar = 0.1\%$. Lower thresholds lead to slightly worse results. It should be mentioned that low thresholds lead to very large one-hot vectors which increase the computation time and lead to large neural networks

architectures compared to the embedding-based approaches.

In contrast to that, different threshold values have more effect on embedding representations than on one-hot encodings. The three chosen embedding variants achieve very similar results and perform similarly with regard to the used thresholds. In general, lower threshold values lead to better results in all evaluation measures. Due to its use of n-grams, FastText can also handle unknown values. Therefore, we also evaluate FastText without any threshold values ($min\_proc = 0\%$ and $min\_tar = 0\%$). However, results for FastText for thresholds between 0% and 0.1% differ only very slightly compared to the significantly higher computational demands without a threshold.

Explorative Analysis. In addition to the analysis of different embedding-techniques and thresholds by numbers, an in-depth review of the embeddings can reveal further insights which aspects are captured well within the latent space. We analyze the latent space exemplarily for the attribute *target* which by itself is the most meaningful single parameter. The embeddings are projected to a two-dimensional space using t-SNE [31]. To visualize the homogeneity of the latent space regarding specificity for attack / normal behavior, we apply the following coloring schema: For all files within the dataset, we count how often each *target* occurs in an attack respectively normal context. We then weight the counts according to the total proportion of attack vs. normal sequences. A *target* only occurring in attacks thereby has a score of 1.0 (red), whereas a *target* only present in normal behavior has a score of 0.0 (blue). A *target* which is present in both, attack and normal behavior equally (relative to the overall proportion) has a score of 0.5 (grey).

Figure 3 shows the representations for *target*, learned with a threshold of $min\_tar = 0.001\%$ for one-hot encoding, Word2Vec, GloVe and FastText embeddings. The latent space differs considerably between the representation variants. One-hot encodings do not yield a meaningful latent structure while GloVe and Word2Vec are able to provide some structures. In contrast to that, the ability of FastText to consider sub-strings seems to be beneficial to create semantically meaningful clusters, which reveal their preference for attack or normal behavior

even in a very low-dimensional space. When inspecting the different clusters closely, folder structure such as common sub-folders, file name structure and file endings become visible as well as semantic or contextual similarity such as temporary files of various types created as web-browser cache or system file clusters. This is also reflected in the attack score (color), where several very homogeneous attack and normal behavior clusters appear.

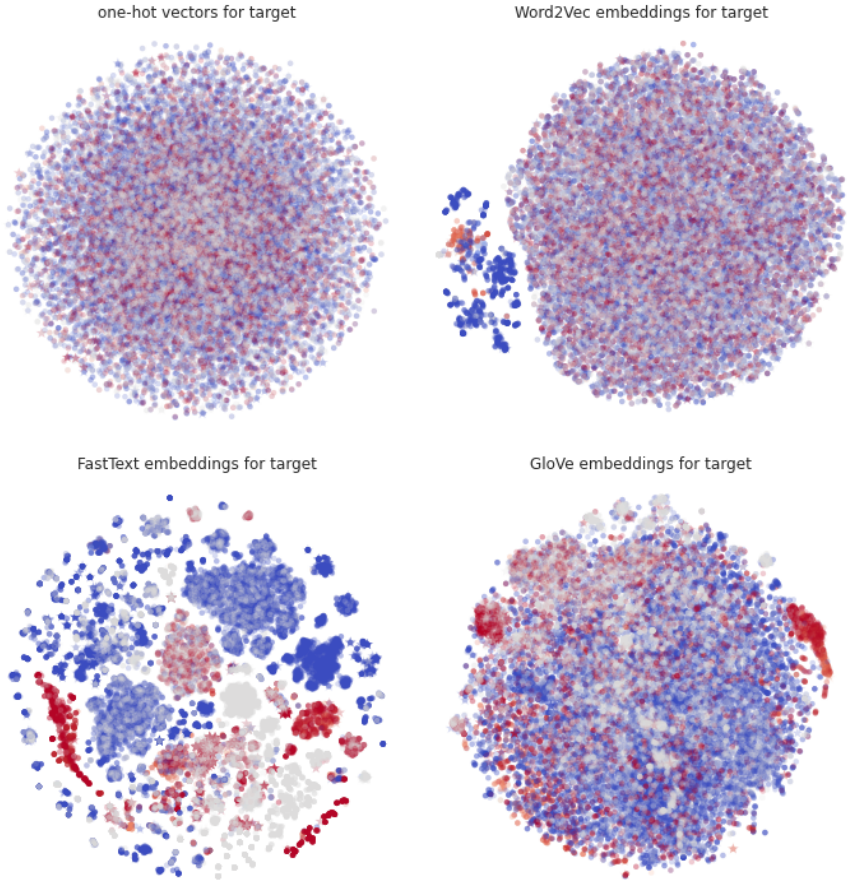

Figure 3: t-SNE visualization of the latent space for *target*

Table 5: Results of experiment 2.

Encoding	subset ID	min_proc	min_tar	TPR	FPR	ACC
one-hot	I	0.01%	0.01%	0.9075 ± 0.0124	0.2572 ± 0.0156	0.7948 ± 0.0081
one-hot	II	0.01%	0.01%	0.9519 ± 0.0160	0.1400 ± 0.0135	0.8890 ± 0.0059
one-hot	III	0.01%	0.01%	0.9791 ± 0.0217	0.1780 ± 0.0148	0.8716 ± 0.0106
one-hot	IV	0.01%	0.01%	0.9808 ± 0.0031	*0.1315 ± 0.0076*	0.9040 ± 0.0042
one-hot	V	0.01%	0.01%	***0.9921 ± 0.0031***	0.1361 ± 0.0049	*0.9049 ± 0.0028*
FastText	I	0.01%	0.01%	0.8972 ± 0.0224	0.2385 ± 0.013	0.8058 ± 0.0067
FastText	II	0.01%	0.01%	0.9384 ± 0.0056	0.1359 ± 0.0102	0.8884 ± 0.0067
FastText	III	0.01%	0.01%	0.9850 ± 0.0074	0.1863 ± 0.0116	0.8697 ± 0.0104
FastText	IV	0.01%	0.01%	0.9858 ± 0.0031	0.1434 ± 0.012	0.8988 ± 0.007
FastText	V	0.01%	0.01%	*0.9903 ± 0.0045*	***0.1314 ± 0.0083***	***0.9084 ± 0.0054***
Word2Vec	I	0.01%	0.01%	0.9086 ± 0.0298	0.2459 ± 0.0211	0.8046 ± 0.0077
Word2Vec	II	0.01%	0.01%	0.9401 ± 0.022	0.1515 ± 0.0104	0.8784 ± 0.0067
Word2Vec	III	0.01%	0.01%	0.9867 ± 0.0034	0.1882 ± 0.0168	0.8689 ± 0.0115
Word2Vec	IV	0.01%	0.01%	0.9859 ± 0.0064	0.1561 ± 0.0234	0.8903 ± 0.0146
Word2Vec	V	0.01%	0.01%	*0.9900 ± 0.005*	***0.1314 ± 0.0087***	*0.9082 ± 0.006*
GloVe	I	0.001%	0.001%	0.9543 ± 0.0493	0.3988 ± 0.1336	0.7165 ± 0.0753
GloVe	II	0.001%	0.001%	0.9684 ± 0.006	0.1515 ± 0.0073	0.8876 ± 0.0055
GloVe	III	0.001%	0.001%	0.9845 ± 0.0082	0.1954 ± 0.0090	0.8634 ± 0.0062
GloVe	IV	0.001%	0.001%	0.9856 ± 0.0083	0.1491 ± 0.0098	0.8949 ± 0.0056
GloVe	V	0.001%	0.001%	*0.9889 ± 0.0058*	*0.1361 ± 0.0100*	*0.9047 ± 0.0068*

5.3. Experiment 2

Experiment 2 analyzes the effects when using different representations and considering additional features of Windows audit log events. Therefore, we use several LSTM models which analyze Windows audit logs based on the different feature subsets of Table 3. For example, one-hot with subset ID III means that feature subset III is chosen in combination with one-hot representations. All feature subsets are evaluated for one-hot, FastText, GloVe and Word2Vec representations. Since the FPR rate is very important in the field of IT security, we have chosen similar thresholds for each representation which result in low FPR rates in experiment 1. Results of experiment 2 are shown in Table 5.

Results. Table 5 shows that one-hot as well as embedding-based representations (GloVe, FastText and Word2Vec) benefit from additional features, since these methods achieve better results for subsets which consider more features. As in experiment 1, different representations achieve very similar results.

5.4. Experiment 3

From the previous experiments, we observe the following: (1) None of the analyzed representations is clearly better or worse than the others. (2) Considering all available features (subset ID V) is beneficial and leads to better results. For those reasons, we select a suitable configuration (*encoding* = Word2Vec, *subset ID* = V, $min\_proc = 0.01\%$ and $min\_tar = 0.01\%$) and analyze whether we can improve the results further by optimizing parameters in experiment 3:

To do this, we first vary the parameters batch size, sequence length and learning rate. Then, we also vary the number of layers and neurons in the neural network. The default configuration of the model (as introduced in Section 4.2) is referenced under the term *normal*. We provide a configuration with only one LSTM-layer with $\frac{n}{2}$ neurons, followed by a dense layer with $\frac{n}{4}$ neurons and an output layer of one neuron. Thereby, n is the number of input features. This configuration is denominated *small*. Additionally, we provide a configuration with three LSTM-layers with n neurons each followed by two dense layers with 32 neurons each and an output layer with one neuron. We refer to this configuration as *large*.

Results of experiment 3 are shown in Table 6. The parameter study does not lead to any significant improvement. However, it can be observed that larger sequence lengths lead to lower false positive rates. In contrast to that, the parameters batch size, learning rate and the size of the neural network have only marginal influence.

6. Discussion

Experiment 1 analyzes the effect of different threshold parameters min_proc and min_tar. Generally, the threshold values can strongly influence the results. If the threshold values are too high, typical values for malware are not taken into account and the detection rate decreases. This effect can be observed in Table 4 for thresholds like $min\_proc = 3\%$ and $min\_tar = 3\%$, which causes

Table 6: Results of experiment 3.

Model	**Batch Size**	**Sequence Length**	**Learning Rate**	**TPR**	**FPR**	**ACC**
small	16	8	0.0001	*0.9936 ± 0.0013*	0.1479 ± 0.0037	0.8977 ± 0.0019
small	16	32	0.0001	0.9925 ± 0.0009	0.1304 ± 0.0097	0.9097 ± 0.0049
small	16	128	0.0001	0.9899 ± 0.003	*0.1256 ± 0.0052*	*0.9099 ± 0.0024*
small	64	8	0.0001	0.9922 ± 0.0012	0.1489 ± 0.0066	0.8965 ± 0.003
small	64	32	0.0001	0.9901 ± 0.0053	0.1346 ± 0.0097	0.9061 ± 0.0068
small	64	128	0.0001	0.9859 ± 0.0057	0.1287 ± 0.0057	0.9066 ± 0.0035
small	96	8	0.0001	0.9916 ± 0.0018	0.1506 ± 0.0049	0.8952 ± 0.0023
small	96	32	0.0001	0.9911 ± 0.0027	0.1369 ± 0.0093	0.9049 ± 0.0046
small	96	128	0.0001	0.9851 ± 0.005	0.1419 ± 0.0224	0.8972 ± 0.015
normal	16	8	0.0001	*0.9932 ± 0.0017*	0.1439 ± 0.0039	0.9003 ± 0.0019
normal	16	32	0.0001	0.9929 ± 0.0022	0.1313 ± 0.009	0.9093 ± 0.0045
normal	16	128	0.0001	0.9893 ± 0.0065	*0.125 ± 0.0059*	*0.9102 ± 0.0034*
normal	64	8	0.00005	0.9931 ± 0.0017	0.1494 ± 0.0043	0.8965 ± 0.0022
normal	64	8	0.0001	0.9911 ± 0.0022	0.1439 ± 0.0053	0.8996 ± 0.0016
normal	64	32	0.00005	0.9903 ± 0.0043	0.1349 ± 0.0118	0.906 ± 0.0076
normal	64	32	0.0001	0.9924 ± 0.0012	0.132 ± 0.0097	0.9086 ± 0.0052
normal	64	32	0.001	0.9895 ± 0.0051	0.1297 ± 0.0095	0.9092 ± 0.0066
normal	64	128	0.00005	0.9865 ± 0.0052	0.1281 ± 0.0054	0.9071 ± 0.0035
normal	64	128	0.0001	0.9831 ± 0.0071	0.127 ± 0.0036	0.9068 ± 0.0026
normal	64	128	0.001	0.9862 ± 0.0046	0.1274 ± 0.0113	0.9076 ± 0.0068
normal	96	8	0.0001	0.9928 ± 0.0024	0.1489 ± 0.0094	0.8968 ± 0.0039
normal	96	32	0.0001	0.9926 ± 0.0017	0.1335 ± 0.0107	0.9077 ± 0.0059
normal	96	128	0.0001	0.9856 ± 0.0053	0.1273 ± 0.0044	0.9074 ± 0.0028
large	16	8	0.0001	0.993 ± 0.0023	0.142 ± 0.0056	0.9015 ± 0.0022
large	16	32	0.0001	***0.9938 ± 0.0027***	0.1314 ± 0.0097	0.9095 ± 0.0048
large	16	128	0.0001	0.9916 ± 0.0029	0.1242 ± 0.0049	***0.9114 ± 0.0018***
large	64	8	0.0001	0.993 ± 0.0019	0.1442 ± 0.0051	0.9 ± 0.0019
large	64	32	0.0001	0.9927 ± 0.0035	0.1335 ± 0.009	0.9077 ± 0.0046
large	64	128	0.0001	0.9885 ± 0.0045	***0.1241 ± 0.0038***	0.9105 ± 0.0024
large	96	8	0.0001	0.9915 ± 0.0026	0.144 ± 0.006	0.8997 ± 0.002
large	96	32	0.0001	0.9934 ± 0.0024	0.132 ± 0.0093	0.909 ± 0.0052
large	96	128	0.0001	0.9895 ± 0.0037	0.1246 ± 0.0055	0.9105 ± 0.0024

typical observations for malware to be lost in the masses of normal behavior behind the *default* token.

Experiment 1 indicates that different thresholds have only marginal effects, given different embedding representations. In principle, better results can be obtained for low thresholds which is probably due to the fact that more detailed information is drawn from the data. For this extracted information, embeddings are learned that take into account similarities between them. Finally, the models benefit from this additional information. For one-hot encodings, too low thresholds have no positive influence. In that case, the model probably does not generalize well and computation time increases considerably with long one-hot vectors.

An in-depth inspection of the latent space of all representation variants showed that FastText is a suitable choice as representation technique due to the capability to introduce previously unseen attributes and the ability to yield meaningful information from the clustering of neighboring samples in latent space for an in-depth analysis of specific targets.

Overall, experiment 1 leads to the assumption that embedding representations benefit from more detailed information whereas one-hot representations benefit from medium-high thresholds. Despite similar classification results for different representations, the visualization of the latent space shows differences. In particular, FastText seems to be most promising.

Experiment 2 analyzes the effect of using different feature subsets. Considering only the *action* of Windows audit log events, all encoding variations lead to worse results (e.g., see results for one-hot or GloVe in Table 5). This fact may be explained by the limited information content available. When considering the additional feature *target* (subset ID II), all representations achieve better results. This effect is boosted by including additional features, see e.g. the results for different features subsets for GloVe in Table 5. Results are very similar for all representations, with the best results being achieved by considering all features (subset ID V). Overall, experiment 2 shows that LSTMs benefit from additional information when analyzing Windows audit log events.

The parameter study of experiment 3 did not lead to a significant change in performance. For optimizing the results, the use of a larger training data set is an obvious starting point, since the used data set contains Windows audit logs from only few Windows systems. Further, the file-based labelling of the data set may lead to the situation that attack labelled files may include audit log sequences at the beginning or end of the file which are exclusively normal. This could be an explanation why larger values for the parameter sequence length lead to lower FPR in experiment 3, since a larger sequence length reduces the probability of extracting attack-free sub-sequences from attack labelled files.

Berlin et al. [9] do malware detection on the same data set, but they process and evaluate the data in a very different way (specifically four minute time Windows and supervised feature selection). In contrast to that, our approach processes Windows audit logs as continuous data stream which is a more realistic scenario. Therefore, both methods are not comparable and we did not use the work of Berlin et al. [9] as further baseline. However, we want to mention that Berlin et al. report a TPR of 83% and FPR of 0.1% for their evaluation setting. In contrast to them, our approach achieves higher TPR and FPR results. We have the following conjecture for the different results. Berlin et al. [9] generate a huge number of features which allows them to create more specific signatures of existing malware patterns within the training data. For this reason, we assume that their approach is less capable of detecting new respectively unknown malware (lower TPR) but also generates fewer false alarms (lower FPR).

7. Summary

Malware is constantly evolving and its detection is still an important topic today. In this paper, we investigated the suitability of LSTMs for detecting malware based on Windows audit log events. Windows audit logs record detailed information about running processes of users and the operating system. Consequently, such data provide deep insights and can be a useful source for malware detection and other types of malicious behavior.

Primary challenge of this work was that Windows audit logs consist of textual and sequential data which have to be represented appropriately for an analysis using neuronal networks. In order to address these challenges, we extracted several features from Windows audit logs and evaluated four different representation approaches: one-hot encoding and learning FastText, GloVe as well as Word2Vec embeddings. Based on this setting, we focused on two research questions. The first one analyzes the effects of different representation approaches. The second one analyzes the effects of considering different amounts of available information. Our experimental study showed that additional information improves the performance of LSTMs. Further, the results indicate that none of the representations can clearly be considered superior to the other. As a result, our study recommends the inclusion of available information and the use of FastText, which showed the most meaningful latent space, has the ability to generalize to previously unseen values and performs well for malware detection on Windows audit logs using LSTMs.

The proposed models achieved very good detection rates, but are associated with too many false alarms for the use in real company networks. However, the aim of this work was to analyze the influence of additional features and different transformation approaches. In order to improve the performance of the models, especially lowering the number of false alarms, the use of larger training data sets is a promising starting point. In the future, we intend to expand the evaluation of the proposed approaches on upcoming data sets.

Acknowledgements

This work is funded by the Bavarian Ministry for Economic affairs through the OBELISK project under grant no. IUK 624/002. M.R. and S.W. are further supported by the BayWISS Consortium Digitization. S.W. is funded by the Bavarian State Ministry of Science and the Arts and coordinated by the Bavarian Research Institute for Digital Transformation (bidt). Further, we gratefully acknowledge the support of NVIDIA Corporation with the donation of the

Quadro GPU used for this research.

References

[1] A. L. Buczak, E. Guven, A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, IEEE Communications Surveys and Tutorials 18 (2) (2016) 1153–1176. doi:10.1109/COMST.2015.2494502.

[2] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, Q. Chen, A Survey of Intrusion Detection Systems Leveraging Host Data, ACM Computing Surveys 52 (6). doi:10.1145/3344382.

[3] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, A. Hotho, A Survey of Network-based Intrusion Detection Data Sets, Computers & Security 96 (2019) 147–167. doi:10.1016/j.cose.2019.06.005.

[4] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, in: Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL), Association for Computational Linguistics, 2019, pp. 4171–4186. doi:10.18653/v1/N19-1423.

[5] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep Contextualized Word Representations, in: Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), Association for Computational Linguistics, 2018, pp. 2227–2237. doi:10.18653/v1/N18-1202.

[6] J. Pennington, R. Socher, C. D. Manning, GloVe: Global Vectors for Word Representation, in: Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543. doi:10.3115/v1/D14-1162.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed Representations of Words and Phrases and their Compositionality, in: Advances in Neural Information Processing Systems (NIPS), 2013, pp. 3111–3119.

[8] Q. Le, T. Mikolov, Distributed Representations of Sentences and Documents, in: International Conference on Machine Learning, 2014, pp. 1188–1196.

[9] K. Berlin, D. Slater, J. Saxe, Malicious Behavior Detection using Windows Audit Logs, in: ACM Workshop on Artificial Intelligence and Security, 2015, pp. 35–44. doi:10.1145/2808769.2808773.

[10] A. Souri, R. Hosseini, A state-of-the-art survey of malware detection approaches using data mining techniques, Human-centric Computing and Information Sciences 8 (3). doi:10.1186/s13673-018-0125-x.

[11] Y. Ye, T. Li, D. Adjeroh, S. S. Iyengar, A Survey on Malware Detection Using Data Mining Techniques, ACM Comput. Surv. 50 (3) (2017) 1–40. doi:10.1145/3073559.

[12] E. Eskin, W. Lee, S. J. Stolfo, Modeling System Calls for Intrusion Detection with Dynamic Window Sizes, in: DARPA Information Survivability Conference & Exposition II (DISCEX), Vol. 1, IEEE, 2001, pp. 165–175. doi:10.1109/DISCEX.2001.932213.

[13] Y. Wang, J. Wong, A. Miner, Anomaly intrusion detection using one class SVM, in: IEEE SMC Information Assurance Workshop, IEEE, 2004, pp. 358–364. doi:10.1109/IAW.2004.1437839.

[14] A. Chawla, B. Lee, S. Fallon, P. Jacob, Host based Intrusion Detection System with Combined CNN/RNN Model, in: International Workshop on AI in Security, 2018, pp. 9–18.

[15] B. Athiwaratkun, J. W. Stokes, Malware classification with LSTM and GRU language models and a character-level CNN, in: IEEE International

Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 2482–2486. doi:10.1109/ICASSP.2017.7952603.

[16] S. Wunderlich, M. Ring, D. Landes, A. Hotho, Comparison of System Call Representations for Intrusion Detection, in: International Conference on Computational Intelligence in Security for Information Systems (CISIS), Springer, 2019, pp. 14–24.

[17] J. Saxe, K. Berlin, Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features, in: International Conference on Malicious and Unwanted Software (MALWARE), 2015, pp. 11–20. doi:10.1109/MALWARE.2015.7413680.

[18] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, Y. Elovici, Unknown Malcode Detection via Text Categorization and the Imbalance Problem, in: IEEE International Conference on Intelligence and Security Informatics, 2008, pp. 156–161. doi:10.1109/ISI.2008.4565046.

[19] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, A. Hamze, Malware Detection Based on Mining API Calls, in: ACM Symposium on Applied Computing, Association for Computing Machinery, 2010, p. 1020–1025. doi:10.1145/1774088.1774303.

[20] A. Boukhtouta, S. A. Mokhov, N.-E. Lakhdari, M. Debbabi, J. Paquet, Network malware classification comparison using DPI and flow packet headers, Journal of Computer Virology and Hacking Techniques 12 (2) (2016) 69–100. doi:10.1007/s11416-015-0247-x.

[21] Y. Fang, W. Zhang, B. Li, F. Jing, L. Zhang, Semi-Supervised Malware Clustering Based on the Weight of Bytecode and API, IEEE Access 8 (2020) 2313–2326. doi:10.1109/ACCESS.2019.2962198.

[22] S. Das, Y. Liu, W. Zhang, M. Chandramohan, Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Mal-

ware, IEEE Transactions on Information Forensics and Security 11 (2) (2016) 289–302. doi:10.1109/TIFS.2015.2491300.

[23] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia, X. Xing, X. Liu, C. L. Giles, Adversary Resistant Deep Neural Networks with an Application to Malware Detection, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 1145–1153. doi:10.1145/3097983.3098158.

[24] W. Guo, Q. Wang, K. Zhang, A. G. Ororbia, S. Huang, X. Liu, C. L. Giles, L. Lin, X. Xing, Defending Against Adversarial Samples Without Security through Obscurity, in: IEEE International Conference on Data Mining (ICDM), 2018, pp. 137–146. doi:10.1109/ICDM.2018.00029.

[25] [Online] Endgame Malware BEnchmark for Research, https://github.com/endgameinc/ember, Accessed: 2020-06-09.

[26] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, M. Ahmadi, Microsoft Malware Classification Challenge, arXiv preprint arXiv:1802.10135.

[27] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Transactions of the Association for Computational Linguistics 5 (2017) 135–146.

[28] M. Ring, D. Landes, A. Dallmann, A. Hotho, IP2Vec: Learning Similarities between IP Adresses, in: Workshop on Data Mining for Cyber Security (DMCS), International Conference on Data Mining Workshops (ICDMW), IEEE, 2017, pp. 657–666. doi:10.1109/ICDMW.2017.93.

[29] R. Řehůřek, P. Sojka, Software Framework for Topic Modelling with Large Corpora, in: LREC Workshop on New Challenges for NLP Frameworks, ELRA, 2010, pp. 45–50.

[30] H. Sak, A. W. Senior, F. Beaufays, Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling, in: In-

ternational Speech Communication Association (INTERSPEECH), 2014, pp. 338–342.

[31] L. v. d. Maaten, G. Hinton, Visualizing Data using t-SNE, Journal of Machine Learning Research (JMLR) 9 (2008) 2579–2605.