Indonesian Journal of Electrical Engineering and Computer Science
Vol. 38, No. 2, May 2025, pp. 1106~1114
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v38.i2.pp1106-1114

Android malware detection through opcode sequences using deep learning LSTM and GRU networks

Annemneedi Lakshmanarao[1], Jeevana Sujitha Mantena[2], Krishna Kishore Thota[3], Pavan Sathish Chandaka[4], Chinta Venkata Murali krishna[5], Madhan Kumar Jetty[6]
[1]Department of Information Technology, Aditya University, Surampalem, India
[2]Department of Computer Science and Engineering, SRKR Engineering College(A), Bhimavaram, India
[3]Department of Computer Science and Engineering (Honors), Koneru Lakshmaiah Education Foundation (Deemed to be University), Vaddeswaram, India
[4]Department of Computer Science and Engineering, Chaitanya Engineering College, Visakhapatnam, India
[5]Department of Computer Science and Engineering (Data Science), NRI Institute of Technology, Pothavarappadu, India
[6]Department of Information Technology, R.V.R and J.C College of Engineering, Guntur, India

Article Info

Article history:

Received Aug 30, 2024
Revised Nov 6, 2024
Accepted Nov 11, 2024

Keywords:

Android malware detection
Deep learning
GRU
LSTM
Machine learning

ABSTRACT

Android malware detection was a complex task due to the intricate structure of Android applications, which consisted of numerous Java methods and classes. Effective detection required the extraction of meaningful features and the application of advanced machine learning (ML) or deep learning (DL) algorithms. This paper presented a novel approach to detecting Android malware by leveraging opcode sequences extracted from Android applications. These opcode sequences, which differed between malicious and benign apps, formed the basis of the detection model. The methodology involved extracting opcode sequences from decompiled Android APK files using the "Androguard" tool and applying recurrent neural networks (RNN) with long short-term memory (LSTM), Bi-LSTM, and gated recurrent unit (GRU) architectures to classify the apps as either malware or benign. The combination of these advanced DL techniques allowed for capturing temporal dependencies in opcode sequences, resulting in a significant improvement in detection capabilities. This work underscored the potential of using opcode sequences in conjunction with RNN, LSTM, and GRU for robust and accurate malware detection, while also highlighting the importance of further exploring additional features for comprehensive classification.

Corresponding Author:

Annemneedi Lakshmanarao
Department of Information Technology, Aditya University
Surampalem, India
Email: laxman1216@gmail.com

1. INTRODUCTION

The rapid proliferation of Android applications has led to an increased risk of malware attacks, posing significant challenges to cybersecurity. Detecting Android malware is a complex task, primarily due to the intricate structure of Android applications, which consist of multiple Java programs and classes. These programs utilize a variety of opcodes-operational codes that serve as the fundamental building blocks of the application's execution. The sequence of opcodes in malicious Android apps often differs from those in benign apps, making opcode sequences a valuable feature for distinguishing between the two.

To accurately classify these applications, it is essential to extract and analyze these opcode sequences effectively. This process requires significant effort, as the structure and behavior of malware can vary widely. To address this, feature selection techniques have been employed to reduce the complexity of the malware detection framework by focusing on the most relevant features, such as opcode sequence. In this context, the construction of a robust malware detection model based on opcode sequences presents a promising approach. By leveraging these sequences, it is possible to identify patterns that are indicative of malicious behavior, thereby improving the accuracy and reliability of detection mechanisms.

Lakshmanarait and Shashi [1] came up with recurrent neural networks (RNN) as an alternative to ANNs. ANNs consider only present inputs and are incapable of remembering previous input values. In this work, several variants of RNN used for android malware detection. The proposed study explores the application of advanced machine learning (ML) and deep learning (DL) algorithms, specifically RNN, long short-term memory (LSTM), Bi-LSTM, and gated recurrent unit (GRU) architectures, to develop a comprehensive malware detection model that can effectively identify and classify Android malware based on opcode sequences.

2. RELATED WORKS

Dickey *et al.* [2] used ML to identify Android malware instead of signatures. The authors processed malware binary characteristics using a convolutional neural network (CNN) and tree-based models. They were 87%-90% accurate. The research addressed overfitting, notably in tree-based models, and showed that models without overfitting performed consistently throughout training and testing. Fatima and Khan [3] developed Android malware prediction algorithms utilizing various app permissions database. XGBoost with gradient boosting classifier ensemble learning yielded 81.47% accuracy. Ensemble learning captured complicated Android app behaviors better than many other techniques, including DL. In addition to DL, the research showed how ML may improve mobile security. Vanusha *et al.* [4] employed static Android APK attributes to distinguish clean from malicious apps. The study trained and evaluated five ML models using the Drebin-215 dataset: decision tree (DT), support vector machine (SVM) with radial basis function (RBF) kernel, logistic regression (LR), k-nearest neighbor (KNN), and SecuDroid neural network. Static feature-based malware detection worked well with the SecuDroid neural network. Gu and Du [5] presented multimodal neural networks and static analysis for Android malware detection. This method retrieved permissions, opcodes, and API call sequences using pseudo-dynamic and static program analyzers. Through multimodal neural networks, these varied features were fused and classified to improve detection accuracy. The MalMem dataset showed that our strategy outperformed previous approaches in detection.

Udayakumar *et al.* [6] used global image shape transform (GIST) characteristics from grayscale application pictures to identify Android malware. The virus sharing website included malware and benign program samples. To represent the applications' global spatial architecture, GIST characteristics were retrieved from grayscale photos. The apps were classified using LR, KNN, and AdaBoost. Malware identification was also improved using a feed-forward neural network (FFNN) over standard classifiers. Odat and Yaseen [7] proposed permissions and API calls for Android malware detection using ML. The model revealed that malware seeks unusual combinations of these traits compared to harmless programs. From a new dataset of permissions and API requests at different levels, the FP-growth algorithm selected the most essential co-existing properties. For Android malware categorization, random forest (RF) outperformed other traditional ML algorithms. On the Malgenome and Drebin datasets, state-of-the-art methods were less accurate. Awais *et al.* [8] developed the ANTI-ANT framework to identify and prevent Android malware. It extracted features using static and dynamic analysis and three-layer detection. SVM and logistic regressor were used for classification. The architecture was accurate on the CCCS-CIC-AndMal-2020 dataset, with SVM performing best. Mahindru *et al.* [9] introduced "YarowskyDroid," a semi-supervised ML and federated learning method to identify malware-infected applications while protecting user privacy. Locally installed apps on cellphones collected data to enhance the detecting algorithm. On 50,000 malware-free and 25,000 malicious program downloads, the framework showed good detection rates with federated learning across different users. Subash *et al.* [10] used static permissions and ML to identify Android malware. The Android API use study found suspected malicious activities in 398 Android apps. After preprocessing, naive bayes, decision tree, and k-neighbors were compared.

Baghirov [11] tested ML techniques for Android malware detection using benign and dangerous applications. The algorithms' accuracy, precision, recall, and F1-score were evaluated. LightGBM performed best across all criteria, indicating it might be used for Android malware detection. Chowdhury *et al.* [12] provided a comprehensive review of Android malware detection techniques using ML. It covered various supervised, unsupervised, and DL approaches, compared their performance, and discussed the metrics used for evaluating their effectiveness. Lakshmanarao *et al.* [13] tested ML techniques for AMD using a dataset of benign and dangerous applications. The sticking was evaluated on accuracy, precision, recall, and F1-score.

Two types of stacking namely blending and stacking applied and reported good results. Doğanay and Bülbül [14] employed ML to identify Android malware using the Drebin dataset's extensive static and dynamic properties. The dataset was reduced to manifest file permissions for speedier detection. ML algorithms included RF, naive bayes, J48, and AdaBoost. Android malware detection was better using the RF method. Sharma and Sangal [15] used ML to identify Android malware on the CICInvesAndMal2019 dataset, which focuses on permissions and intents. PCA selected features. The dataset was analyzed using Naive Bayes (NB), decision tree classifier (DTC), RF, and KNN. RF was the most successful binary and malware category classifier. Smmarwar *et al*. [16] suggested XAI-AMD-DL, an explainable AI-based hybrid model for Android malware detection, using CNNs and Bi-GRUs. Research tackled the important problem of increasing DL model interpretability while retaining high detection accuracy. The XAI-AMD-DL model outperformed conventional DL approaches. Lakshmanarao and Shashi [17] addressed the shortcomings of signature-based malware detection, notably against advanced Android malware obfuscation. The authors presented a framework to extract Android app permissions, opcodes, API packages, system calls, intents, and API calls. RF was initially the most accurate classifier. The work used multilayer autoencoders for feature extraction and a RF classifier to improve detection accuracy. Real-world datasets showed that this integrated technique can detect Android malware with excellent accuracy. Salah *et al*. [18] addressed the increased need for automated malware detection in Android applications due to mobile phone use and privacy and security concerns. Analysis of program permissions identified static malware. A large application dataset was used to calculate permissions. The study classified these features using tree-based ML.

Guyton *et al*. [19] considered Android malware detection feature selection, a key but often overlooked factor. It evaluated 11 feature selection approaches on three Android feature sets-permissions, intents, and API calls using ML classifiers. Gupta and Anne [20] compared ML malware detection technologies to conventional methods. It assessed three ML models for harmful software detection and described their accuracy and efficiency. Lee *et al*. [21] examined Android malware detection feature selection using genetic algorithms. Genetic algorithm-based feature selection enhanced malware detection performance and time efficiency. Mantoro *et al*. [22] used dynamic analysis in the mobile security framework to identify Android malware, especially obfuscators. A percentage of malware samples were identified using dynamic analysis. Though successful, the solution suffered hardware restrictions and emulator application behavior unpredictability. To overcome dataset quality, Wang *et al*. [23] presented selective ensemble learning for Android malware detection. The evolutionary algorithm selects the top component learners, making the model more resilient to weak training data. The findings showed that the suggested Android malware detection approach was more resilient and effective. Han *et al*. [24] used API calls as characteristics to identify fraudulent Android apps in a large, sparse dataset. A large dataset of Android apps and features was employed. Using SVM, a machine-learning strategy for malicious application detection performed competitively. Jhasi *et al*. [25] examined Android malware, specifically from apps that ask users for rights they may unintentionally authorize. The research used ML to find the most important permissions for categorizing malware and benign apps.

3. METHOD

The proposed methodology is shown in Figure 1. The proposed model for android malware detection using LSTM from opcode sequences was shown in Figure 1. Android apks are collected (both malware and non-malware) and opcode sequences are extracted from android apps. Smali files were created from classes.dex files. Details about opcodes is obtained from smali files. To extract the features, a python utility called "Androguard" was utilized. Androguard comes with a number of instructions for working with Android apps. This tool can be installed using pip. It is also available in Ubuntu/Debian. It can be easily installed using apt command in ubuntu. It can also be directly installed through git. There several commands available in androguard for doing several operations with android applications. "androguard decompile" generates control flow graphs (CFG) for the specified android app. It also creates.ag files (smali-like form) for all of the decompiled classes and methods. Opcode sequences are extracted using the.ag files. After getting a sequence of opcodes, various variants of RNN namely LSTM, Bi-LSTM, GRU are applied to these sequences for detection of android malware.

3.1. Data collection

Malware applications are gathered from the website "virusshare.com". Apks that do not include malicious software have been selected from apkpure.com and Play Store. In the collected apks, 1000 malware apps and 1000 benign apps are used in this experiment.

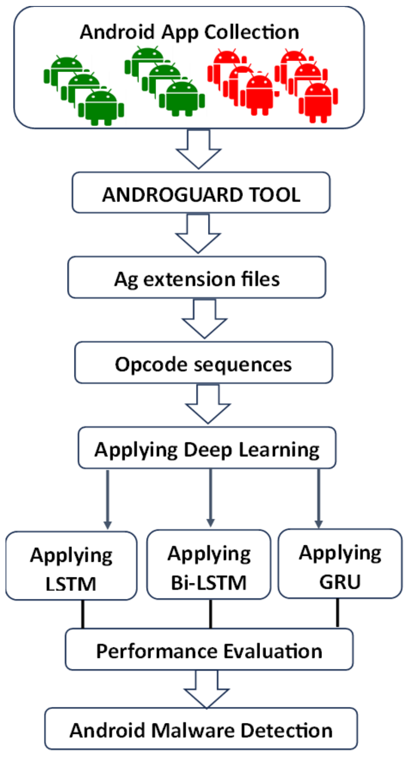


Figure 1. Proposed methodology for android malware detection

3.2. Creation of control flow graph

A CFG, is a diagram that shows how control move through an application as it runs. The data flow routes in a CFG are represented by edges, while the basic blocks are represented by nodes. The CFG illustrates all of the possible directions that might be taken while a programmed is being run. So, it is important to analyze CFGs for differentiating malware and benign apks. There is a command in androguard for creating CFGs for android apks.

3.3. Opcode sequence extraction from android apps

Android app generates multiple files, including the classes.dex file, the manifests file, the assets file, and the reference files. The “classes.dex” contains the Java code for the Android app. Smali files are retrieved after decompiling dex files. The number of opcode sequences varies from one app to another app. The decompile command creates files with an ag extension along with CFGs. One app can produce multiple ag files and each of these files is associated with a specific method. Opcodes sequences are extracted from these ag files. Figure 2 shows sample CFG and Figure 3 shows sample ag file for the same file. The opcodes used in this CFG are “const/4”, “invoke-virtual”. “Invoke-result”, “if-nez”, and “invoke-virtual”.

The extraction of opcode sequences from ag files are done with below process. The process of generating opcode sequences from an APK begins with the APK file, alongside a Dalvik opcode list and an initially empty opcode sequence list. Using the Androguard tool, the APK is decompiled to produce an output folder containing various files, including CFGs and .ag files. Each .ag file is then processed by reading its contents line by line, comparing each line with the Dalvik opcode list, and adding matching opcodes to the opcode sequence list. Finally, any opcode sequence with fewer than 15 entries is filtered out to ensure only significant sequences are retained for further analysis. From this method, it is observed that it creates several CFGs along with ag files in an output folder. All the output folders are parsed to extract opcode sequences from ag files.

After applying algorithm, a list of lists with opcodes is created. Later, all these opcode sequences are transformed to csv file using python script. The python script produces n csv files (here n is number of apks) for all applications using a loop structure. Each row of csv file contains opcode sequences for one application. After this step, all the opcode sequences are available in excel file.

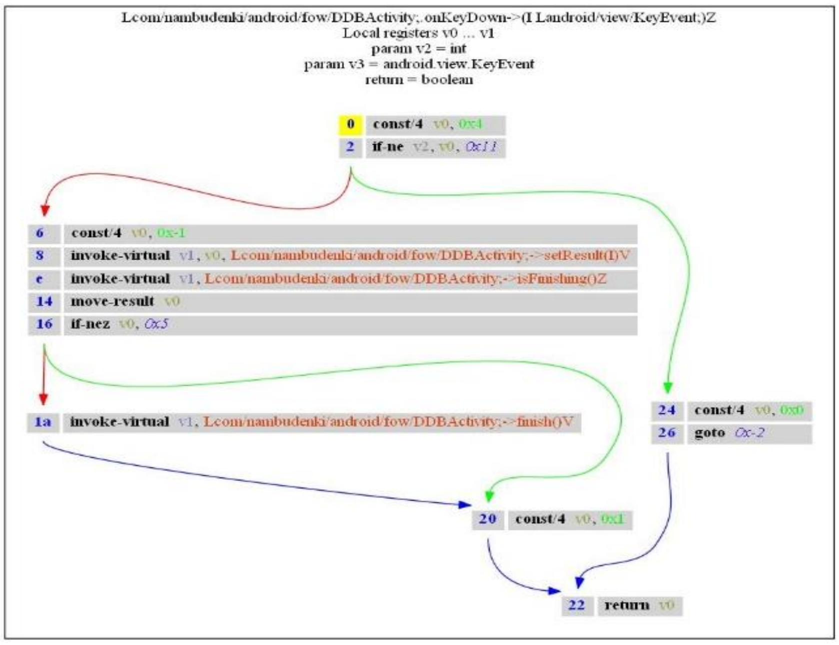
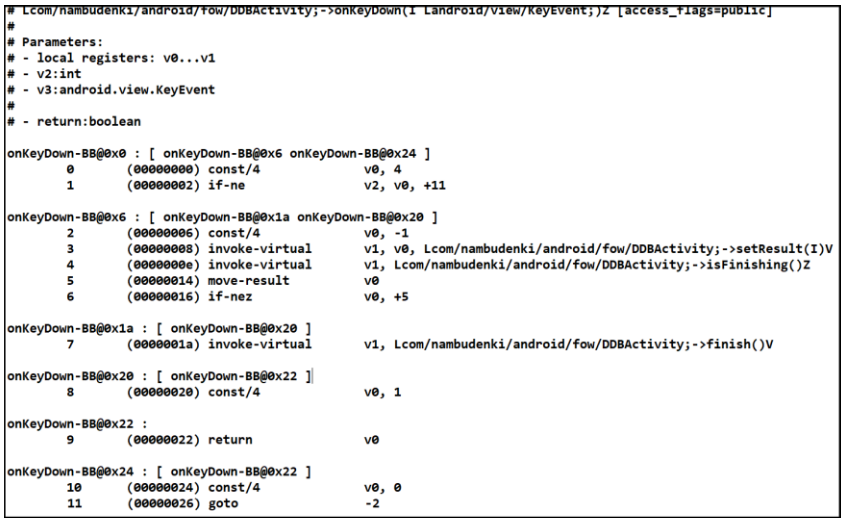

Lcom/nambudenki/android/fow/DDBActivity;.onKeyDown->(I Landroid/view/KeyEvent;)Z
Local registers v0 ... v1
param v2 = int
param v3 = android.view.KeyEvent
return = boolean
0 const/4 v0, 0x4
2 if-ne v2, v0, 0x11
6 const/4 v0, 0x-1
8 invoke-virtual v1, v0, Lcom/nambudenki/android/fow/DDBActivity;->setResult(I)V
e invoke-virtual v1, Lcom/nambudenki/android/fow/DDBActivity;->isFinishing()Z
14 move-result v0
16 if-nez v0, 0x5
1a invoke-virtual v1, Lcom/nambudenki/android/fow/DDBActivity;->finish()V
24 const/4 v0, 0x0
26 goto 0x-2
20 const/4 v0, 0x1
22 return v0

Figure 2. Sample CFG



```
# Lcom/nambudenki/android/fow/DDBActivity;->onKeyDown(I Landroid/view/KeyEvent;)Z [access_flags=public]
#
# Parameters:
# - local registers: v0...v1
# - v2:int
# - v3:android.view.KeyEvent
#
# - return:boolean

onKeyDown-BB@0x0 : [ onKeyDown-BB@0x6 onKeyDown-BB@0x24 ]
        0       (00000000) const/4              v0, 4
        1       (00000002) if-ne                v2, v0, +11

onKeyDown-BB@0x6 : [ onKeyDown-BB@0x1a onKeyDown-BB@0x20 ]
        2       (00000006) const/4              v0, -1
        3       (00000008) invoke-virtual       v1, v0, Lcom/nambudenki/android/fow/DDBActivity;->setResult(I)V
        4       (0000000e) invoke-virtual       v1, Lcom/nambudenki/android/fow/DDBActivity;->isFinishing()Z
        5       (00000014) move-result          v0
        6       (00000016) if-nez               v0, +5

onKeyDown-BB@0x1a : [ onKeyDown-BB@0x20 ]
        7       (0000001a) invoke-virtual       v1, Lcom/nambudenki/android/fow/DDBActivity;->finish()V

onKeyDown-BB@0x20 : [ onKeyDown-BB@0x22 ]
        8       (00000020) const/4              v0, 1

onKeyDown-BB@0x22 :
        9       (00000022) return               v0

onKeyDown-BB@0x24 : [ onKeyDown-BB@0x22 ]
        10      (00000024) const/4              v0, 0
        11      (00000026) goto                 -2
```

Figure 3. Sample .ag file

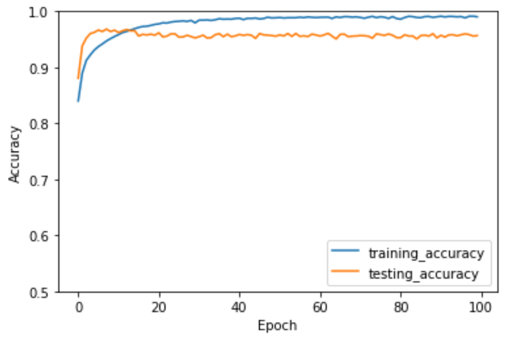
4. RESULTS AND DISCUSSION

4.1. Applying RNN

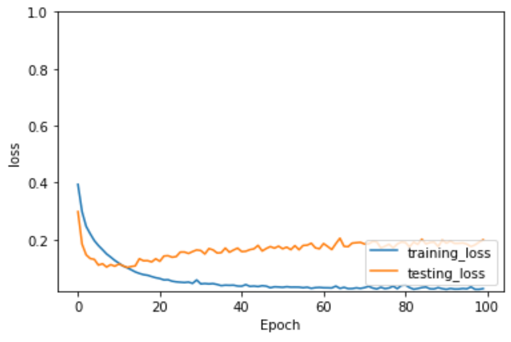
In the first experiment, we applied a RNN to the opcode sequences extracted for Android malware detection. The RNN model was designed with three hidden layers consisting of 200, 100, and 5 neurons, respectively. The training process involved a batch size of 64, a learning rate of 0.001, and was conducted over 100 epochs. While the RNN effectively captured temporal dependencies in the opcode sequences, it struggled with retaining long-term dependencies, leading to an overall accuracy of 87%. The vanishing gradient problem inherent in RNNs likely contributed to this moderate performance, particularly when handling longer sequences.

4.2. Applying LSTM

To improve on the limitations observed with the RNN, a LSTM network was implemented using the same three hidden layers (200, 100, and 5 neurons) and trained over 100 epochs. LSTM networks are designed to better manage long-term dependencies through their internal gating mechanisms, which address the vanishing gradient issue. This model achieved a significantly higher accuracy of 96%, demonstrating its effectiveness in differentiating between malicious and benign opcode sequences. The LSTM's ability to preserve information over long sequences was key to its superior performance compared to the RNN. Figure 4 shows epoch wise performance of LSTM. Figure 4(a) shows epoch wise accuracies and Figure 4(b) shows epoch wise loss values with LSTM. Later, Bi-LSTM also applied and achieved accuracy of 96.2%



(a)



(b)

Figure 4. Epoch wise (a) accuracy with LSTM and (b) loss with LSTM

4.3. Applying GRU

Further experimentation was conducted using a GRU network, which is a variant of the LSTM that simplifies the gating mechanisms while maintaining the ability to handle long-term dependencies. The GRU model was configured with the same three hidden layers, consisting of 200, 100, and 5 neurons. The model was trained under the same conditions as the LSTM. The GRU demonstrated comparable performance to the LSTM, with a slight improvement in training efficiency due to its simpler architecture. The GRU achieved high accuracy, successfully capturing the temporal patterns in the opcode sequences and effectively distinguishing between malware and benign applications. The results indicate that GRU is a viable alternative to LSTM, offering a good balance between accuracy and computational efficiency.

4.4. Comparison of RNN variants for malware detection

The standard RNN achieved an accuracy of 87%, which is lower compared to the LSTM and GRU. The LSTM model performed the best, reaching an accuracy of 96%, Bi-LSTM given accuracy of 95.6%. The GRU model, while slightly less accurate than LSTM at 93%, offered faster training times and good overall performance. The comparison highlights LSTM as the most effective, with GRU as a strong alternative when computational efficiency is important. Figure 5 shows accuracy comparison of RNN variants for malware detection.

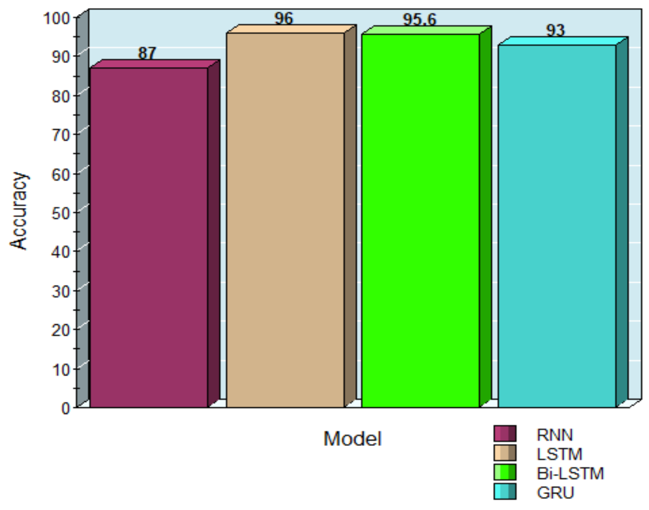


Figure 5. Comparison of deep learning RNN variants for malware detection

5. CONCLUSION

This research demonstrated the effectiveness of using opcode sequences for Android malware detection through advanced DL models, specifically RNN, LSTM, and GRU. The study showed that LSTM achieved the highest accuracy of 96%, highlighting its superior ability to capture long-term dependencies in the data. Bi-LSTM produced a good accuracy of 95.6% for malware detection. GRU, with an accuracy of 93%, proved to be a strong alternative, offering a good trade-off between performance, and computational efficiency. The standard RNN, while effective, lagged behind with an accuracy of 87%. Overall, this work underscores the potential of DL techniques, particularly LSTM, and GRU, in enhancing malware detection capabilities. The results suggest that further exploration of these models, possibly incorporating additional features, could lead to even more robust and accurate malware detection systems.

REFERENCES

[1] A. Lakshmanarao and M. Shashi, "Android Malware Detection with Deep Learning using RNN from Opcode Sequences," I*nternational Journal of Interactive Mobile Technologies (iJIM)*, vol. 16, no. 01. International Association of Online Engineering (IAOE), pp. 145–157, Jan. 18, 2022. doi: 10.3991/ijim.v16i01.26433.

[2] K. Dickey, D. Hwang, and D. Kim, "Analyzing various machine learning approaches for detecting Android malware," *SoutheastCon 2024*, Atlanta, GA, USA, 2024, pp. 1288-1293, doi: 10.1109/southeastcon52093.2024.10500178.

[3] N. Fatima and H. F. Khan, "A comprehensive analysis and evaluation of Android malware prediction using AI," *2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS)*, Manama, Bahrain, 2024, pp. 1-5, doi: 10.1109/ICETSIS61505.2024.10459543.

[4] D. Vanusha, S. Singh, A. B. Jha, and D. R. S, "SecuDroid: Android malware detection using ML classifier on static features," *2024 2nd International Conference on Networking and Communications (ICNWC)*, Chennai, India, 2024, pp. 1-9, doi: 10.1109/ICNWC60771.2024.10537417.

[5] F. Gu and Z. Du, "Multimodal neural network based malware detection for Android," *2024 2nd International Conference On Mobile Internet, Cloud Computing and Information Security (MICCIS)*, Changsha City, China, 2024, pp. 63-67, doi: 10.1109/MICCIS63508.2024.00019.

[6] P. Udayakumar, S. Yalamati, L. Mohan, M. J. Haque, G. Narkhede, and K. M. Bhashyam, "Android malware detection using GIST based machine learning and deep learning techniques," *Indonesian Journal of Electrical Engineering and Computer Science, vol. 35, no. 2. Institute of Advanced Engineering and Science*, vol. 35, no. 2, pp. 1244-1252, 2024, doi: 10.11591/ijeecs.v35.i2.pp1244-1252.

[7] E. Odat and Q. M. Yaseen, "A novel machine learning approach for Android malware detection based on the co-existence of features," *IEEE Access*, vol. 11, pp. 15471-15484, 2023, doi: 10.1109/ACCESS.2023.3244656.

[8] M. Awais, M. A. Tariq, J. Iqbal, and Y. Masood, "Anti-ant framework for Android malware detection and prevention using supervised learning," *2023 4th International Conference on Advancements in Computational Sciences (ICACS)*, Lahore, Pakistan, 2023, pp. 1-5, doi: 10.1109/ICACS55311.2023.10089629.

[9] A. Mahindru, S. K. Sharma, and M. Mittal, "YarowskyDroid: semi-supervised based Android malware detection using federation learning," 2*023 International Conference on Advancement in Computation & Computer Technologies (InCACCT)*, Gharuan, India, 2023, pp. 380-385, doi: 10.1109/InCACCT57535.2023.10141735.

[10] A. Subash, R. S. Shane, G. Vijay, G. E. Selvan, and M. P. Ramkumar, "Malware detection in Android application using static permission," *In 2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 1241-1245, 2023, doi: 10.1109/ICIRCA57980.2023.10220934.

[11] E. Baghirov, "Evaluating the performance of different machine learning algorithms for android malware detection," *2023 5th International Conference on Problems of Cybernetics and Informatics (PCI)*, Baku, Azerbaijan, 2023, pp. 1-4, doi: 10.1109/PCI60110.2023.10326006.

[12] M. N.-U.-R. Chowdhury, A. Haque, H. Soliman, M. S. Hossen, T. Fatima, and I. Ahmed, "Android malware detection using machine learning: a review," *arXiv*, 2023, doi: 10.48550/ARXIV.2307.02412.

[13] A. Lakshmanarao et al.,"An efficient android malware detection framework with stacking ensemble model," *International Journal of Engineering Trends and Technology*, vol. 70, no. 4. Seventh Sense Research Group Journals, pp. 294–302, Apr. 25, 2022. doi: 10.14445/22315381/ijett-v70i4p22.

[14] H. A. Doğanay and H. İ. Bülbül, "Detection success assessment of machine learning algorithms through manifest file permissions demanded by malicious android wares," 2*023 International Conference on Machine Learning and Applications (ICMLA)*, Jacksonville, FL, USA, 2023, pp. 1684-1686, doi: 10.1109/ICMLA58977.2023.00254.

[15] N. Sharma and A. L. Sangal, "Machine learning approaches for analysing static features in Android malware detection," *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, Jalandhar, India, 2023, pp. 93-96, doi: 10.1109/ICSCCC58608.2023.10176445.

[16] S. K. Smmarwar, G. P. Gupta and S. Kumar, "XAI-AMD-DL: an explainable AI approach for android malware detection system using deep learning," *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)*, Sonbhadra, India, 2023, pp. 423-428, https://ieeexplore.ieee.org/document/10263974.

[17] A. Lakshmanarao and M. Shashi, "Android malware detection using multilayer autoencoder and random forest," *International Journal of Engineering Trends and Technology*, vol. 70, no. 11, pp. 249-257, 2022, doi: 10.14445/22315381/IJETT-V70I11P227.

[18] A. T. Salah, M. A. Hassan, M. I. Abbas, Y. H. Sayed, Z. M. Elsahaer, and G. A. Khoriba, "Android static malware detection using tree-based machine learning approaches," *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, Cairo, Egypt, 2022, pp. 3-10, doi: 10.1109/MIUCC55081.2022.9781765.

[19] F. Guyton, W. Li, L. Wang, and A. Kumar, "Analysis of feature selection techniques for android malware detection," *SoutheastCon 2022*, Mobile, AL, USA, 2022, pp. 96-103, doi: 10.1109/SoutheastCon48659.2022.9764071.

[20] M. Gupta and S. V. N. S. Anne, “Predicting malicious activity in Android using machine learning,” *2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, Greater Noida, India, 2022, pp. 121-124, doi: 10.1109/CISES54857.2022.9844358.
[21] J. Lee, H. Jang, S. Ha, and Y. Yoon, “Android malware detection using machine learning with feature selection based on the genetic algorithm,” *Mathematics*, vol. 9, no. 21, p. 2813, 2021, doi: 10.3390/math9212813.
[22] T. Mantoro, D. Stephen, and W. Wandy, “Malware detection with obfuscation techniques on Android using dynamic analysis,” *2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED)*, Sukabumi, Indonesia, 2022, pp. 1-6, doi: 10.1109/ICCED56140.2022.10010359.
[23] J. Wang, Q. Jing, J. Gao, and X. Qiu, “SEdroid: a robust Android malware detector using selective ensemble learning,” *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, Seoul, Korea (South), 2020, pp. 1-5, doi: 10.1109/WCNC45663.2020.9120537.
[24] H. Han, S. Lim, K. Suh, S. Park, S. -j. Cho, and M. Park, “Enhanced Android malware detection: an SVM-based machine learning approach,” *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Busan, Korea (South), 2020, pp. 75-81, doi: 10.1109/BigComp48618.2020.00-96.
[25] K. S. Jhasi, S. Chakravarty, and R. K. Varma P., “Feature selection and evaluation of permission-based android malware detection,” *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184)*, Tirunelveli, India, 2020, pp. 795-799, doi: 10.1109/ICOEI48184.2020.9142929.

BIOGRAPHIES OF AUTHORS

Dr. Annemneedi Lakshmanarao currently working as assistant professor in Aditya University, Surampalem. He completed his B. Tech in CSIT and M. Tech in Software Engineering. He Completed Ph.D. in Andhra University, Vishakhapatnam. His areas of interest are machine learning, cyber security, and deep learning. He is a life member of Computer Society of India (CSI) and ISTE. He can be contacted at email: laxman1216@gmail.com.

Jeevana Sujitha Mantena Jeevana Sujitha Mantena is working as an assistant professor, Dept. of CSE in SRKR Engineering College(A), Bhimavaram, Andhra Pradesh. India. She is Perusing Ph.D in KLUniversity ,Vijayawada in the area of Software Engineering and Machine Learning. She has more than 12 years of teaching experience. She had published papers in reputed National and International Journals. She had attended many workshops, conferences and presented various research papers at National and International conferences. His areas of interest software engineering, artificial intelligence, and machine learning. She can be contacted at email: jeevanasujitha@gmail.com.

Krishna Kishore Thota Mr. Krishna Kishore Thota, working as an assistant professor in the Department of Computer Science and Engineering (Honors), Koneru Lakshmaiah Education Foundation (Deemed to be University), Vaddeswaram. He received his M.Tech. (CSE) degree from JNT University, Kakinada in 2010 and B.Tech. (CSE) degree from JNT University, Hyderabad in 2005. He is Pursuing Ph.D. (CSE) in Sathyabama Institute of Science and Technology (Deemed to be University), Chennai. He has more than 17 years’ experience in teaching and ratified as an assistant professor by JNT University, Kakinada. His areas of interest include machine learning, principles of compiler design, artificial neural networks and deep learning, network security, and theory of computation. He has presented around 12 research papers in International Conferences, published around 10 research articles from his research finding in various reputed International Journals and 2 Indian patents. He has been an active member of several professional societies like ISTE, CSI, IAENG. He has received awards as best teacher, best academician and best researcher for his academic and research work. He can be contacted at email: tkrishnakishore@kluniversity.in.

Dr. Pavan Sathish Chandaka Pavan Sathish Chandaka working in the Department of CSE at Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh, India. He is having 9 years of academic experience and 5 years of research expertise. His work primarily focuses on medical image segmentation using machine and deep learning techniques. He has published several articles in reputed journals. He can be contacted at email: pavansatishch@gmail.com.

Chinta Venkata Murali Krishna currently working as associate professor and HOD in CSE (Data Science) department at NRI Institute of Technology. He is a member of IAENG, IFERP, and INSC. He completed his M.Tech. in Computer Science and Engineering in 2009 and is currently pursuing a Ph.D. in Computer Science and Engineering at GITAM (Deemed to be University), Visakhapatnam. He has published research papers in various conferences and journals and has been granted three patents with ten others in the pipeline for the grant. Four of his books have been published by international and national publishing agencies. He was awarded the “Best Researcher Award” from IOSRD in 2018. He can be contacted at email: muralikrishna_chinta2007@yahoo.co.in.

Madhan Kumar Jetty Madhan Kumar Jetty is working as an assistant professor, Dept. of Information Technology in R.V.R and J.C College of Engineering, Guntur, Andhra Pradesh, India. He is pursuing his Ph.D. from JNTU Kakinada in the area of machine learning. He has 10 years of teaching experience. He had published papers in reputed National and International Journals. He had attended many workshops, conferences and presented various research papers at National and International conferences. His areas of interest include image processing, machine learning, deep learning, natural language processing, and cyber security. He can be contacted at email: madhanjetty.rvr@gmail.com.