

Ransomware Detection with Deep Neural Networks

Matan Davidian^a, Natalia Vanetik^b and Michael Kiperberg^c

Shamoon College of Engineering, Beer Sheva, Israel

Keywords: Dynamic Analysis, Malware Detection, Neural Networks, Ransomware.

Abstract: The number of reported malware and their average identification time increases each year, thus increasing the mitigation cost. Static analysis techniques cannot reliably detect polymorphic and metamorphic malware, while dynamic analysis is more effective in detecting advanced malware, especially when the analysis is performed using machine-learning techniques. This paper presents a novel approach for the detection of ransomware, a particular type of malware. The approach uses word embeddings to represent system call features and deep neural networks such as Convolutional Neural Networks (CNN) and Long Short-Term Memory Networks (LSTM). The evaluation, performed on two datasets, shows that the described approach achieves a detection rate of over 99% for ransomware samples.

1 INTRODUCTION

The AV-TEST Institute¹ registered 1139 million malicious programs in 2020. The number of reported malware increases each year. According to IBM², the average time to identify a breach was 206 days in 2019—a 5% increase over the identification time in 2018. The costs associated with a breach increase with its identification time. Therefore, better identification techniques are required to shorten the identification time and lower the associated costs.


Malicious programs vary by their goal. Ransomware is a type of malicious software (malware), which when deployed on the computer encrypts or locks a computer or files, requesting that a ransom be paid to the author of the ransomware for the successful decryption and release of the user's data and system. Ransomware aims to compromise the availability, confidentiality, and integrity of the victim's data (Sharma and Sahay, 2016; Egunjobi et al., 2019).


The majority of techniques for detecting and classifying malware involve the use of static or dynamic features (Islam et al., 2013). The most basic approach for malware detection is checking program


samples against a predefined repository of patterns. Pattern comparison is usually performed as an exact match of executable file signatures generated, for instance, by the SHA-256 cryptographic hash function (Gilbert and Handschuh, 2003). In other cases, a pattern is described by a regular expression over instructions (Idika and Mathur, 2007). For this approach, patterns should be sufficiently general to describe slight malware variations but not too broad to capture benign programs.

Unfortunately, polymorphic and metamorphic malware mutate themselves upon each replication, thus allowing them to achieve a high degree of variance. These types of malware evade static analysis by obfuscating their semantics and altering their syntactic structure on each replication. However, malware's behavior remains intact between replications. This is the main motivation for using dynamic analysis for malware detection. Unlike static analysis, dynamic analysis is immune to evasions based on obfuscation, making it the preferred choice for analyzing unknown, zero-day malware.

The goal of behavioral analysis is to classify a program as malicious based on its behavior. Conceptually, the behavioral analysis system consists of two components: a process or a system monitor and an analyzer. The monitoring component can be implemented as a user-mode agent, a kernel module, a hypervisor, an emulator, or an external device. More privileged and stealthy monitoring components deliver more reliable descriptions of the system's behavior.

^a  <https://orcid.org/0000-0002-3676-0066>

^b  <https://orcid.org/0000-0002-4939-1415>

^c  <https://orcid.org/0000-0001-8906-5940>

¹ <https://www.av-test.org/en/statistics/malware/>

² [https://www.ibm.com/downloads/cas/ZBZLY7KL?](https://www.ibm.com/downloads/cas/ZBZLY7KL?_ga=2.148238199.1762516747.1577395260-1128561362.1577395260)

While the analyzer can be based on deterministic policies with certain guarantees regarding the classification outcomes, writing such policies requires a deep understanding of the overall system operation and the security risks. Therefore, machine learning techniques become more favorable for securing systems from known and unknown malware.

In this paper, we propose a method for the classification of system call sequences that utilizes Deep Neural Networks (DNNs). We divide all system calls made by a process into relatively short sequences, and then classify those sequences as belonging to either malicious (belonging to ransomware) or benign processes.

The main contribution of this paper is three-fold:

1. two new datasets for training machine learning models for malware detection called REY (**R**ansomware **E**s**Y**stem calls) and CryptoRansom.
2. a novel representation of string Win32 API attributes of a system call with *word vectors* (Mikolov et al., 2013) (also called *word embeddings*) that express how a word is used in the text and what it means; they are learned by considering the context in which the words appear;
3. a novel approach for Ransomware detection that uses Convolutional Neural Networks (CNN) (Kim, 2014), Long Short-Term Memory (LSTM) neural networks (Hochreiter and Schmidhuber, 1997) and their combinations. We compare our approach with baseline machine learning methods and state-of-the-art solutions and show that our method achieves both higher accuracy (the ratio of correctly classified samples) and higher sensitivity (the ratio of correctly classified samples for the positive, i.e., ransomware, class of samples) than the baselines.

This paper is organized as follows. Section 2 describes related work; Section 3 describes dataset construction, representation of data features, and neural models we employ for data classification. Section 4 is dedicated to experimental evaluation, and Section 5 concludes our work.

2 BACKGROUND AND RELATED WORK

Malware analysis systems can be classified as static or dynamic. Static analysis systems (Iwamoto and Wasaki, 2012; Fereidooni et al., 2016) extract information from the malicious file itself without executing it. This type of analysis is fast and secure because the malicious behavior is never invoked. Un-

fortunately, modern malware employs evasion techniques whose purpose is to conceal its malicious intent. In particular, polymorphism and metamorphism are widely used in modern malware (You and Yim, 2010), thus imposing limitations on the applicability of static analysis (Moser et al., 2007).

The dynamic analysis method (Or-Meir et al., 2019) attempts to solve the difficulties imposed by polymorphic and metamorphic malware. This method concentrates on the behavior of the malware rather than on its structure. The dynamic analysis system executes the potentially malicious program, observes its actions, and classifies them as malicious or benign.

Dynamic analysis systems vary by multiple aspects: some systems perform the analysis online during normal computer operation and notify the system administrator about any potential detections (Leon et al., 2021). Other dynamic analysis systems perform the analysis in an isolated environment (Jamalpur et al., 2018).

Another aspect by which dynamic analysis systems vary is the set of observed actions. Some systems acquire fine-grained information about malware's execution, like calls to system functions (system calls) (Jamalpur et al., 2018) or even private functions of the malware itself (Dash et al., 2016). Other systems collect general information, like CPU utilization or values of performance counters (Demme et al., 2013; Zhou et al., 2018; Bahador et al., 2014; Alam et al., 2020).

Finally, dynamic analysis systems differ in the analysis method of the acquired information. Some systems determine whether the acquired behavior is malicious or benign according to a set of pre-defined rules of malicious behavior (Amit et al., 2021). Other systems, and this work, use machine-learning techniques for the classification of malicious behavior.

The machine-learning techniques can operate on dynamic features (observed actions) (Or-Meir et al., 2019), static features (e.g., executable files) (Iwamoto and Wasaki, 2012), or a combination of the two (Zhou, 2018). Multiple Machine Learning (ML) designs have been used for malware detection and classification. Extensive surveys of these works can be found in (Sneha et al., 2020). Traditional techniques include Random Forests (RF), Decision Trees (DT), Support Vector Machines (SVM), k -nearest neighbors algorithm (kNN), Logistic Regression (LR), Multi-Layered Perceptron (MCP), and so on. Additionally, multiple techniques based on deep neural networks (NN) have been utilized for ransomware detection. We mention the most prominent papers in Table 1.

Our method belongs to the dynamic analysis cat-

Table 1: ML methods and feature representations in works malware detection and classification.

paper	methods
(Islam et al., 2013)	SVM, RF, DT
(Han et al., 2019)	rule-based, tf-idf
(Rhode et al., 2019)	NN, RF, SVM
(Zhang et al., 2019)	DT, RF, tf-idf
(Khan et al., 2020)	DNA sequences, feature selection
(Cusack et al., 2018)	RF
(Alhawi et al., 2018)	Bayes Network, MCP, RF, KNN, LR
(Scalas et al., 2019)	RF
(Chen et al., 2019)	TF-IDF, Linear Discriminant Analysis, Extremely Randomized Trees
(Pal et al., 2016)	RF
(Shaukat and Ribeiro, 2018)	LR, SVM, NN, RF
(Egunjobi et al., 2019)	Gradient Tree Boosting
(Vinayakumar et al., 2017)	Naive Bayes
(Rhode et al., 2018)	shallow and deep NN
(Homayoun et al., 2019)	RNN
(Agrawal et al., 2019)	LSTM, CNN
(Al-Hawawreh and Sitnikova, 2019)	LSTM with attention
(Arabo et al., 2020)	Convolutional Autoencoders (CAE), Variational Autoencoders (VAE)
	traditional ML and NN

egory as it uses information about system calls and neural ML classifiers for malware detection. We aim to use Win32 API attributes of system calls issued by processes to determine whether a subsequence of process system calls belongs to malicious ransomware.

3 PROPOSED METHOD

3.1 Dataset Construction

3.1.1 Data Collection

Our datasets consist of execution logs of benign programs and ransomware. Execution logs were collected using Windows Internals Process Monitor (Microsoft Corporation,) under Windows 7 SP1 running on a virtual machine. We chose this OS because it has been shown to have more vulnerabilities related to ransomware in comparison to Windows 10 OS (Zavarsky et al., 2016). The execution strategy was different for benign and malicious samples. Benign logs were collected by running the PCMark 8 benchmark (UL Benchmarks,) until completion. Specifically, we executed the “Storage” and the “Work” tests provided by PCMark.

We have extracted a different number of system calls for benign and ransomware samples. Our goal was to obtain a balanced dataset with a comparable number of system calls in both categories. Therefore, we extracted a different number of system calls from each sample to achieve this goal. We note that during “malicious runs” the system calls of benign processes

were also recorded. However, these system calls were excluded from the dataset.

We have recorded all Win32 API attributes that appeared in the runs, but only a subset of them was included in the datasets, as explained below. We included the following attributes: (1) **process identifier**, denoted by PID, which is a unique positive integer number, (2) system call name, which stands for an operation performed by the call, (3) **return value**, a string attribute representing either that the operation completed successfully or, if not, the error that occurred during its execution, (4) duration in seconds, computed from the start and end time of a process, (5) **system call arguments**. The remaining attributes were excluded from the dataset because they provide either irrelevant (e.g., parent process identifier, image path) or overly specific information (e.g., thread identifier, completion time) that cannot be used to train a generalizable ML model. A final dataset, therefore, contains a separate sequence of system calls for each process. Each sequence is identified by the run number and the process identifier.

3.1.2 REY (RansomwarE sYstem Calls) Dataset

The REY dataset includes a single ransomware sample — the Jigsaw ransomware. In addition, it contains 30 benign programs that are distributed with the Windows OS.

Two runs were made to build REY, as specified below.

1. The first run with the ‘jigsaw’ malware running in the background, that took 13 minutes. At that time 994,816 system calls were performed, 613,501 system calls of the malware. The system calls the malware only, from 150,000th to 236,505th, making it a total of 86,505 calls.
2. The second recording with only benign processes running in the background lasted for 10 minutes. 30 different processes were running. For every process, we added its first 5,000 system calls or less (if a process performed less than 5,000 calls) to the dataset.

The final size of the REY dataset is 173,010, with 86,505 malicious and 86,505 benign calls.

3.1.3 CryptoRansom Dataset

The CryptoRansom dataset contains recordings of 46 different benign processes and 12 different ransomware samples. The ransomware samples are a subset of the VirusShare (Corvus Forensics,) repository, a publicly available repository of malware samples. Each sample of the VirusShare repository was

submitted to VirusTotal (VirusTotal,) for classification by multiple antivirus vendors. If the classification string of at least 10 vendors contained the substring "ransom", we included that sample in our dataset. Additionally, we verified manually that these processes indeed encrypt files.

Malicious samples (ransomware) were executed for 15 minutes. After the execution period of each sample, the virtual machines were reset to their original state. The process that performed the highest number of system calls during this 15 minutes-period was considered malicious, and it was included in the dataset. Moreover, because malicious processes manifest their malicious behavior after they complete their initialization process, we extract from the recorded system call sequence the last 10,000 entries. We recorded every one of the ransomware sample runs for 20 minutes, and included the last 10,000 system calls from these runs into the dataset. To collect data for benign processes, we performed the following three runs.

1. 10 minute recording of processes running in the background. There were 26 different processes, and we collected the first 5,000 system calls for every process. Overall, 36,664 system calls were collected during this run.
2. PCMark8 benchmark complete Work tests (UL Benchmarks,). The Work tests are designed for testing typical office notebooks and desktop PCs that lack media capabilities and they contain the following workloads: Web Browsing, Writing, Video Chat, Spreadsheet.
There were 7 different processes (one for every test), and we collected the first 5,000 system calls (at most) for every process. Overall, 35,000 system calls were collected during this run.
3. PCMark8 benchmark complete Storage tests that contain the following workload traces – Adobe Photoshop light, Adobe Photoshop heavy, Adobe Illustrator, Adobe InDesign, Adobe After Effects, Microsoft Word, Microsoft Excel, Microsoft PowerPoint, World of Warcraft, Battlefield 3.
There were 10 different tests run on a single process, and we collected the first 5,000 system calls (at most), making it a total of 50,000 system calls.

As a result, the CryptoRansom dataset is balanced and includes the data for 46 different benign processes and 12 different ransomware samples, with 121,664 system calls of benign processes(50.34%) and 120,000 processes of ransomware processes (49.66%).

3.2 Data Representation

In this section, we describe the Win32 API attributes of system calls that we have used in our neural models and baselines, and representations we built for these features.

3.2.1 Feature Selection

In general, attributes for Win32 API functions can be divided into three categories by their domain - time attributes, string attributes, and numeric attributes. We collected all of the attributes produced by processes. We did not use time attributes in our data model, but used subsets of string and numeric attributes that do not identify the process.

3.2.2 Feature Representation

All string attributes were first split to separate words by using either delimiter (e.g., underscore, space, tab as in 'END OF FILE') or capital letters (as in 'Query-BasicInformationFile'). Additionally, some shortcuts were expanded into full words using hand-crafted rules – for example, 'RegQueryKey' was replaced with 'Registry Query Key'. Finally, all strings representing words were transformed into lower-case.

As a result, every string attribute a is associated with a sequence of English words (w_1^a, \dots, w_n^a) . For every word w_i^a of an attribute a we extracted its k -dimensional word vector \vec{w}_i^a (see (Mikolov et al., 2013)). We have used fastText vectors pre-trained on English webcrawl and Wikipedia of length $k = 300$ (see (Grave et al., 2018)). The final representation of an attribute a is an average of word vectors of its words:

$$a_{vec} = \text{avg}(\vec{w}_1^a, \dots, \vec{w}_n^a) \quad (1)$$

All numeric attribute values were max-min normalized and prefix zero-padded to vectors of length 300, namely

$$a_{norm} = (0, \dots, 0, a/a_{\max} - a_{\min}), \quad (2)$$

where a is the original numeric attribute value, and a_{\max} and a_{\min} are the maximal and the minimal values of this attribute in the dataset.

3.3 The Pipeline

The final pipeline of our approach includes the following steps: (1) data preprocessing, during which the parameters are collected for a process, as described in Section 3.2.1; (2) data representation where all features are represented by numeric vectors as described in Section 3.2.2; (3) randomly splitting the

Table 2: Data summary.

dataset	virus types	total syscalls	virus syscalls	benign syscalls	majority (%)
REY	1	173,010	86,505	86,505	50
CryptoRansom	12	241,664	120,000	121,664	50.34

process data into training and test sets; (4) generation of system call sequences for train and test set separately; (5) training and evaluation of a neural model. This approach is depicted in Figure 1.

3.4 Neural Classification Models

To represent a sequence of system calls of length W , we use the following neural models:

- 2-dimensional CNN, where every system calls a 300-dimensional representing either a numeric or a string attribute as described in Section 3.2.2. A system call sequence of length W is then represented by $N_{features} \times 300 \times W$ tensor $S_{N_{features} \times 300 \times W}$, where $N_{features} = 57$ is the number of system calls features. This model uses two convolutional layers with kernel size 3 and "relu" activation, followed by three decreasing fully connected layers.
- Many-to-one LSTM layer with the number of neurons set to $N_{features}$ with sigmoid activation.
- CNN-LSTM that combines the CNN and the LSTM; it is built from a two-dimensional CNN layer followed by a unidirectional LSTM layer.

For all the models loss function was defined to be binary crossentropy, and Adam optimization algorithm (Chang et al., 2018) was used.

4 EXPERIMENTS

4.1 Datasets

We evaluate our approach on two datasets – REY and CryptoRansom – whose construction is described in detail in Section 3.1. In all cases, the data were randomly shuffled and split to the 80% training set/20% test set. All the methods were trained on the training set and evaluated on the test set; evaluation results are reported below. A summary of dataset parameters is given in Table 2, together with the majority vote.

4.2 Setup and Metrics

The tests were performed on a server with Tesla K80(NVIDIA) GPU, Intel Xeon 2.3GHz CPU, 24 GB or RAM and 400GB SSD. Neural models were

implemented using Keras (Chollet et al., 2015) with the TensorFlow backend (Abadi et al., 2015), and we used sklearn (Pedregosa et al., 2011) implementations of RF and SVM algorithms. We report binary classification accuracy (measured as the ratio of correctly labeled samples to all samples), the runtime it too to train and test each system, and sensitivity (the proportion of true positives to all positives).

4.3 Evaluation Results and Analysis

We used RandomForest (Ho, 1995; Breiman, 2001) (RF) and Support Vector Machine (SVM) (Cortes and Vapnik, 1995), with linear and poly-kernel, as baseline models. We have also used a dense Fully-Connected Neural model (FCN) with 7 layers as a neural baseline. The neural models we compared to baselines are the CNN model, the LSTM model, and the stacked CNN-LSTM model, all described in Section 3.4. All neural models, including the baseline, ran for 10 epochs with batch size 12.

Table 3 shows evaluations results for baselines on both datasets. We report binary classification accuracy and sensitivity for the 'Yes' class, e.g., the ratio of correctly determined virus sequences to all virus sequences in the data. SVM with poly kernel on the CryptoRansom dataset was discovered to be too slow and it was stopped after 10 hrs of training; therefore, its results are not reported. Table 4 shows evaluations results for neural models that use CNN or LSTM or both; W denotes the length of system call sequences analyzed by a model. The aim of this experiment was also to determine what sequence length is optimal w.r.t. the accuracy and training and test times; best scores are marked in bold.

As can be seen from Table 4, classification accuracy is very high for CNN and CNN-LSTM models, and it is much lower for the LSTM model. Different values of W produce minor variation in accuracy scores of CNN and CNN-LSTM models, and we conducted tests to check if the difference in these results is statistically significant by applying Wilcoxon paired non-parametric two-tail test (Wilcoxon, 1992). It is worth noting that the Wilcoxon test applied to the CryptoRansom dataset showed no statistical significance in any of the above tests, and therefore we focused our efforts on the REY dataset.

Table 5 demonstrates results of prediction comparison for different values of W for both CNN and CNN-LSTM models on the REY dataset, and comparison of predictions of these models for the same W . We can conclude that the difference between predictions of CNN and CNN-LSTM for the same W is statistically significant in every case, while CNN model

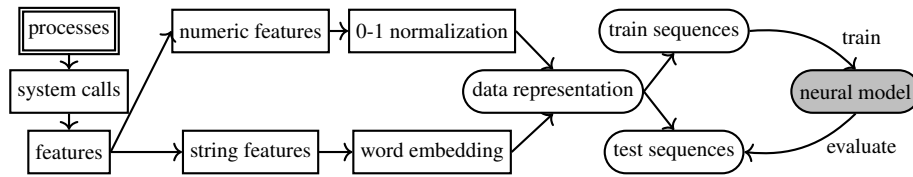


Figure 1: Pipeline of the ransomware detection with neural models.

Table 3: Baselines evaluation scores and runtime.

REY dataset				CryptoRansom dataset			
baseline	train time (m)	test time (m)	acc (%)	baseline	train time (m)	test time (m)	acc (%)
RF	36.7	0.02	87.85	RF	14.25	0.04	93.34
SVM (linear kernel)	7.5	0.003	83.28	SVM (linear kernel)	174.57	0.73	92.97
SVM (poly kernel)	59	11.2	52.05	SVM (poly kernel) ³	-	-	-
FCN	4.61	0.47	85.7	FCN	14.25	0.83	93.34

Table 4: Evaluation scores and runtime for CNN, LSTM and CNN-LSTM neural models; W is the length of a system calls sequence. All times are in minutes.

REY dataset					
model	W	acc (%)	sens (%)	train time	test time
CNN	3	96.67	93.71	4.15	0.028
CNN	6	99.51	99.02	6.88	0.567
CNN	9	99.41	99.23	10.75	0.789
CNN	12	99.74	99.51	18.83	1.191
CNN	15	99.82	99.65	36.13	1.516
LSTM	3	51.40	100	3.46	0.4
LSTM	6	95.23	97.02	7.01	0.539
LSTM	9	85.12	86.31	10.95	0.8
LSTM	12	79.76	82.55	18.83	1.842
LSTM	15	82.34	87.68	35.88	1.442
CNN-LSTM	3	95.79	93.81	4.54	0.031
CNN-LSTM	6	97.93	98.76	8.23	0.551
CNN-LSTM	9	99.62	99.23	12.36	0.812
CNN-LSTM	12	99.71	99.41	20.8	1.192
CNN-LSTM	15	99.74	99.48	36.73	1.054

CryptoRansom dataset					
model	W	acc (%)	sens (%)	train time	test time
CNN	3	95.31	97.59	2.27	0.13
CNN	6	99.27	99.29	5.72	0.7
CNN	9	99.59	99.77	9.92	2.02
CNN	12	99.53	99.95	20.1	2.51
CNN	15	99.70	99.70	39.1	2.28
LSTM	3	89.72	83.23	4.86	0.14
LSTM	6	89.73	85.47	10.96	0.71
LSTM	9	92.14	88.93	16.68	5.68
LSTM	12	48.77	48.83	24.39	2.47
LSTM	15	94.47	95.42	42.51	2.17
CNN-LSTM	3	97.32	96.74	6.14	0.38
CNN-LSTM	6	98.84	98.39	11.19	0.72
CNN-LSTM	9	99.60	99.65	17.02	1.32
CNN-LSTM	12	99.69	99.72	24.8	2.51
CNN-LSTM	15	99.74	99.82	72.15	2.35

predictions do not improve from $W = 9$ to $W = 15$. Because the highest accuracy is achieved by CNN with $W = 6$, we selected this model as the best one.

Table 5: Statistical significance tests on the REY dataset for CNN and CNN-LSTM models.

model	from W	to W	significant?	p-value
CNN-LSTM	3	6	yes	0.0057
CNN-LSTM	6	9	yes	<0.0001
CNN-LSTM	9	12	no	0.6694
CNN-LSTM	12	15	no	0.9115
CNN	3	6	yes	0.0124
CNN	6	9	yes	0.0056
CNN	9	12	no	0.3482
CNN	12	15	no	0.9557

model 1	model 2	W	significant?	p-value
CNN	CNN-LSTM	3	yes	<0.0001
CNN	CNN-LSTM	6	yes	<0.0001
CNN	CNN-LSTM	9	yes	<0.0001
CNN	CNN-LSTM	12	no	0.125
CNN	CNN-LSTM	15	yes	0.0146

5 CONCLUSIONS

In this paper, we present a method for ransomware detection that classifies system call sequences of a process as belonging to either malicious and or benign categories with very high accuracy. We represented processes as sequences of system calls and used NLP-based representation of system call features that are words or multi-word expressions. Then, we defined neural models that use CNN and LSTM neural layers, and process system call sequences of predefined length W . We evaluated our approach on two datasets: (1) REY dataset, and (2) CryptoRansom dataset, whose construction and annotation are described in Section 3.1.

From the evaluation, we can conclude that models using CNN as their first layers, such as pure CNN and CNN-LSTM, achieve the highest classification accuracy of over 99% on both datasets, while the pure LSTM model falls far behind. Moreover, statistical significance tests we conducted show that there is no

need to enlarge W beyond 6 and that the results of the CNN neural model are just as good as the results of the CNN-LSTM that contains a recurrent layer and is therefore slower. From a practical perspective, it means that short system calls sequences are enough to determine whether or not the process is malicious, and that this detection can be done using a fast pre-trained CNN model.

REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Agrawal, R., Stokes, J. W., Selvaraj, K., and Marinescu, M. (2019). Attention in recurrent neural networks for ransomware detection. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3222–3226. IEEE.
- Al-Hawawreh, M. and Sitnikova, E. (2019). Leveraging deep learning models for ransomware detection in the industrial internet of things environment. In *2019 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6. IEEE.
- Alam, M., Sinha, S., Bhattacharya, S., Dutta, S., Mukhopadhyay, D., and Chattopadhyay, A. (2020). Rapper: Ransomware prevention via performance counters. *arXiv preprint arXiv:2004.01712*.
- Alhawi, O. M., Baldwin, J., and Dehghantanha, A. (2018). Leveraging machine learning techniques for windows ransomware network traffic detection. In *Cyber threat intelligence*, pages 93–106. Springer.
- Amit, G., Yeshooron, A., Kiperberg, M., and Zaidenberg, N. J. (2021). Dlp-visor: A hypervisor-based data leakage prevention system. In *ICISSP*, pages 416–423.
- Arabo, A., Dijoux, R., Poulain, T., and Chevalier, G. (2020). Detecting ransomware using process behavior analysis. *Procedia Computer Science*, 168:289–296.
- Bahador, M. B., Abadi, M., and Tajoddin, A. (2014). Hpc-malhunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 703–708. IEEE.
- Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.
- Chang, Z., Zhang, Y., and Chen, W. (2018). Effective adam-optimized lstm neural network for electricity price forecasting. In *2018 IEEE 9th international conference on software engineering and service science (ICSESS)*, pages 245–248. IEEE.
- Chen, Q., Islam, S. R., Haswell, H., and Bridges, R. A. (2019). Automated ransomware behavior analysis: Pattern extraction and early detection. In *International Conference on Science of Cyber Security*, pages 199–214. Springer.
- Chollet, F. et al. (2015). Keras. <https://github.com/fchollet/keras>.
- Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.
- Corvus Forensics. Virusshare.com. <https://virusshare.com/>. Accessed: 2021-05-11.
- Cusack, G., Michel, O., and Keller, E. (2018). Machine learning-based detection of ransomware using sdn. In *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 1–6.
- Dash, S. K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., and Cavallaro, L. (2016). Droidscribe: Classifying android malware based on runtime behavior. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 252–261. IEEE.
- Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., and Stolfo, S. (2013). On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570.
- Egunjobi, S., Parkinson, S., and Crampton, A. (2019). Classifying ransomware using machine learning algorithms. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 45–52. Springer.
- Fereidooni, H., Conti, M., Yao, D., and Sperduti, A. (2016). Anastasia: Android malware detection using static analysis of applications. In *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE.
- Gilbert, H. and Handschuh, H. (2003). Security analysis of sha-256 and sisters. In *International workshop on selected areas in cryptography*, pages 175–193. Springer.
- Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Fasttext word vectors. <https://fasttext.cc/docs/en/crawl-vectors.html>.
- Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., and Mao, L. (2019). Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Computers & Security*, 83:208–233.
- Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R., Choo, K.-K. R., and Newton, D. E. (2019). Drthis: Deep ransomware threat hunting and intelligence system at the fog layer. *Future Generation Computer Systems*, 90:94–104.

- Idika, N. and Mathur, A. P. (2007). A survey of malware detection techniques. *Purdue University*, 48(2).
- Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, 36(2):646–656.
- Iwamoto, K. and Wasaki, K. (2012). Malware classification based on extracted api sequences using static analysis. In *Proceedings of the Asian Internet Engineering Conference*, pages 31–38.
- Jamalpur, S., Navya, Y. S., Raja, P., Tagore, G., and Rao, G. R. K. (2018). Dynamic malware analysis using cuckoo sandbox. In *2018 Second international conference on inventive communication and computational technologies (ICICCT)*, pages 1056–1060. IEEE.
- Khan, F., Ncube, C., Ramasamy, L. K., Kadry, S., and Nam, Y. (2020). A digital dna sequencing engine for ransomware detection using machine learning. *IEEE Access*, 8:119710–119719.
- Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.
- Leon, R. S., Kiperberg, M., Zabag, A. A. L., and Zaidenberg, N. J. (2021). Hypervisor-assisted dynamic malware analysis. *Cybersecurity*, 4(1):1–14.
- Microsoft Corporation. Process monitor. <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>. Accessed: 2021-05-11.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the Advances in neural information processing systems*, pages 3111–3119.
- Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE.
- Or-Meir, O., Nissim, N., Elovici, Y., and Rokach, L. (2019). Dynamic malware analysis in the modern era—a state of the art survey. *ACM Computing Surveys (CSUR)*, 52(5):1–48.
- Pal, A., Dasgupta, R., Saha, A., and Nandi, B. (2016). Human-like sensing for robotic remote inspection and analytics. *Wireless Personal Communications*, 88(1):23–38.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- Rhode, M., Burnap, P., and Jones, K. (2018). Early-stage malware prediction using recurrent neural networks. *computers & security*, 77:578–594.
- Rhode, M., Tuson, L., Burnap, P., and Jones, K. (2019). Lab to soc: Robust features for dynamic malware detection. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks—Industry Track*, pages 13–16. IEEE.
- Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C. A., Martinelli, F., and Giacinto, G. (2019). On the effectiveness of system api-related information for android ransomware detection. *Computers & Security*, 86:168–182.
- Sharma, A. and Sahay, S. K. (2016). An effective approach for classification of advanced malware with high accuracy. *arXiv preprint arXiv:1606.06897*.
- Shaukat, S. K. and Ribeiro, V. J. (2018). Ransomwall: A layered defense system against cryptographic ransomware attacks using machine learning. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, pages 356–363. IEEE.
- Sneha, M., Arya, A., and Agarwal, P. (2020). Ransomware detection techniques in the dawn of artificial intelligence: A survey. In *2020 The 9th International Conference on Networks, Communication and Computing*, pages 26–33.
- UL Benchmarks. Benchmarks pemark 10. <https://benchmarks.ul.com/pcmark10>. Accessed: 2021-05-11.
- Vinayakumar, R., Soman, K., Velan, K. S., and Ganorkar, S. (2017). Evaluating shallow and deep networks for ransomware detection and classification. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 259–265. IEEE.
- VirusTotal. Virustotal. <https://virustotal.com>. Accessed: 2021-05-11.
- Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer.
- You, I. and Yim, K. (2010). Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE.
- Zavarsky, P., Lindskog, D., et al. (2016). Experimental analysis of ransomware on windows and android platforms: Evolution and characterization. *Procedia Computer Science*, 94:465–472.
- Zhang, H., Xiao, X., Mercaldo, F., Ni, S., Martinelli, F., and Sangaiah, A. K. (2019). Classification of ransomware families with machine learning based on n-gram of opcodes. *Future Generation Computer Systems*, 90:211–221.
- Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., and Joshi, A. (2018). Hardware performance counters can detect malware: Myth or fact? In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 457–468.
- Zhou, H. (2018). Malware detection with neural network using combined features. In *China cyber security annual conference*, pages 96–106. Springer.