API Call Based Malware Detection Approach Using Recurrent Neural Network—LSTM

J. Mathew(✉) and M. A. Ajay Kumara

Department of Computer Science and Engineering,
Amrita School of Engineering,
Bengaluru Amrita Vishwa Vidyapeetham, Bengaluru, India
mailmathewjoy@gmail.com, ma_ajaykumara@blr.amrita.edu

Abstract. Malware variants keep increasing every year as most malware developers tweak existing easily available malware codes to create their custom versions. Though their behaviours are coherent, because of change in signature, static signature-based malware detection schemes would fail to identify such malware. One promising approach for detection of malware is dynamic analysis by observing the malware behaviour. Malware executions largely depend on Application Programming Interface (API) calls they issue to the operating systems to achieve their malicious tasks. Therefore, behaviour-based detection techniques that eye on such API system calls can deliver promising results as they are inherently semantic-aware. In this paper, we have used Recurrent Neural Network's (RNN) capability to capture long-term features of time-series and sequential data to study the scope and effectiveness of RNNs to efficiently detect and analyze malware and benign based on their behaviour, i.e. system call sequences specifically. We trained the RNN-Long Short Term Memory (LSTM) model to learn from the most informative of sequences from the API-dataset based on their relative ranking based on Term Frequency-Inverse Document Frequency (TF-IDF) recommended features and were able to achieve accuracy as high as 92% in detecting malware and benign from an unknown test API-call sequence.

Keywords: API system call · Malware detection · TF-IDF · RNN-LSTM

1 Introduction

The word 'Malware' is formed by the vicious combination of two words- 'malicious' and 'software'. It refers to programs that are deliberately designed to penetrate systems of high value; gain access through unauthorized means gravely undetected with the sole purpose to steal data, rupture business and bring it to a standstill, or sometimes systems even being held hostages for ransom. The form and features of malware evolve continuously and it demands that the technology

A. Abraham et al. (Eds.): ISDA 2018, AISC 940, pp. 87–99, 2020.
https://doi.org/10.1007/978-3-030-16657-1_9

used to detect and stop it should also up the ante and get ahead of the curve. The current generation malwares are mostly clickless, living off one's own systems and has a rise in worming components[1]. In [1], a case study was conducted on WannaCry ransomware (which in 2017 had infected 400,000 machines in 150 countries in no time) to automatically identify their distinguishing features from system logs and API calls played a big role in the analysis.

Static analysis extracts information without executing the sample but analyzing all possible execution paths from the binary code. It is prone to code obfuscations. Behavioral-based dynamic analysis [13] is done by executing the binary sample to run in a virtual machine environment. API calls provide the interface between user programs and Operating Systems (OS) by requesting services from the kernel of the OS. API provides a set of well-defined commands that programs will have to invoke and make use of. A truly effective means of defending against rapidly evolving attacks is to deploy solutions that can learn and recognize common behaviors and elements that continue to be reused. A well-informed choice to track behavior is by analyzing system calls. In this paper, the proposed approach considers system calls as words and system call sequences as sentences. This is to leverage the idea that just as how meaningful arrangement of words make sensible sentences, API calls and their order would give us sensible information to study and predict malware and benign behaviors.

From a general scenario where conventional statistical Machine Learning (ML) techniques [2,5,6] are used for most malware detection and analysis, the significant contribution of this paper is that it uses API call sequences where the particular order of the data-points matter, to train an advanced RNN architecture- LSTM model to detect malware and benign programs. Finally by bringing together the good works of Natural Language Processing (NLP) algorithms and deep learning models, evaluating on a recent and conclusive dataset and achieving fair and acceptable results, this paper seeks to prove that RNN-LSTM realization of a malware detection model based on API call sequences is worthy of further study as it would do great justice and present a larger scope in terms of resilience and interpretation.

2 Related Work

Malware classification and detection approaches have seen many research ideas and inquisitive models over the years. Though many early works like [2–6], etc. attempted behavioral analysis, they were unable to self-learn patterns because all of them used conventional machine learning techniques for model evaluation. In an attempt to recognize malware functions based on the system calls they relied on, Ki *et al.* in [10] used sequence alignment algorithms for common call sequence extraction. However, the computing cost of sequence alignment algorithms was too high. Adding the developments of NLP into API call sequence analysis, Tran *et al.* in [11] introduced doc2vec, N-gram and TF-IDF methods along with conventional ML algorithms. Due to the lack of samples, the model

[1] The 3 Biggest Malware Trends to Watch in 2018. https://www.securityweek.com.

was evaluated only on classifying types of malware classes. With over 96% accuracy NLP algorithms proved its real worth. A large share of works on malware detection happened with respect to androids like Sugunan *et al.*'s [16]; but used conventional ML techniques.

The current trend in malware analysis demands learning models which can learn and seek to find patterns, sequences and other informative features. On the other hand, RNN provides us that learning edge of opportunity. Pascanu *et al.* in [12] pioneered the use of RNN to create a hybrid model of Echo State Networks as RNN model and an ML classifier. The model was meant to predict the next API call. Though the accuracy of the model is good, the length of the API call trace was custom-constrained. While Kolosnjaji *et al.* in [7] implemented a combination of Convolutional Neural Network (CNN) and feed-forward layers for malware classification and detection through static analysis of portable executable files and series of opcodes and achieved recall and F1-scores of 92%, but the input was not exactly sequential as convolutional filters were used. Thus the model was still not sequence-learning and vulnerable to advanced code obfuscation and detection evasion techniques. Nearly closing the gap, Tobiyama *et al.* in [8] focused to detect malware based on process behavior (logged operations) in infected terminals. A combination of RNN and CNN was used for feature extraction and malware detection. But here RNN was used only to extract the features and the model that was trained remained to be CNN.

However, Wang *et al.* in [14] designed a multi-tasking model using RNN that could identify malwares based on API call sequence analysis and also provide more interpret-ability of the results. While the work focused highly on malware class classification, effective detection of both malware and benign programs from a mix remains a potential opportunity. Based on LSTM networks, a new classifier was built to detect malware based on android call sequences by Xiao *et al.* in [15]. The classifier had two LSTMs, which were trained by benign and malware applications respectively. With a new sequence as input, similarity scores would be calculated from both the models, consequently leading to its classification. The model achieved good accuracy, but used two engines and was specific to Android malware detection, not Windows, etc.

Considering the insights obtained from these related works, this paper aims to study the effectiveness of RNN featured with N-gram and TF-IDF techniques to classify and detect unknown malware based on its API call sequence.

3 Background

In this section, we attempt to provide background notions of topics that form the basic building blocks of the proposed method.

3.1 API Call Sequence

Every single API call is an exact action performed by malware or benign activity on run state of the system, e.g. creation, read, write and deletion of files or

registry keys. We choose API call sequences as features since their order of calls show how malware or benign behaves with an operating system. Malware and benign will have their own specific API calls' patterns or unique order of the calls.

3.2 Natural Language Processing (NLP)

NLP is an automated way to understand and analyze natural human language and derive information from it using ML algorithms. NLP finds applications in automatic summarization, topic segmentation, machine translation, named entity recognition, etc. Since computers and programs have a fixed vocabulary of commands and their sequential usage always converges into an intended function, NLP can be applied to the analysis of API call sequences. The work by Tran *et al.* in [11] is a pioneer work that incorporated NLP algorithm to analyse malwares.

3.3 Recurrent Neural Network

An RNN would learn sequences as they are fed in because they can remember events that are presented to them in the past. They are called 'recurrent' because the same task is performed on every element in the sequence and the output depends on the present state and previous states' computations. Hidden state h_t at each time step t is given by

$$h_t = f(h_{t-1}, x_t) \tag{1}$$

where f is the activation function and x_t is the input at time step t. At each t, RNN performs the same calculations with same shared parameters on different inputs in the sequence. Figure 1 shows the unfolded RNN with U, V and W being the input, output and state weights respectively.

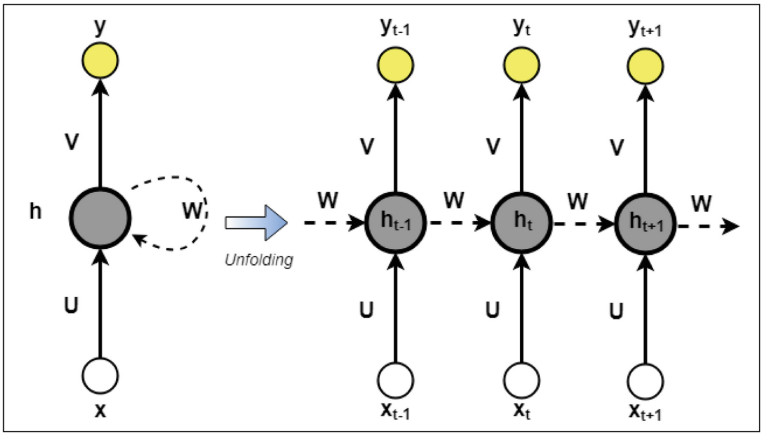

Fig. 1. Unfolded RNN

One of the major advantage with RNN is that the lengths of the input and output sequence can be different. As explained in [14], through the whole process of training an RNN to learn sequences- say to intake sequence $X = (x_1, x_2, x_3, ..., x_T)$ and output sequence $Y = (y_1, y_2, y_3, ..., y_{T'})$, the RNN will effectively learn the conditional distribution $p(Y|X)$, i.e. mathematically,

$$p(y_1, y_2, \ldots, y_{T'} \mid x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid h_{t-1}, y_1, y_2, \ldots, y_{t-1}) \tag{2}$$

Once the output is generated, then it is compared to the true value and error is generated. The error is then back-propagated in the network to update the weights and the network is thus trained. If we represent a sentence as W_1^N, containing N words of w_i then in the RNN, the probability of generating a sentence W_1^N can be represented as

$$p(W_1^N) = \prod_{m=2}^{N} p(w_m \mid W_1^{(m-1)}) \tag{3}$$

3.4 Long Short Term Memory

In standard RNNs, derivatives of *tanh* and sigmoid functions have 0 at both ends. Thus they will have zero gradients and drive other gradients in previous layers to 0 in back-propagation. Therefore, the RNN will eventually end up not learning long-range dependencies. This problem of vanishing gradients (also similarly exploding gradients) in RNNs is solved with an advanced RNN architecture called LSTMs. The repeating module in LSTM has four interacting neural layers that interact in a very special way. Regulated by structures called gates, LSTM has the ability to remove or add information (only as best required) to the cell state. The gates are composed of a sigmoid neural net layer and a pointwise multiplication operation. As the sigmoid layer outputs values between 0 and 1, the gates decide how much of each component should be let through.

4 Proposed Method

The main focus of this work is to realize and study effectively the use of RNN-LSTM to detect malware by interpreting the program behavior from their API system call sequences using the language model. The features extracted from the API call sequences are used for malware or benign detection. Figure 2 shows the model of our proposed system designed. The model is first trained to learn the API call sequence patterns. The model is then tested to evaluate its performance with the unknown (to the model) test dataset. Explained in the forthcoming subsections are the steps followed in our method.

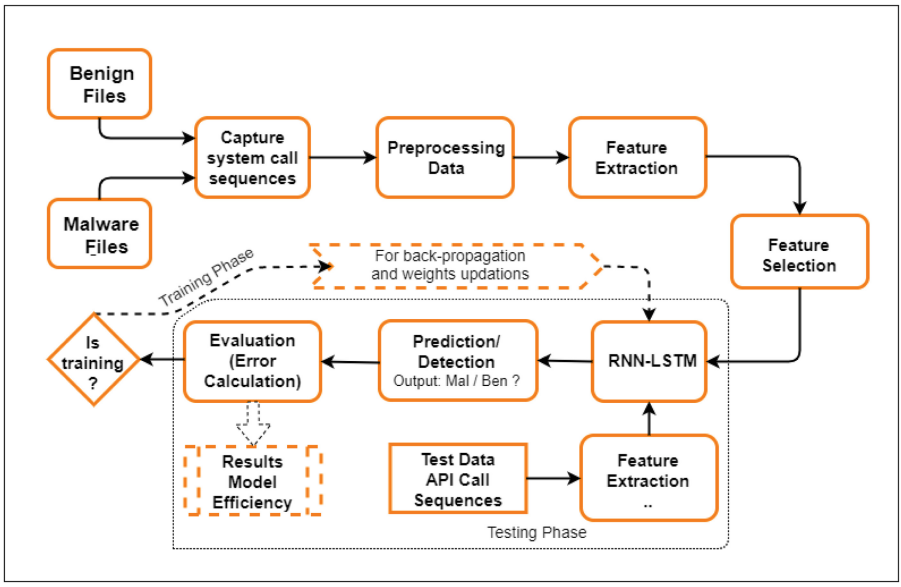

Fig. 2. System design

4.1 Data Preprocessing

To validate the proposed system, API call sequences are considered for behavioral analysis. Prior to training and classification, the data must be normalized and preprocessed to improve model convergence speed in training. As only system calls are considered as features, redundancies if present are removed. Extra fields are added if required. The resulting dataset is ensured to contain only unique system call sequences that are invoked by the malware and benign programs.

4.2 Feature Extraction

A challenge with API call traces is their variable lengths. N-grams model is used for feature extraction. In it, the sequences of N continuous system calls from the original call trace forms the features. These are obtained by performing n-length sliding window operation on all the API call traces. In this work, N is chosen as 10. It was a conservative choice for accuracy improvement based on previous work [17] as programs would typically require reasonably long sequences of API calls to fully serve its purpose.

4.3 Feature Selection and Vectorization

Feature selection techniques will judiciously trim down the training set to contain the most meaningful records. The engine would be better off handling a smaller weight of sensible data resulting in better performance while preserving

the accuracy. Feature selection reduces dimensions of the original feature vector, allowing ML techniques to effectively function. TF-IDF technique has been explored in this paper. Accuracy greater than 90% was achieved in this paper by using only a small portion (i.e. top TF-IDF ranked sequences, as discussed in Sect. 6) of the total 10-gram sequences generated.

Term Frequency-Inverse Document Frequency. TF is the frequency of a term in a document, while IDF quantifies the information provided by a term, relative to that term's presence in the other documents. Here, terms refer to API calls and documents refers to call sequences. For a particular type of malware, a particular system call may appear in the sequence corpus a significant number of times based on its relevance. While it can be seen that most programs tend to use some system calls as common in all of their sequences, there could be some characteristic system call pertaining to malware and benign, that occurs rarely. Such valuable system calls should have a higher rank compared to others. Information relative term weighting scheme TF-IDF provides a suitable ranking mechanism for the same purpose. In order to perceive TF-IDF mathematically, here are some of the terms and variables to be referred:

$$S = \{s_1, s_2, \ldots, s_n\} \qquad V = \{c_1, c_2, \ldots, c_n\} \tag{4}$$

S is the sequence corpus and s is a particular sequence. V is the vocabulary of system calls and c is a particular system call that appears in the corpora. Frequency of the system call c in a sequence s is given by (5) where $f_s(c)$ is the frequency of the system call $\mathrm{c} \in \mathrm{V}$ in $\mathrm{s} \in \mathrm{S}$.

$$tf(c, s) = f_s(c) \tag{5}$$

IDF is given as in (6) where $sf(s, c)$ is the number of sequences where the system call c appears. This is a logarithmic scaled value of the number of sequences in the corpus divided by the number of times system call c appears throughout the corpus.

$$idf(c, S) = log \frac{1 + |S|}{1 + sf(s, c)} \tag{6}$$

$$tfidf(c, s, S) = tf(c, s) * idf(c, S) \tag{7}$$

In (7), we see that the TF-IDF value increases proportionally by the frequency of c in a sequence, decreases proportionally by the log of the frequency of c in the corpus.

1-Hot Vectorization. After ranking the sequences by TF-IDF, only the top-ranked sequences will proceed further. This would reduce the loading on the RNN-LSTM engine, remove redundancies and ensure that the model is faithfully unbiased. 1-hot vector encoding method is used to convert these call grams into a numerical representation that can be fed into the neural network. List (or vocabulary) of various unique system calls in the dataset would form a dictionary.

The length of vector that represents each system call would be the size of the vocabulary. Each vector is a binary series of 1s and 0s where only that position in the vector corresponding to the position of the call in the dictionary would have a 1 and remaining of the vector would be 0.

4.4 Training the RNN-LSTM

LSTM is one of the best choices for a large dataset. Once the 10-gram sequences ranked on the basis of their TF-IDF scores, we discard the least significant sequences from the training dataset. Those sequences which are above a TF-IDF score and form the cream of informative call sequences are considered for model training. As noted in Sect. 4.3 a dictionary of API calls and sequence label is made. At each epoch of training, corresponding to each element in a sequence, their numerical indexes from the dictionary are passed into the LSTM as inputs for training the model and its parameters. RNN-LSTM has a hidden representation that gets updated with every new input. The output from the RNN is truly a vector of probability distributions. The value attributed to the highest probability in the vector is predicted as the output. The error is a cross-entropy between the predicted label and actual value. The parameters are then adjusted each time to minimize this error and thus the learning happens.

5 Implementation

5.1 Dataset Preparation

In this work, we used API system call traces sourced from 3000 malware and various benign traces obtained from Hacking and Countermeasure Research Lab[2] and [17] respectively. While the malware traces were extracted after executing them in a dynamic environment and made available by Kim [10], the benign traces were obtained using native Windows API tracer called NtTrace[3].

5.2 Feature Extraction and Selection

This stage is to do with the 10-gram extraction. 10 API calls from the sequences are bundled together and padded with the correct Malware or Benign label at the end. Each sequence would then essentially have 11 grams in one N-gram. This is required as we are implementing malware detection using RNN-LSTM Language Classifier model. *TfidfVectorizer* method from scikit-learn library is used to N-gramize and also calculate the TF-IDF scores of the resulting sequences. After generating N-gram features, about 250,000 N-gram sequences are generated. From these, equal shares of malware and benign N-gram sequences were utilized for model analysis. Once the scores and N-grams are ready, only the top-ranked sequences are used to train the model.

[2] http://ocslab.hksecurity.net/apimds-dataset.
[3] https://github.com/rogerorr/NtTrace.

5.3 Realizing the RNN-LSTM Learning Engine

The RNN-LSTM malware detection engine in this project was implemented with TensorFlow framework from Google. As can be seen in Fig. 3, an LSTM fed with sequences of system calls from the training dataset- with 10-grams as inputs and 11th being the label. After thousands of iterations, the model will eventually learn to predict the next gram correctly. 50,000 iterations were set as the general norm in this analysis. *RMSProp* [9] at a learning rate of 0.001 was used as the optimizer. The model had 2 layers of LSTM with 512 hidden units in each. The accuracy of the model can be bettered by optimization of the hyper-parameters.

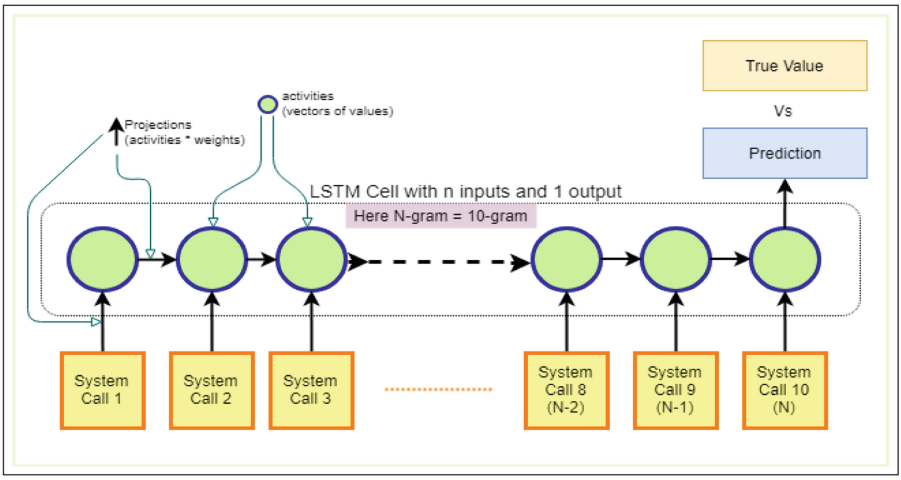

Fig. 3. RNN-LSTM

6 Evaluation and Results

The preprocessed dataset of N-gram feature sequences that are unknown to the trained model are selected for testing. Same test dataset is used for all the analyses, making the evaluation trustworthy. The model was assessed using top TF-IDF ranked 10000 , 7500 , 5000 and 2500 sequences. The evaluation metrics used can be seen in Table 1.

Table 1. Evaluation metrics

Metric	Formula
Accuracy	$(TP + TN)/(TP + FP + TN + FN)$
True Positive Rate (Recall)	$TP/(TP + FN)$
False Positive Rate	$FP/(FP + TN)$
Precision	$TP/(TP + FP)$
F1 Score	$2 * (\text{Recall} * \text{Precision})/(\text{Recall} + \text{Precision})$

As can be seen in Fig. 5 the model gained a good flair in detecting malware with a top average recall of 97%. Similar to the recall score, the overall average accuracy of the model increased with increase in the top TF-IDF sequences to reach as high as 92% for top 10K sequences (Fig. 4). With accuracy and recall improving with the increase in the top TF-IDF training sequences, it is noted that the model is not only able to detect both malware and benign with good prediction but is also gaining an upper hand in detecting malware given that it is present in the test sample. This can be optimized and targeted to reach above 97% with parameter optimization and more training.

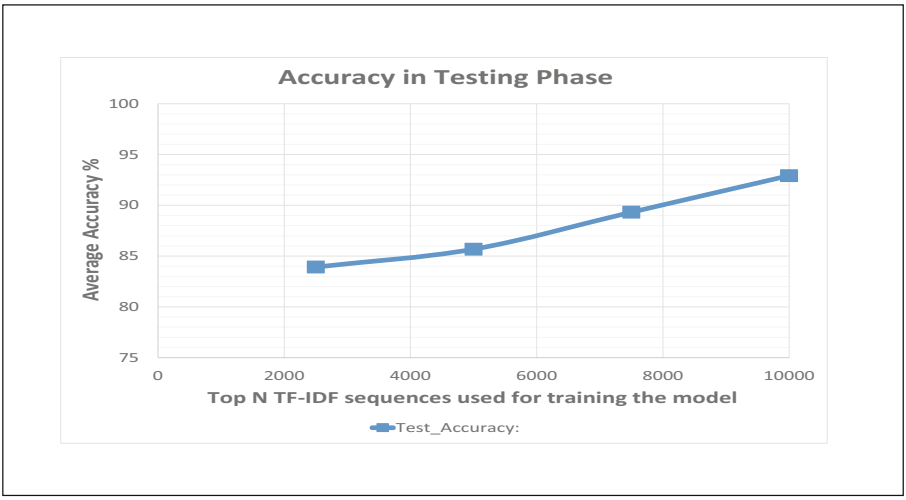

Fig. 4. Average accuracy in testing phase

While it would be always interesting to look at the error rate of the model in any evaluation, it would be less preferred in this paper as the model needs to be evaluated on how well it detects both malware and benign and not only one of it. Figure 6 presents a view of F1 Scores and most importantly the complement of it (1-F1), which represents the error loss of the LSTM model.

F1 Score which is the weighted average of Recall and Precision provides us with better visibility. While the average F1 Scores also follows suit just like accuracy and Recall in behaviour and achieves a 90% high for Top 10K sequences, it should be noted that the error complement(1-F1) reduces as the TF-IDF top sequences are selected in the range from 2,500 to 10,000. That is to say that the model is learning better and in a quick context to learn the API system call behavioural pattern of both benign and malware.

It should be noted that the iterations for all the analyses have been 50,000. Any model would expect a higher number of training sequences to have a higher number of training iterations. Thus, it has not been fair enough towards the

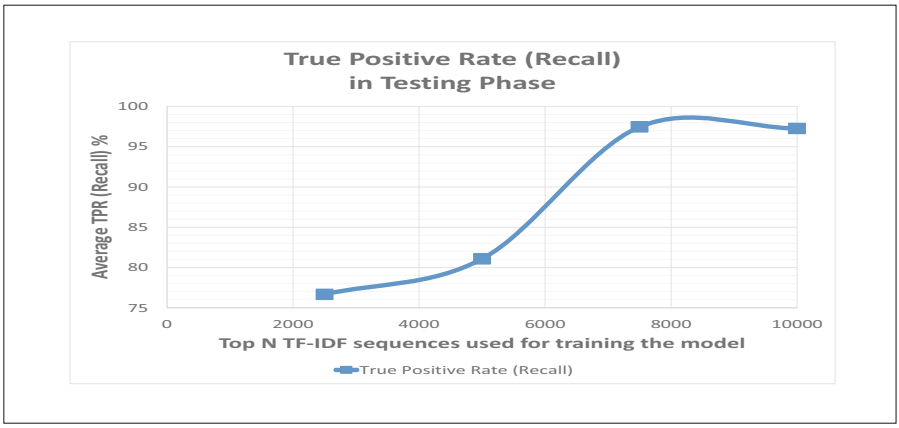

Fig. 5. Average recall (TPR) in testing phase

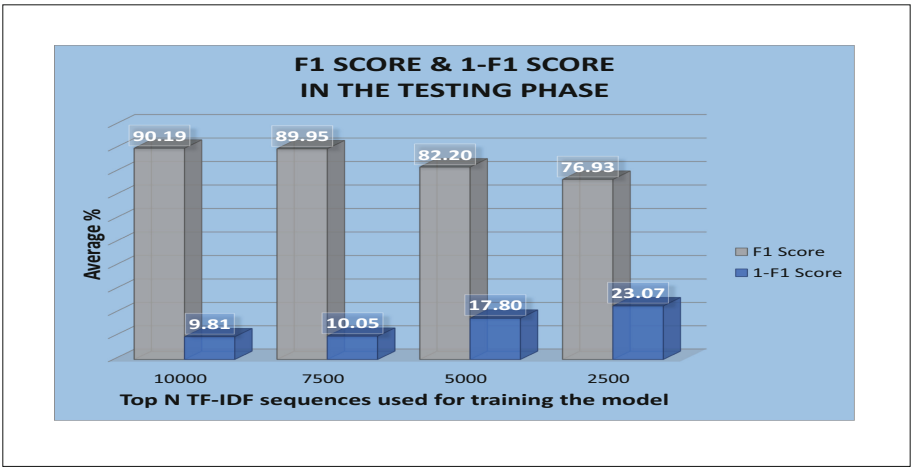

Fig. 6. Average F1 score & 1-F1 score in testing phase

designed RNN-LSTM model training owing to the varying sequences per iteration ratio. Yet, the model counteracts and provides high accuracy and lower error rate for a higher number of sequences. Thus it can be stated that increasing the iterations further will not only increase true detection, but also decrease false alarms and misses.

7 Future Work

The current model is evaluated using the features recommended only by one TF-IDF technique. The results obtained inspires confidence to optimize the parameters to increase the proposed model efficiency to a greater accuracy rate. The

approach followed must be evaluated by leveraging high-end systems on large datasets. The areas that would be explored in the extended work related to this paper would be to study the implications and effects of applying other popular feature selection techniques such as Chi-square, Fisher's score and combinations of these (hybrid), with different values of N-gram sizes, all evaluated over the extensive list of evaluation metrics.

8 Conclusion

In this work, we presented a behavioural-based classification of unknown malware by leveraging RNN-LSTM technique based on API call sequences. We used N-gram for feature extraction and TF-IDF for feature selection of the sequences. The novel contribution of this work incorporating RNN-LSTM for malware detection was performed for effective analysis of unknown malware. The system evaluated leverages the popular and powerful Google's TensorFlow framework. The training dataset was ensured to contain zero redundancy and equal share of top malware and benign sequences. It should be duly noted that the high accuracy system realized in this paper is a learning model that continuously seeks to recognize not only known patterns but also evolving unknown patterns of malware. This fact about the proposed model is what gives it a strong footing over the existing conventional statistics-based machine learning techniques. The promising results with a highest average accuracy of 92% and average recall of 97% achieved, it reassures faith to explore more the potential space of malware detection based on a combination of an RNN-LSTM model and API call sequences in the current series of researches.

References

1. Chen, Q., Bridges, R.A.: Automated behavioral analysis of malware a case study of WannaCry Ransomware (2017). arXiv:1709.08753v1 [cs.CR], Cryptography and Security
2. Ajay Kumara, M.A., Jaidhar, C.D.: Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM. Future Gener. Comput. Syst. **79**(Part 1), 431–446 (2018)
3. Anju, S.S., Harmya, P., Jagadeesh, N., Darsana, R.: Malware detection using assembly code and control flow graph optimization. In: Proceedings of the 1st Amrita ACM-W Celebration of Women in Computing in India, A2CWiC 2010, Coimbatore (2010)
4. Kang, B., Han, K.S., Kang, B., Im, E.G.: Malware categorization using dynamic mnemonic frequency analysis with redundancy filtering. Digit. Investig. **11**, 323–335 (2014)
5. Salehi, Z., Sami, A., Ghiasi, M.: Using feature generation from API calls for malware detection. Comput. Fraud Secur. **2014**, 9–18 (2014)
6. Galal, H.S., Mahdy, Y.B., Atiea, M.A.: Behavior-based features model for malware detection. J. Comput. Virol. Hacking Tech. **12**(2), 59–67 (2016)

7. Kolosnjaji, B., Zarras, A., Eraisha, G., Webster, G., Eckert, C.: Empowering convolutional networks for malware classification and analysis. In: International Joint Conference on Neural Networks (IJCNN) (2017)
8. Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., Yagi, T.: Malware detection with deep neural network using process behavior. In: 40th Annual Computer Software and Applications Conference (COMPSAC) (2016)
9. Athira, V., Geetha, P., Vinayakumar, R., Soman, K.P.: DeepAirNet: applying recurrent networks for air quality prediction. In: International Conference on Computational Intelligence and Data Science (ICCIDS) (2018)
10. Ki, Y., Kim, E., Kim, H.K.: A novel approach to detect malware based on API call sequence analysis. Int. J. Distrib. Sens. Netw. **11**, 659101 (2015)
11. Tran, T.K., Sato, H.: NLP-based approaches for malware classification from API sequences. In: 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES) (2017)
12. Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A.: Malware classification with recurrent networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2015)
13. Rhodes, M., Burnap, P., Jones, K.: Early-stage malware prediction using recurrent neural networks. Comput. Secur. **77**, 578–594 (2018). arXiv:1708.03513 [cs.CR]
14. Wang, X., Yiu, S.M.: A multi-task learning model for malware classification with useful file access pattern from API call sequence (2016). arXiv:1610.05945 [cs.SD], Cryptography and Security
15. Xiao, X., Zhang, S., Mercaldo, F., Hu, G., Sangaiah, A.K.: Android malware detection based on system call sequences and LSTM. Multimed. Tools Appl. (2017)
16. Sugunan, K., Gireesh Kumar, T., Dhanya, K.A.: Static and dynamic analysis for android malware detection. Advances in Intelligent Systems and Computing, vol. 645, pp. 147–155. Springer, Cham (2018)
17. Kim, C.W.: GitHub repository (2018). https://github.com/codeandproduce/NtMalDetect