CONVOLUTIONAL NEURAL NETWORK FOR MALWARE CLASSIFICATION BASED ON API CALL SEQUENCE

Matthew Schofield[1], Gulsum Alicioglu[2], Russell Binaco[1], Paul Turner[1], Cameron Thatcher[1], Alex Lam1 and Bo Sun[1]

[1]Department of Computer Science, Rowan University,
Glassboro, New Jersey, USA
[2]Department of Electrical and Computer Engineering,
Rowan University, Glassboro, New Jersey, USA

ABSTRACT

Malicious software is constantly being developed and improved, so detection and classification of malicious applications is an ever-evolving problem. Since traditional malware detection techniques fail to detect new or unknown malware, machine learning algorithms have been used to overcome this disadvantage. We present a Convolutional Neural Network (CNN) for malware type classification based on the Windows system API (Application Program Interface) calls. This research uses a database of 5385 instances of API call streams labeled with eight types of malware of the source malicious application. We use a 1-Dimensional CNN by mapping API call streams as categorical and term frequency-inverse document frequency (TF-IDF) vectors respectively. We achieved accuracy scores of 98.17% using TF-IDF vector and 95.40% via categorical vector. The proposed 1-D CNN outperformed other traditional classification techniques with overall accuracy score of 91.0%.

KEYWORDS

Convolutional Neural Network, Malware Classification, Windows API Calls, Term Frequency-Inverse Document Frequency Vectors

1. INTRODUCTION

There are many different forms of malicious applications present in the highly connected software environment of the current technological world. Malicious applications such as viruses are constantly being developed and distributed in an attempt to extract information from computers or networks, and software such as firewalls and antivirus programs are constantly evolving to attempt to protect benign users and software from this threat [1]. Malicious software is constantly being developed and improved, so the classification of malicious applications is an ever-evolving problem. Signature-based, behavior-based and specification-based techniques are commonly used to detect malware [1-3]. Signature-based techniques detect malware fast and require less computational resources. However, it cannot detect new or unknown malware. Behavior-based detection method has ability to detect known and unknown malware, but it requires high computational resources [3]. To overcome these disadvantages, a specification-based technique, which is basically a behavior-based approach, is developed. Researchers have employed data mining and machine learning techniques and obtained good performance in malware detection and classification with high accuracy scores [4-6]. These methods provide

David C. Wyld et al. (Eds): AIAP, SIGML, CNSA, NIAI - 2021
pp. 85-98, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.110106

reliable and accurate results, especially for classifying metamorphic malware. Metamorphic malware indicates itself with different sequences in various environments, but it must demonstrate the same behavioral features in all environments. Hence, most of the methods used behavioral features for malware classification and detection rather than structural features [7-8]. API call sequence can provide considerable information about the behavioral features of malware. Most of the researchers conducted their study by using API calls to analyze behavior-based malware [9-12]. One benefit of having the ability to classify the type of malware from a malicious program's system call behavior is to more quickly attribute a source and better understand the effects of a piece of malware. Understanding the class of malware to which a malicious program belongs gives administrators of an infected a device insight on resolution strategies regarding the attack.

In this paper, we proposed two different approaches to classify the type of malware of a malicious program based on its produced API call stream [1-3]. The study presented both binary and multiclass classification problems. We used a public Windows API call dataset [8] with 8-class malware for the experiments. We used a 1-D Convolutional Neural Network by converting API call sequences to categorical vectors. We compared our results with existing methods. We also presented a text-based analysis for classification by using Random Forest Classifier, Bernoulli Naïve Bayes (NB), Multinomial NB, Gaussian NB, Decision Tree Classifier, AdaBoost Classifier, Bagging Classifier, KNeighbors Classifier, and MLP Classifier. We also presented overall comparisons for both approaches.

Another goal of this study was to investigate the utility of visual analytic approaches in data analysis, exploration and computational model development. The purpose of visual analytics is to form a union between computationally complex algorithms and human intuition [13]. We used visual approaches to understand the patterns of API calls by overall and malware type. This study is organized as follows. Section 2 provides a literature review. Section 3 describes the malware dataset, comprehensively. Section 4 provides a detailed methodology and visual approaches for data exploration. Section 5 presents the results of the study. Finally, Section 6 concludes the paper along with the discussion.

2. BACKGROUND

Since the volume of malware being spread has had rapid growth, many studies have been carried out to analyze and detect malware automatically [3,14-17]. There are three major approaches for malware detection based on the analysis types: Static (code analysis), dynamic (behavioral analysis) and hybrid analysis which uses both static and dynamic attributes of malware. The most common and traditional way to detect malware is for a system to maintain a hash signature-based blacklist of known malware. These signature-based methods fail to detect new, unknown, or obfuscated malware. Data mining and machine learning approaches have overcome disadvantages of signature-based methods by detecting new and unknown malware with high accuracy and detection rate [5, 17]. Hence, most of the studies used various machine learning algorithms to detect malware accurately and quickly [4, 14-16]. In recent studies, API call sequences are mostly used as a feature to detect malware. Xiaofeng et al. [15] used a combined deep learning and machine learning model for malware behavior analysis with binary classification (i.e., benign and malicious). Random Forest (RF) was used to extract API calls features. The combined RF and Long Short-Term Memory (LSTM) model classified malware with 96.7% accuracy. Malware analyses are carried out by using static or dynamic analysis. Researchers have studied to improve the usage of static and dynamic analysis. Han et al. [14] used RF, Decision Tree (DT), k-nearest neighbor, and XGboost methods to detect and classify multiclass malware by combining static and dynamic API calls sequences. The study explains the differences and relation between API sequences and explains malicious behavior types via

MalDAE framework. The study achieved a 97.8% detection rate and 94.4% classification accuracy with RF. The study also examined the effect of sequence length on measuring time. Salehi et al. [17] proposed a novel dynamic malware feature selection method based on generating new features to overcome disadvantages of using only static or dynamic analysis. RF, DT, sequential minimal optimization, and Bayesian Logistic Regression (BLR) methods were used, and 98.1% accuracy rate is obtained with BLR by generating three feature sets. As the volume of cyber-attacks with advanced malware increases, it makes malware hard for detection and classification. Bahtiyar et al. [5] proposed a multidimensional machine learning approach to detect advanced malware. They applied various regression models by using five distinguished features: stealthiness, Stuxnet closeness, behavioral instability, conventional malware arsenal and metamorphic engine. The proposed method provided 0.0001 mean squared error rate. Belaoued and Mazouzi [18] presented a fast-portable executable system to detect malware by using an efficient feature selection method. They used decision tree, boosted decision tree, AdaBoost, Random forest, and rotation forest algorithms for binary classification. They evaluated the proposed system with different subsets of data and achieved more than 98% accuracy in a short detection time (0.09 seconds). Gupta et al. [19] conducted a comprehensive study to capture malware behaviors based on API calls sequences. The experiments were conducted with five malware classes for 2000 samples. They encoded 534 important API calls to 26 categories (A... Z). N-gram analysis was carried out to extract class specific patterns. They also calculated fuzzy hashed scores with ssdeep algorithm to use as classification criteria. Yazi et al. [7] generated a dataset contains 8 different malware type with their Windows API call sequences and utilized LSTM classification models for 8 different types of malware. The highest accuracy rate was obtained with 97.5% in the adware class. Zhang et al. [20] proposed a feature-hybrid malware variants detection method by integrating various features. They extracted bi-gram model and encoded API calls as a frequency vector. The study achieved 90% classification accuracy by utilizing Convolutional Neural Network (CNN).

In the light of the literature review, we conducted a comprehensive study by using Windows system API calls [8] dataset to classify malware. API call sequences are used as a feature for malware classification. In the literature, most of the studies are carried out for binary classification to detect malware by utilizing one-vs-rest strategy [7, 15,18-19]. However, different types of malware require different precautions and treatments. For this reason, it is crucial to classify and determine the type of a given malware sample. We conducted experiments on both binary and multiclass to detect malware by using various decision trees (DT, RF, AdaBoost DT and Bagging DT), Naïve Bayes, k-NN, and Multilayer Perceptron algorithms and a 1D (CNN) algorithm. We implemented two different data representation approaches: text-based analysis through TF-IDF vectors and binary categorical vector.

3. WINDOWS MALWARE API CALL DATASET

The Windows Malware API Call Dataset is a public malware dataset containing 7,107 samples of malicious files among eight classes of malware, found in [8]. The breakdown of sample size per label is available in Table 1.

Table 1. API call dataset sample counts

Malware Type	**Sample Size**
Spyware	832
Downloader	1001
Trojan	1001
Worms	1001
Adware	379
Dropper	891
Virus	1001
Backdoor	1001
Total	**7107**

The dataset contains a class label and an arbitrary length list of API call strings for each data entry. For this dataset, API calls represent a system call on the Windows operating system occurring during the runtime of the malicious file. The dataset does not include benign samples.

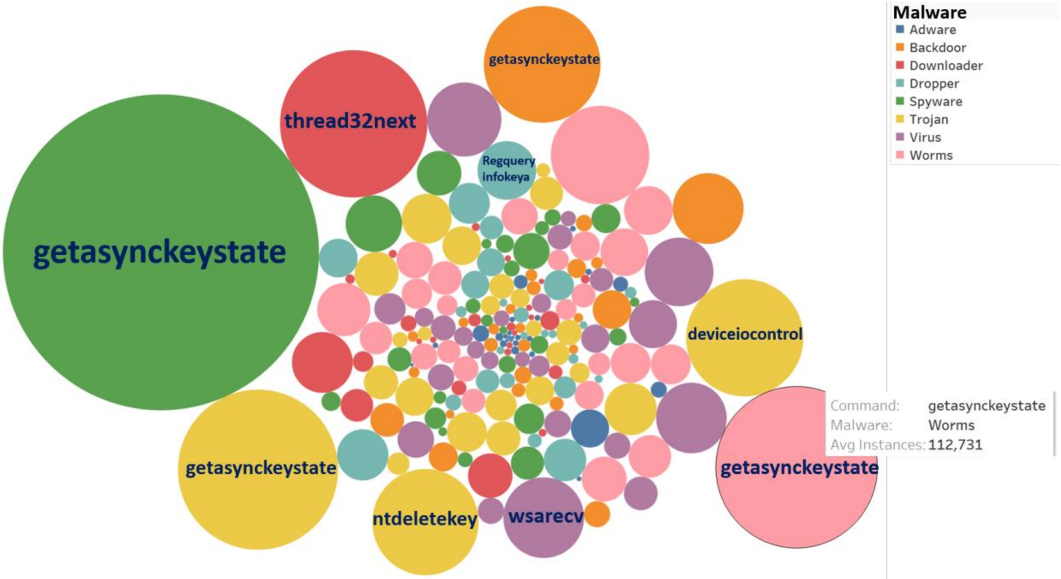


Figure 1. Bubble plot of frequent API calls by class

Exploratory data visualizations are presented to provide some human-readable insight to the breakdown of data features and metrics for Windows API call dataset by using Tableau [21]. Figure 1 shows a bubble plot of the most frequent API calls per class. The labels in the bubbles represent the API calls. The size of each bubble indicates the average instances that contain the related API call. Get AsyncKeyState is the most frequent call for Spyware, Worms, Trojan and Backdoor malware classes, and thread32next is the most frequent call for the Downloader class. Similar bar charts were constructed for each class with their record percentage, as seen in Figure 2. However, since these most frequent API calls are often used in many of the mentioned classes, statistical analysis may not be sufficient for identifying what class a call stream belongs to. This justifies the need for a more complex system than an expert rule set.

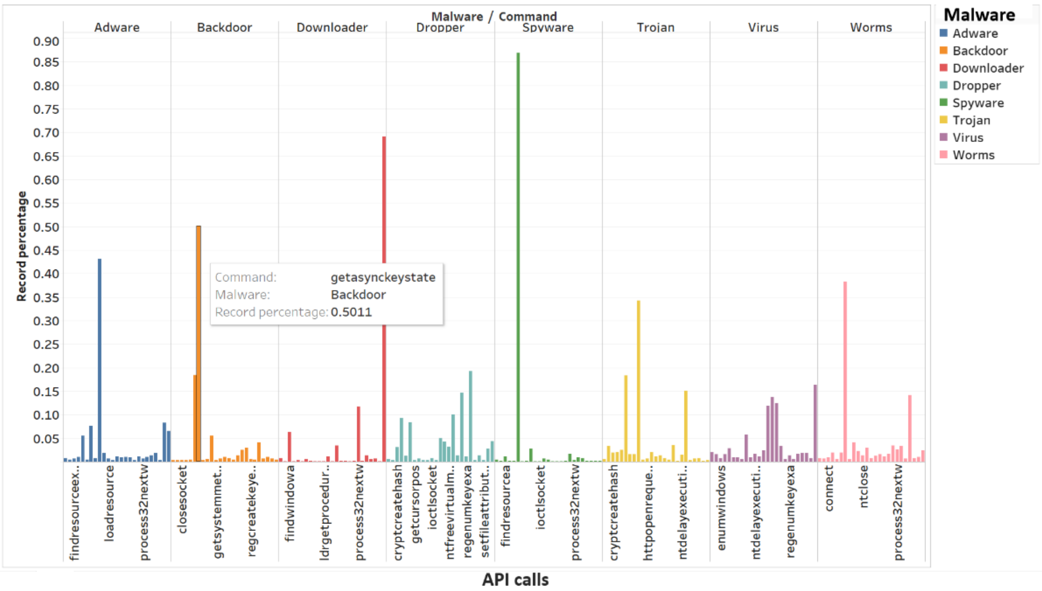


Figure 2. API calls frequency of each malware

4. METHODS

4.1. Categorical Vector with Convolutional Neural Network

Two separate analytic methods were used for the API call dataset. First, a one-dimensional convolutional neural network (CNN) was used to classify the type of malware of a malicious program based on its produced API call stream. For the data to be formatted for the CNN, preprocessing was necessary. Each sample was comprised of a list of system API calls. In order to reduce the dimensionality of the data, we reduced all call streams that were longer than 2000 API calls to its first 2000 API calls. This reduction effected 1466 of 6851 records. The dataset contains 278 unique API calls in total. These unique API calls may be repeated in the API call stream. Each unique API call was encoded into a categorical vector. The length of the vector is the same as the number of unique API calls plus one, totaling 279. For each unique API call, 1 was placed in the vector's index corresponding to that call and the rest of the vector was set to 0's. Additionally, a padding vector was created as a categorical vector with a 1 in the previously unused last index. This padding vector is then added to encoded API call streams that have a length less than 2000 until the API call stream's length reaches 2000. The padding vector allows equalizing the API call streams of different lengths so that we can provide batch optimization. Each unique API call mapped to an integer and converted to a categorical vector to use in CNN. Figure 3 provided a specific example. As seen in Figure 3, API call "ldrloaddll" was mapped to 133 and API call "ldrgetprocedureaddress" was mapped to 132 according to their positions in alphabetical order (see Figure 10). Then, using their mapped value, categorical vectors are obtained by replacing with 1 at position 132 and the rest of the index with 0 for the "ldrloaddll" call. For the "ldrgetprocedureaddress" call, we placed 1 at position 133 and 0 at the rest of the index. These categorical vectors represented the unique API calls and they were substituted with their corresponding position in a given API call sequence.

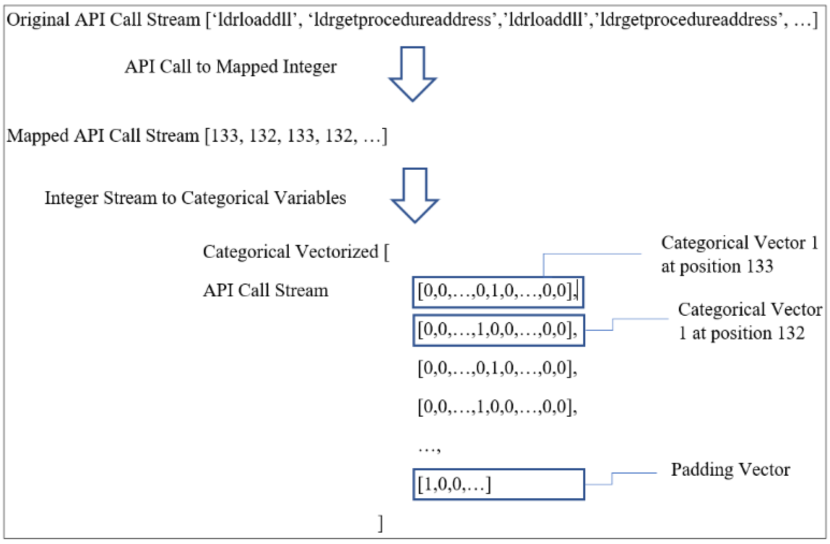

Original API Call Stream [‘ldrloaddll’, ‘ldrgetprocedureaddress’,’ldrloaddll’,’ldrgetprocedureaddress’, …]
API Call to Mapped Integer
Mapped API Call Stream [133, 132, 133, 132, …]
Integer Stream to Categorical Variables
Categorical Vectorized [
API Call Stream
[0,0,…,0,1,0,…,0,0],
[0,0,…,1,0,0,…,0,0],
[0,0,…,0,1,0,…,0,0],
[0,0,…,1,0,0,…,0,0],
…,
[1,0,0,…]
]
Categorical Vector 1 at position 133
Categorical Vector 1 at position 132
Padding Vector

Figure 3. Encoding methodology for categorical vector

The convolutional neural network model architecture is shown in Figure 4. The first layer of the network is a 1-D convolutional layer, this uses convolutional filters to create a feature map from an input API call stream seen as a vector with 279 channels. This layer uses 64 filters with ReLU (Rectified linear unit function) activation function [22], which has an almost linear function and therefore retains the properties of linear models. These properties provide ease to optimize with gradient descent methods. This activation function rectifies the input values that have less than zero, by forcing them to be zero [23]. The following two layers flatten the generated feature map using mean-pooling [23, 24] on segments of length two within the generated feature map. Pooling is used to gradually decrease the dimensions of the attribute representation. Hence, it provides low computational cost by shrinking memory cost and the number of parameters [23,24]. Mean-pooling calculates the average of each feature values of the feature map [24]. The next layer is a standard feed-forward layer in a deep learning architecture which transforms the generated feature vector using ReLU activation [22], shown in Eq. (1). The final layer then uses softmax activation [22] to generate a vector, shown in Eq. (2). Softmax provides an output whose value ranges between 0 and 1 and returns probabilities of each class [22]. Softmax represents the probabilities of the input malware API call stream belonging to each class. Each class being statically assigned to a specific index of this probability vector within the training data.

$$ReLU(x) = x^{+} = max(0,x) \tag{1}$$

where x is the pre-activation output value of the nodes.

$$SoftMax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2}$$

where x is an input vector of pre-activation values. The letter e represents the base of the natural logarithm system.

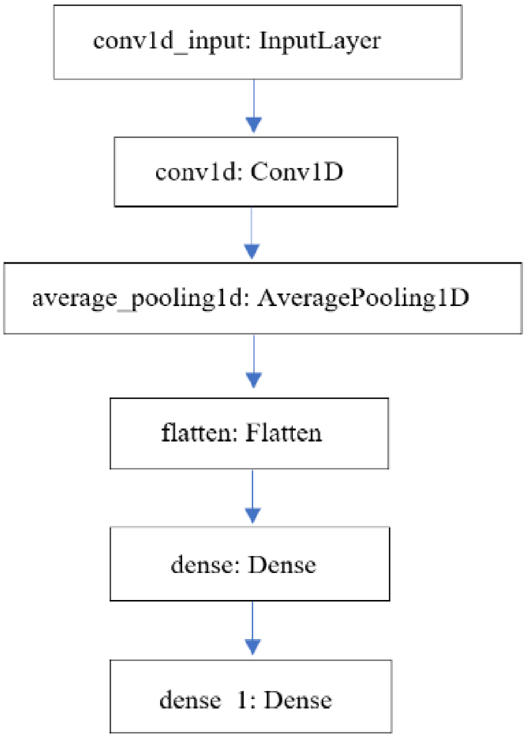


Figure 4. Model Architecture

The proposed 1-D CNN model and encoding methodology can be used to classify multiclass malware API call streams. Converting a list of words to a categorical variable, can be utilized in any text data. This encoding provides ease in the analysis and classification. After encoding, the index of the largest probability within the output probability vector will correspond to the predicted class.

4.2. TF-IDF Vector

Another computational approach used for the API call dataset was the use of text-based analysis. N-grams, described in [25], are sequential series of words or terms in an ordered sequence. In the case of this research, N-grams can be created from a moving window of API calls along the call streams in the dataset. Due to the nature of the API call streams (hundreds or thousands of terms in which calls were often repeated sequentially), 10-grams, sequences of ten terms, were used. Figure 5 provides general structure of n-grams. We provided first three grams for API call streams as an example with a small piece of API call sequence. As seen in Figure 5, when unigram is utilized (N=1), each API call is grouped with only 1 API call. Consequently, we obtained 4 unigrams, namely “getAsyncKeyState”, “ldrloaddll”, “ldrgetprocedureaddress”, and “thread32next”. Similarly, for bigram (N=2), API calls are grouped with only 2 consecutive calls. In this case, we obtained 3 bigrams: “getAsyncKeyState, ldrloaddll”, “ldrloaddll, ldrgetprocedureaddress”, and “ldrgetprocedureaddress, thread32next”. In trigram (N=3), we obtained 2 trigrams that 3 consecutive API calls including “getAsyncKeyState, ldrloaddll, ldrgetprocedureaddress” and “ldrloaddll, ldrgetprocedureaddress, thread32next”. We created 10-grams from a moving window of API calls along the call streams. To parse API call streams, we modeled call streams to represent each n-gram as *n* words, where *n* is the position of the API call.

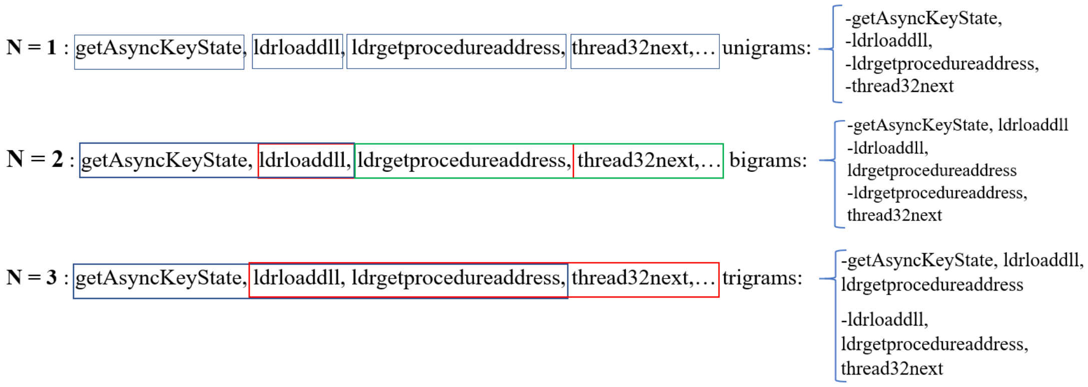

N = 1 : getAsyncKeyState, ldrloaddll, ldrgetprocedureaddress, thread32next,… unigrams:
-getAsyncKeyState,
-ldrloaddll,
-ldrgetprocedureaddress,
-thread32next
N = 2 : getAsyncKeyState, ldrloaddll, ldrgetprocedureaddress, thread32next,… bigrams:
-getAsyncKeyState, ldrloaddll
-ldrloaddll,
ldrgetprocedureaddress
-ldrgetprocedureaddress,
thread32next
N = 3 : getAsyncKeyState, ldrloaddll, ldrgetprocedureaddress, thread32next,… trigrams:
-getAsyncKeyState, ldrloaddll,
ldrgetprocedureaddress
-ldrloaddll,
ldrgetprocedureaddress,
thread32next

Figure 5. N-grams representation

These identified vectors of terms were translated to Term Frequency, Inverse Document Frequency (TF-IDF) vectors. TFIDF vectors are described in [26]. Term frequency is used to calculate the number of times a term is present in a document [27]. Inverse document frequency is a measure of information provided by words. IDF assigns a value to the word according to their rareness. If a word is frequent, then IDF assigns less weight and if a word is infrequent the more weight is assigned. The formulation of TF, IDF and TF-IDF are provided in Eq. (3), Eq. (4) and Eq. (5), respectively [26].

$$TF(x) = log(x+1) \tag{3}$$

$$IDF(x) = \log(\frac{1+p}{1+d})+1 \tag{4}$$

$$TF\text{-}IDF = TF * IDF \tag{5}$$

where x is a vector of raw frequencies, p is the number of tracks in the X set, and d is a vector that counts the tracks where every 10-grams appears.

First, we determined TF for each API call. Then IDFs are calculated and multiplied with TF values to get TD-IDF values. These TF-IDF vectors were used as inputs for a suite of decision tree classifiers for both binary cases (i.e., Trojan versus Not Trojan, and so on) and for the multiclass classification problem. Implementations of these decision trees were used: Random Forest (RF), Bernoulli Naïve Bayes (BNB), Multinomial NB (MNB), Gaussian NB (GNB), Decision Tree (DT) Classifier, AdaBoost Classifier, BaggingClassifier, kNN, and MLP. Additionally, we also used CNN to classify the TD-IDF vector. We utilized the same CNN structure as described in the previous section, only changing its input layer to accept the larger 14666 length TF-IDF vectors. This TF-IDF based CNN performed competitively with the previously described CNN which received API call streams as input directly. This is likely the case as the CNN which receives the API call stream directly is able to produce a feature map that is more effective than the TF-IDF feature extraction we provide. Python scikit-learn library was used for the n-grams, TF-IDF vectors, decision tree, and CNN implementations.

4.3. Visual Analytic Methods

This research used a variety of visual analytic methods to explore the dataset as well as the computational performance of the analytical model. There are four metrics adopted in our experiment with the malware API dataset, namely length, unique instances, varibility and call sequence. Thus, a visual tool that can highlight and compare the metrics per each malware type would help to analyze a dataset initially and assist in computation model choice for machine learning. Tableau was used for data exploration and preliminary analysis of feature correlations. D3.js, a JavaScript tool, was used to develop interactive, web-based visualization platforms for exploring datasets and computational results.

5. Results

5.1. Categorical Vector with Convolutional Network

We converted API call streams to categorical variables to employ 1-D CNN. We utilized 80-20 train-test split of API call stream data. The proposed 1-D CNN model has achieved an accuracy of 91.0%, a macro F1 score of 91.3%, and a weighted F1 score of 91.0%. We compared our proposed algorithm with Random Forest (RF) classifier, Logistic Regression (LR), Support Vector Machine (SVM), and k-nearest neighbor (kNN) with 3, 5 and 7 neighbors. A comparison of our model and the existing classification algorithms is shown in Figure 6. 1-D CNN outperformed the existing algorithms with 91.0% accuracy score. The algorithm that performed most competitively with CNN was the Random Forest which achieved an accuracy of 89.8% (-1.2%), a macro F1 score of 90.3% (-1.0%), and a weighted F1 score of 89.8% (-1.2%). Naïve Bayes algorithm has the lowest accuracy and F1 scores. For kNN, an increase in the number of neighbors affected accuracy scores, adversely. The confusion matrix for CNN is shown in Figure7. Confusion matrix shows that each malware has high true positives.

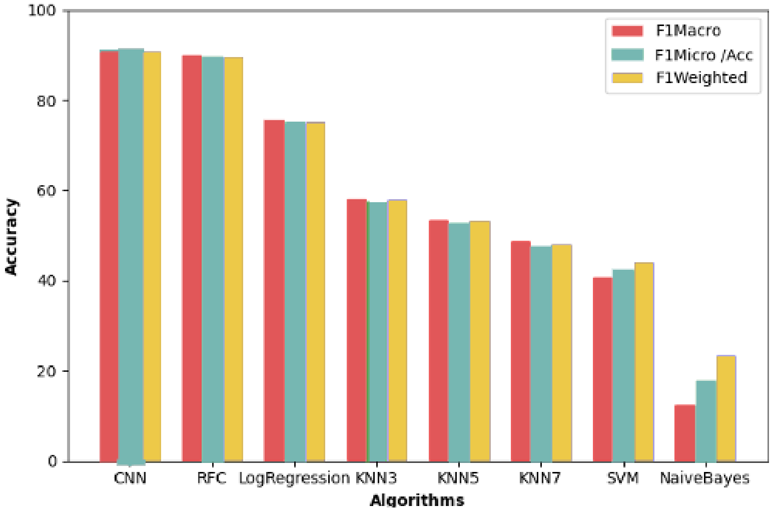


Figure 6. Traditional models compared to our 1-D CNN predicting the types of multiclass malware using TF-IDF vectors

5.2. TF-IDF Vector

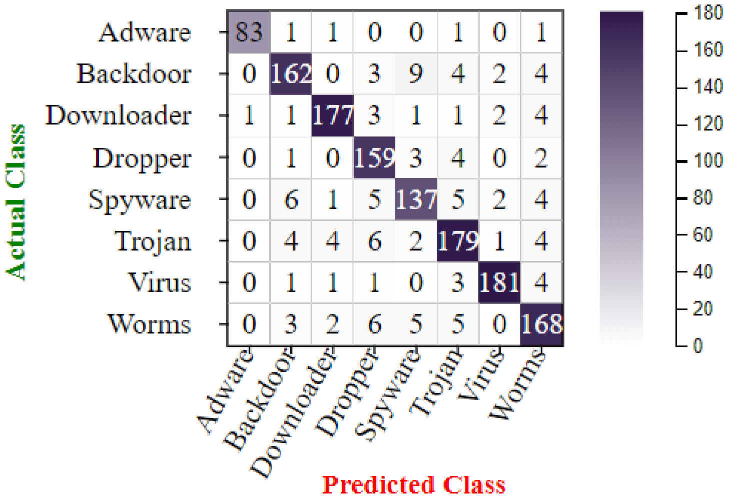
When using TF-IDF feature extraction, CNN outperformed the decision tree classifiers including AdaBoost DT, Random Forest, Bernoulli Naïve Bayes (NB), Multinomial NB, Gaussian NB, Decision Tree Classifier, Bagging Classifier, kNN, and MLP. The CNN model has achieved an accuracy of 91.0%, a macro F1 score of 91.3%, and a weighted F1 score of 91.0% Of the suite of the decision tree classifiers, AdaBoost Decision Tree performed the best among other algorithms.MLP has the worst accuracy values, specifically at classification of Adware, Dropper, and Spyware. Table 2 summarized these results on class accuracy in detail.

5.3. Overall Comparison

Table 2 presents the overall comparison in terms of class accuracy for the two approaches of feature extraction. When comparing CNN performance between TF-IDF vector with categorical vector, the highest class accuracy varied depending on the type of malware. The performances are competitive with differences of accuracy ranged from 0.26%-7.32% for eight classes respectively. The highest accuracy scores are highlighted in bold, and scores in the 2nd rank are underlined for each type of malware. Specifically, CNN performed the best at classification of malware type, suggesting that this form of malware is the most unique relative to its counterparts.

Table 2. Performance comparisons on class accuracy scores

	Categorical Vector	**TF-IDF Vector**									
Malware Type	**CNN**	**CNN**	**AdaBoost**	**Bagging**	**BNB**	**DT**	**GNB**	**kNN**	**MLP**	**MNB**	**RF**
Adware	*95.40%*	**98.17%**	76.00%	48.00%	51.00%	45.00%	70.00%	70.00%	0%	32.00%	48.00%
Backdoor	*88.00%*	**88.26%**	52.00%	46.00%	40.00%	40.00%	53.00%	57.00%	46.00%	52.00%	62.00%
Downloader	**93.20%**	*91.14%*	69.00%	54.00%	51.00%	50.00%	59.00%	67.00%	64.00%	64.00%	52.00%
Dropper	**94.10%**	*86.78%*	57.00%	60.00%	27.00%	37.00%	23.00%	45.00%	1.50%	29.00%	35.00%
Spyware	*85.60%*	**88.54%**	41.00%	38.00%	23.00%	11.00%	25.00%	32.00%	0%	8.30%	17.00%
Trojan	**89.50%**	*84.04%*	51.00%	45.00%	11.00%	16.00%	18.00%	32.00%	20.00%	32.00%	16.00%
Virus	**94.80%**	*92.26%*	74.00%	66.00%	75.00%	41.00%	77.00%	62.00%	72.00%	63.00%	80.00%
Worms	**88.90%**	*87.69%*	57.00%	52.00%	28.00%	78.00%	34.00%	49.00%	29.00%	32.00%	58.00%



	Adware	Backdoor	Downloader	Dropper	Spyware	Trojan	Virus	Worms
Adware	83	1	1	0	0	1	0	1
Backdoor	0	162	0	3	9	4	2	4
Downloader	1	1	177	3	1	1	2	4
Dropper	0	1	0	159	3	4	0	2
Spyware	0	6	1	5	137	5	2	4
Trojan	0	4	4	6	2	179	1	4
Virus	0	1	1	1	0	3	181	4
Worms	0	3	2	6	5	5	0	168

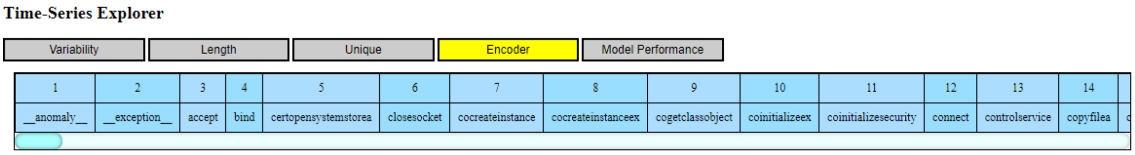
Figure 7. 1-Dimensional convolutional neural network confusion matrix

5.4. D3 Visualizations

To analyze API call streams, we presented a visualization tool by using D3, a JavaScript library. We considered API calls as time series and examined their features. We determined length, uniqueness and variability features to investigate malware behaviors.

The distribution of the length, unique instances, and variability metrics is viewed per malware class within a API call stream dataset. These metrics can be defined as follows:

- Length: Number of entries in a time-series record
- Unique Instances: Number of unique entries in a time-series record
- Variability: Number of times in a time-series where entry n differs from entry n + 1, divided bylength of the time-series minus 1.



1	2	3	4	5	6	7	8	9	10	11	12	13	14
__anomaly__	__exception__	accept	bind	certopensystemstorea	closesocket	cocreateinstance	cocreateinstanceex	cogetclassobject	coinitializeex	coinitializesecurity	connect	controlservice	copyfilea

Figure 8. A representation of the encoding map for a given time-series

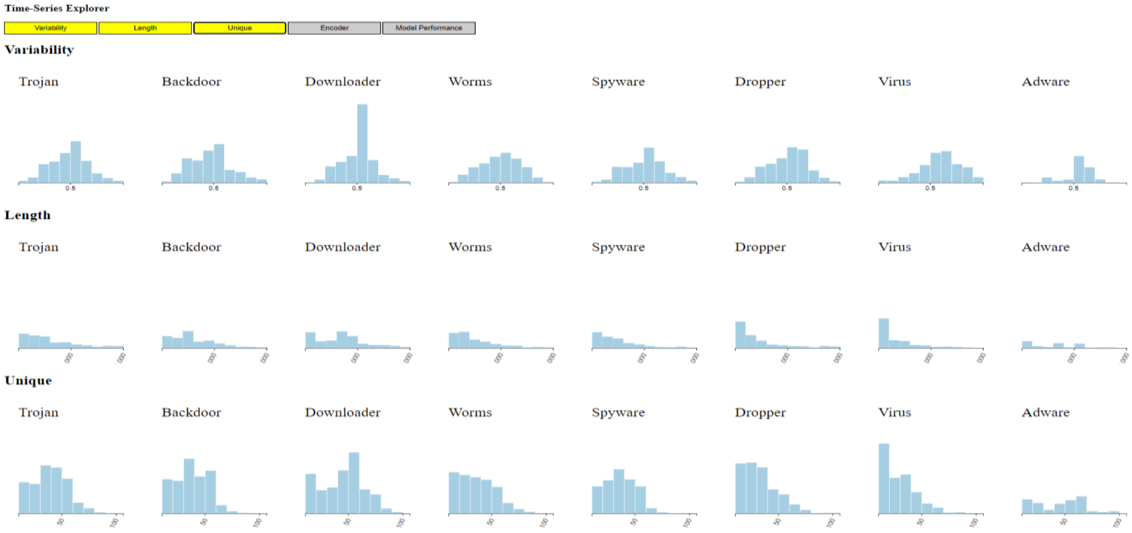


Figure 9. The selected metric distributions of malware over the system API calls

We also created an encoder view to see unique API calls and their assigned integer values, which is used in mapping. Fig. 8 shows the encoder for each API calls. The encoder for the time series data can be used to be inspected for any errors as well. Activated button returns yellow color. Fig. 9 shows the length, uniqueness and variability metrics for each malware. These metric buttons can be activated simultaneously. This helps to analyze data comprehensively. Variability metric indicates that downloader (3rd) and adware (8th) have different variability among other types of malware. This justifies that Adware has the best accuracy score with both approaches of feature extractions under CNN because it has a lower than usual variability. We also added Model Performance button to reach confusion matrix of 1-D CNN. The tool has ability to analyze any time series data with similar structure. In this application a user can upload a dataset of time series data and find out how a model performs in classification of each record through our designed metrics. The visualization will help users to view various attributes in determining the model performance.

6. Conclusion and Discussion

We utilized two different approaches of feature extraction to classify a malware API dataset at both binary and multiclass levels. In our first approach, we encoded API call streams by converting them to categorical variable. We proposed 1-D CNN and compared results with four other ML methods, namely RF, LR, SVM, and k-NN. We conducted text-based analysis in our 2nd approach. We used 10-grams for API calls and converted them to TD-IDF vectors. A set of decision tree classifiers and CNN were used to classify the malwares using these TD-IDF vectors.

This research has ultimately produced Convolutional NeuralNetwork models that achieved above90% accuracy in classifying the type of malware using a malicious program based on its system call stream. For the purpose of identifying and classifying malware, the results of this research demonstrate the advantages of using a CNN to label the type of malware that a malicious system API call stream belongs to. When using categorical vector, the proposed CNN outperformed RF, LR, SVM, and k-NN. While using TF-IDF vector, CNN also received the highest accuracy than AdaBoostDT, Random Forest, Bernoulli Naïve Bayes (NB), Multinomial NB, Gaussian NB, Decision Tree Classifier, Bagging Classifier, kNN, and MLP. Overall comparisons indicate that both feature extraction approaches produced competitive performance on classification of malware. Depending on the type of malware, the class accuracy varies between 0.26%-5.4% respectively. The results also suggest that the high impact feature in an API call stream executed by a malware are related to both call variability and call sequence.

We also demonstrate a visual analysis platform for time-series data to assist our machine learning model choices. The D3 visualization tool aided in human understanding of the API call stream data. The distributions of the variability, uniqueness, and length of call streams can be inspected using the visualizer. The tool also helps other time series formatted data as well.

Future work includes expanding the length of call streams and including non-malicious programs for classification. An anti-virus system can adapt the model to aid in attack attribution by quickly gaining an understanding as to the type of malware they are dealing with. Finally, a goal for improving the visual analytic process of this research is to use the correlations identified in the D3 tool to adjust the computational model so that it can be more successful in differentiating between similar classes.

References

[1] Daniel Gibert, Carles Mateu, & Jordi Planes, (2020) “The rise of machine learning for detection and classification of malware: Research developments, trends and challenges”, Journal of Network and Computer Applications. 10.1016/j.jnca.2019.102526.

[2] Zahra Bazrafshan, Hashem Hashemi, Fard Hazrati, Mehdi Seyed, & Ali Hamzeh, (2013) “A survey on heuristic malware detection techniques”, 2013 5th Conference on Information and Knowledge Technology. 113-120. 10.1109/IKT.2013.6620049.

[3] Jyoti Landage, & M. P. Wankhade, (2013) “Malware and Malware Detection Techniques : A Survey”, International journal of engineering research and technology, 2.

[4] Dainius Ceponis, & Nikolaj Goranin, (2019) “Evaluation of Deep Learning Methods Efficiency for Malicious and Benign System Calls Classification on the AWSCTD”, Security and Communication Networks, 2317976:1-2317976:12.

[5] SerifBahtiyar, Mehmet Baris Yaman, & Can Yilmaz Altinigne, (2019) “A multi-dimensional machine learning approach to predict advanced malware”, Comput. Networks, 160, 118-129.

[6] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, & Sungroh Yoon, (2016) "LSTM-Based System-Call Language Modeling and Robust Ensemble Method for Designing Host-Based Intrusion Detection Systems", ArXiv, abs/1611.01726.

[7] AhmetYazi, Ferhat Ozgur Catak, & EnsarGul, (2019) "Classification of Methamorphic Malware with Deep Learning (LSTM)", 10.1109/SIU.2019.8806571.

[8] Ferhat Ozgur Catak, & AhmetYazi, (2019) "A Benchmark API Call Dataset for Windows PE Malware Classification", https://arxiv.org/abs/1905.01999.

[9] Eslam Amer, & Ivan Zelinka, (2020) "A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence", Computers & Security. 10.1016/j.cose.2020.101760.

[10] Yuntao Zhao, Bo Bo, Yongxin Feng, ChunYu Xu, & Bo Yu, (2019) "A feature extraction method of hybrid gram for malicious behavior based on machine learning", Secur. Commun. Netw.

[11] Chang Choi, Christian Esposito, Mungyu Lee, & Junho Choi, (2019) "Metamorphic malicious code behavior detection using probabilistic inference methods", Cognit. Syst. Res. 56, 142–150.

[12] Asghar Tajoddin, & Saeed Jalili, (2018) "HM3alD: polymorphic Malware detection using program behavior-aware hidden Markov model", Appl. Sci. 8 (7), 1044.

[13] Jeffrey Heer, Micheal Bostock, & Vadim Ogievetsky, (2010) "A Tour through the Visualization Zoo", ACM Queue, 8, 20.

[14] Weijie Han, Jingfeng Xue, YongWang, Lu Huang, Zixiao Kong, & Limin Mao, (2019) "MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics", Comput. Secur., 83, 208-233.

[15] Lu Xiao-Feng, Zhou Xiao, Jiang Fangshuo, Yi Sheng-wei, & Sha Jing, (2018) "ASSCA: API based Sequence and Statistics features Combined malware detection Architecture", Procedia Computer Science, 129, 248-256.

[16] Matilda Rhode, Pete Burnap, & Kevin Jones, (2018) "Early Stage Malware Prediction Using Recurrent Neural Networks", Comput. Secur., 77, 578-594.

[17] Zahra Salehi, Ashkan Sami, & Mahboobe Ghiasi, (2017) "MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values", Eng. Appl. Artif. Intell., 59, 93-102.

[18] MohamedBelaoued, & Smaine Mazouzi, (2016) "A Chi-Square-Based Decision for Real-Time Malware Detection Using PE-File Features", JIPS, 12, 644-660.

[19] Sanchit Gupta, Harshit Sharma, & Sarvjeet Kaur, (2016) "Malware Characterization Using Windows API Call Sequences", SPACE.

[20] Jixin Zhang, Zheng Qin, Hui Yin, Lu Ou, & Kehuan Zhang, (2019) "A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding", Comput. Secur., 84, 376-392.

[21] Tableau Software. (2020). Retrieved from www.tableau.com.

[22] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, & Stephen Marshall, (2018) "Activation Functions: Comparison of trends in Practice and Research for Deep Learning", ArXiv, abs/1811.03378.

[23] Yinzheng Gu, Chuanpeng Li, & Jinbin Xie, (2018) "Attention-aware Generalized Mean Pooling for Image Retrieval", ArXiv, abs/1811.00202.

[24] Mark Cheung, John Shi, Lavender Jiang, Oren Wright, &Jose Moura, (2019) "Pooling in Graph Convolutional Neural Networks", 53rd Asilomar Conference on Signals, Systems, and Computers, 462-466.

[25] William Cavnar, & John Trenkle, (1994) "N-gram-based text categorization", Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval. Vol. 161175.

[26] Raymond Canzanese, Spiros Mancoridis, & Moshe Kam, (2015) "Run-time classification of malicious processes using system call analysis", 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, 2015, pp. 21-28.

[27] ShahzadQaiser, & Ramsha Ali, (2018) "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents", International Journal of Computer Applications, 181, 25-29.

AUTHORS

Matthew Schofield is currently enrolled at Rowan University pursuing his B.S/M.S degree in Computer Science anticipating graduation in December 2021. He is currently working on his master’s thesis on Deep Reinforcement Learning in Incentivization Systems. His research interests are in Machine Learning and Deep Reinforcement Learning.

Gulsum Alicioglu received M.Sc. Degree in Industrial Engineering from Gazi University, Turkey, in 2018. Currently, she is a Ph.D. candidate at the Department of Electrical and Computer Engineering of Rowan University, USA. Her research interests are data visualization, machine learning and explainable artificial intelligence.

Russell Binaco graduated from Rowan University with a M.S. in Computer Science in Spring 2020. He now works as a software engineer for Innovative Defense Technologies, and as an adjunct faculty for Rowan University. At Rowan, he earned undergraduate degrees in Computer Science and Electrical and Computer Engineering.

Paul Turner received his B.S. in Computer Science from Rowan University in 2018 and is currently enrolled in a M.S. program at Rowan University. His research interests include machine learning, text mining, and cloud computing.

Cameron Thatcher received his B.S in Computer Science from Rowan University in 2019 and is currently pursuing his M.S. in Computer Science at Rowan University. His research interests include Machine Learning and Data Mining.

Alex Lam is currently attending Rowan University pursuing his B.S/M.S degree in Computer Science and Data Analytics. His research interests include Machine Learning.

Bo Sun received her B.S. in Computer Science from Wuhan University, her M.S. in Computer Science from Lamar University, and her Ph.D. in Modeling and Simulation from Old Dominion University. She is an associate professor of Computer Science at Rowan University. Her research interests are Visual Analytics, Data Visualization, Serious Gaming, and Virtual Reality/Augmented Reality-based Simulation.