Int. J. Adv. Res. Sci. Technol. Volume 13, Issue 10, 2024, pp.1510-1515.

IJARST

International Journal of Advanced Research in Science and Technology

https://doi.org/10.62226/ijarst20242513

ISSN 2319 – 1783 (Print)

ISSN 2320 – 1126 (Online)

Detection of Android Malware Using Word2Vec and Deep Learning

Zainab Fatima[1], Sujata Terdal[2]

Department of Computer Science and Engineering, PDA College of Engineering, Kalaburagi, Karnataka, India[1,2]

zainabcs084@gmail.com[1], Sujatha.terdal@gmail.com*[2]*

ARTICLEINFO

Article history:

Received 20 Oct 2024

Accepted 28 Oct 2024

Available online 05 Nov 2024

Keywords:

Android malware,
Convolutional Neural Networks (CNN),
Long Short-Term Memory (LSTM),
Random Forest.

ABSTRACT

To fortify Android devices against evolving threats, the project unfolds with a commitment to contribute to security endeavors significantly. The core purpose is to leverage machine-learning techniques, particularly deep-learning architectures, to construct robust mechanisms for identifying and classifying Android malware. A hybrid deep learning model combining Long Short-Term Memory (LSTM) networks with Convolutional Neural Networks (CNN) is employed to classify the applications as benign or malicious. The LSTM-CNN model is designed to effectively capture both temporal dependencies and spatial patterns within the feature vectors. Additionally, a Random Forest classifier is used to identify the most important features for classification, improving the performance and efficiency of the deep learning model. The proposed approach is evaluated on the CICMalDroid2020 dataset, demonstrating its ability to detect and classify Android malware accurately. This highlights the effectiveness of static feature analysis combined with advanced deep learning architectures for enhancing Android malware detection systems.

Introduction:

In the rapidly expanding world of mobile applications, Android devices have become prime targets for malicious software, commonly known as malware. It is essential to create efficient methods for malware identification and categorization due to the growing complexity of malware attacks. Using deep learning models in conjunction with NLP (natural language processing) methods such as Word2Vec embeddings to detect and classify malware is one potential approach. By treating sequences of API calls or other behaviour patterns in malicious software as words in a sentence, Word2Vec can map these sequences into a continuous vector space that captures semantic relationships. This allows deep learning models to effectively learn patterns and features associated with both known and novel malware variants. Android malware is a danger to mobile device security and privacy, and conventional detection techniques are challenged by the increasing number of malicious applications. The development of models using deep learning in recent years has opened up new possibilities for improving malware detection skills. These models, with their ability to learn complex patterns and representations from raw data, offer promising prospects for accurately identifying and mitigating Android malware threats. In this context, the application of deep learning models for Android malware detection is explored. Use cutting-edge designs like neural networks with recurrent connections (RNNs), CNNs designs, and their variations to create resilient and flexible solutions that can accurately discern between malicious and benign applications. Through the analysis of static and dynamic features extracted from mobile applications, deep learning models can learn to detect subtle indicators of malicious behaviour, including code obfuscation, unauthorized data access, and network communication with malicious servers.

Related Work:

The rapid increase in the use of Android devices has brought along an alarming rise in malware targeting the platform. Traditional detection methods, such as signature-based techniques, have proven ineffective against sophisticated and evolving malware. As a result, machine learning (ML) has emerged as a promising solution for Android malware detection, offering the ability to generalize and detect novel malware by analyzing behavioral and structural patterns. This review examines the advances in ML-based Android malware detection, highlighting the use of feature extraction methods, machine learning algorithms, adversarial attacks, and challenges faced in implementing these techniques.

The effectiveness of ML-based Android malware detection largely depends on the quality of the extracted features. Static analysis, which examines the code structure without executing the app, remains popular due to its scalability and speed. [1] has

provided a comprehensive review of feature selection in mobile malware detection, highlighting static features such as permissions and API calls as important indicators of malicious behavior. [2] employed static features, including permissions and API calls, to build effective ML classifiers for malware detection. However, despite its efficiency, static analysis is often vulnerable to obfuscation techniques used by attackers to disguise malicious code.

Dynamic analysis, which involves monitoring an app's runtime behavior, offers a more robust solution by capturing features such as system calls and network traffic. [3] has introduced **MaMaDroid**, a system that analyzes the sequences of API calls to build behavioral models for detecting malware. This approach proved effective against obfuscated malware that alters its code structure without changing its behavior. However, dynamic analysis can be resource intensive and difficult to apply in real-time. To combine the strengths of both methods, hybrid approaches have emerged. For instance, [4] has combined static and dynamic analysis to improve malware detection accuracy, though hybrid methods often require higher computational resources, limiting their feasibility in real-time scenarios.

Traditional ML algorithms such as Decision Trees, Support Vector Machines (SVM), Random Forests, and Naive Bayes have been widely adopted in early Android malware detection studies. [1] has noted the effectiveness of traditional models like Random Forests and SVM in detecting malware based on static permissions and API calls. [8] has similarly utilized Random Forests to classify Android applications as either benign or malicious, achieving satisfactory results. While these models are easy to interpret and relatively fast to train, they often struggle with high-dimensional data and the complex behavioral patterns exhibited by modern malware.

Methodology:

The LSTM layers are made to record the temporal relationships present in the dataset; they are initially used by the LSTM-CNN model to process the sequential data. These layers help the model retain memory over long sequences, allowing it to grasp the sequential patterns typical of malware activities within the CICMaldroid2020 dataset. Afterward, the features extracted by the LSTM layers are further refined by convolutional layers, which are particularly effective at identifying spatial patterns within the feature space.

Input Layer:

- The input layer accepts sequences of feature vectors representing malware samples. Each feature vector is represented as a single timestep.
- Shape: (number of samples, number of timesteps, 1) LSTM Layer:
- The LSTM (Long Short-Term Memory) layer processes the sequential input data and returns sequences for subsequent layers.
- Number of units: Tunable between 32 to 256 (Random Search)
- Return Sequences: True that is it will output all the hidden states of each time
- steps.

Convolutional Layer:

- The Conv1D layer applies convolutional operations to the output sequences from the LSTM layer to extract spatial features.
- Number of filters: Tunable between 16 to 64 (Random Search)
- Kernel Size: Tunable between 3 to 5 (Random Search)
- Activation: ReLU Flatten Layer:
- The Flatten layer flattens the output from the convolutional layer that is used to make the multidimensional input one-dimensional to prepare it for the dense layers.

Output Layer:

- The output layer produces the final classification output, with each unit representing the probability of a particular class.
- Units: 5 (Assuming it's a multi-class classification problem)
- Activation: Sigmoid Training Process.

The input data, both training and testing sets are reshaped to match the input requirements of the LSTM-CNN architecture. [9] The framework is compiled using the accuracy metric, categorical a cross-en loss function, and optimizer (Adam or the RMSprop) that are supplied. The training data is used to train the model. Hyperparameters are tuned via Random Search, optimizing for validation accuracy. Training is performed for a specified number of epochs (100) to iteratively update the network parameters and optimize performance with a defined batch size (512) to efficiently process data in mini batches. The test data is used to assess the model's performance following training. A few classification measures are calculated, including ROC AUC score, MCC, F1 score, accuracy, precision, recall, and specificity. Visual performance evaluation of the model is done through the generation of a confusion matrix and classification report. This comprehensive training process and model architecture allow the LSTM-CNN model to effectively capture both sequential and spatial features from the input data, making it suitable for malware detection and classification tasks.

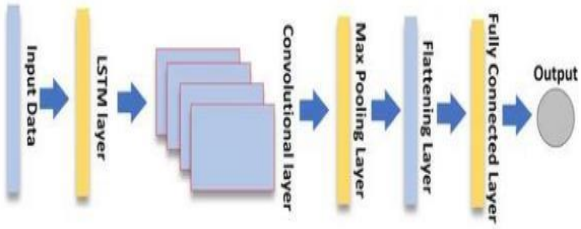

Figure 1: LSTM-CNN Model.

System Design

The following section breaks down the components and workflow of the design, highlighting how each step contributes to the final classification outcome.

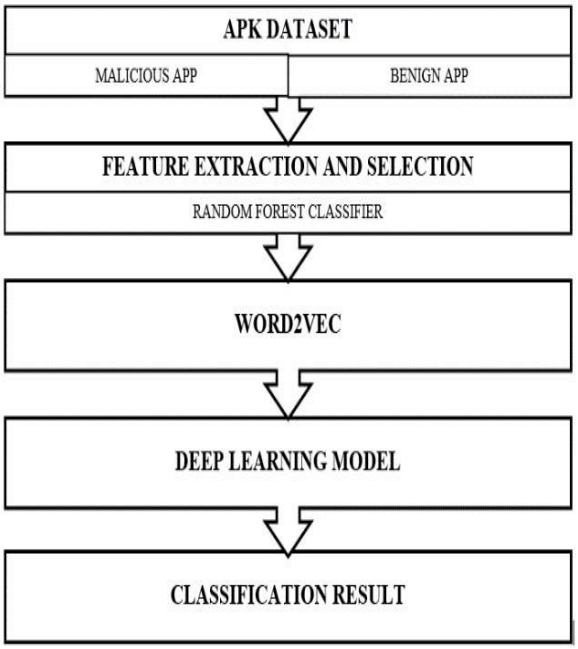

Figure 2: System Design.

To categorize Android applications as dangerous or benign, we provide a hybrid strategy in this research that combines modern methods from deep learning and machine learning with static analysis. The process follows a systematic approach involving feature extraction, feature selection, Word2Vec embedding, and the use of a deep learning model to achieve effective classification.

APK Dataset and Initial Classification

- APK Dataset: The dataset comprises APK files (Android applications) which include both malicious and benign apps. These applications serve as the input for the classification system.
- Malicious App: These apps are designed to harm the device or steal sensitive information. They include various types of malware, such as Trojans, spyware, ransomware, etc.
- Benign App: These are legitimate apps that perform as expected without any malicious intent or behavior.

Feature Extraction and Selection

- Feature Extraction: In this phase, static features from the APK files are extracted. These features include system calls, permissions, API calls, and other relevant characteristics of the Android apps. Examining the APK's code and structure without running it is known as static analysis. Although disguised or packaged malware can be a hindrance to this method, it is effective for large-scale investigation.
- Feature Selection via Random Forest Classifier: Following feature extraction, feature selection is carried out to determine which features are most important for categorizing the apps as benign or dangerous. A Random Forest Separator is used in this instance. During training, the algorithm known as Random Forest generates several decision trees, making it a potent machine learning model. By evaluating each feature's contribution to lowering the classification error, it determines which features are most significant.

The selected features are those that provide the highest discriminatory power in identifying malicious and benign apps. This reduces the feature space and optimizes the performance of subsequent models.

Word2Vec Embedding

- One of the critical steps in this project is the transformation of the selected features into a format suitable for a deep learning model. This transformation is accomplished using **Word2Vec embedding**, a powerful technique developed to capture semantic relationships between words and apply them to sequences of discrete features extracted from Android applications.
- Word2Vec Working: Word2Vec is a two-layer neural network that transforms words, or in this case, features, into vectors of real numbers in a continuous vector space. Unlike traditional methods that represent words or features as one-hot encoded vectors (where each word is a distinct dimension in a sparse vector), Word2Vec creates dense, lower-dimensional embeddings that preserve semantic relationships between features. The key concept behind Word2Vec is that similar words appear in similar contexts. By learning from the sequences of features (like system calls, permissions, and API calls) that occur together in the APK files, Word2Vec places similar features close to one another in the embedding space. This proximity implies that if certain features commonly appear together in both malicious and benign apps, their embeddings will reflect that relationship.

Deep Learning Model

The core of this project lies in using a deep learning model for classification. Specifically, a hybrid LSTM-CNN model is employed:

- LSTM (Long Short-Term Memory): Recurrent neural networks (RNNs) of the LSTM type function especially well for learning from data sequences. In the context of this project, LSTM is used to capture temporal dependencies between sequences of features extracted from the APK files. This is crucial for modeling complex relationships between different aspects of an app's behavior.

- CNN (Convolutional Neural Network): CNNs are designed to capture spatial patterns in data. Here, CNNs are employed to detect local feature patterns within the Word2Vec embeddings. These patterns can represent common behavioral traits of malicious or benign apps, allowing the model to learn the structural properties of the feature vectors.
- Hybrid LSTM-CNN Model: The combination of LSTM and CNN allows the model to leverage both temporal dependencies and spatial patterns, improving the overall performance of the classifier. The LSTM layers process sequences of feature vectors, while the CNN layers capture local patterns within these sequences. The combined output is passed through dense layers to produce a classification result.

Classification Result

A deep learning algorithm can identify an app's class based on its attributes once it has been trained. The categorizing result, which indicates whether a program is malicious or benign, is the model's output.

- Accuracy and Performance: A model's efficacy is assessed using metrics like F1-score, accuracy, precision, and recall. These metrics shed light on the model's ability to discriminate between benign and dangerous apps.
- False Positives/Negatives: The accessibility and safety of the categorization system can be seriously impacted by false positives, which are innocuous apps categorized as malicious, and false negatives, which are harmful applications classified as benign.

Result:

The heatmap visualizes the importance or impact of different features (permissions, API calls, etc.) on the classification of Android applications as either malicious or benign within the CICMaldroid2020 dataset.

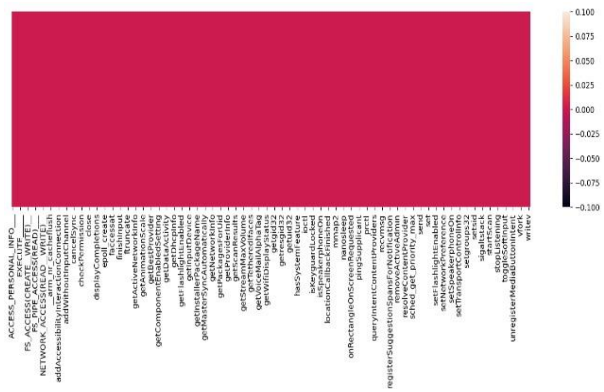

Figure 3: Heatmap of CICMaldroid2020 dataset.

Color Scale:

The color bar on the right in fig 3 indicates the magnitude of feature importance or correlation. Positive values (shades of red) suggest a positive correlation or importance, indicating that these features are more associated with a particular class (e.g., malicious apps). Negative values (darker shades) would suggest a negative correlation or importance, but in this heatmap, it seems all values are either positive or neutral. If the color scale ranged below zero, it would indicate that certain features are more common in benign apps.

X-Axis:

Represents the different features in the dataset (e.g., permissions like ACCESS_FINE_LOCATION, API calls like get System Service).

Interpretation:

A uniformly red heatmap might indicate that all features are relatively equally important in the classification task, or it could suggest that the model or method used to determine feature importance didn't find significant variation among features.

If some features had stood out with higher values, it would suggest they have a stronger influence on the outcome. The bar chart in fig 4 indicates that it represents the number of samples for each class in the dataset. The x-axis lists the classes, numbered from 1 to 5. Each class likely corresponds to a different category of Android applications, such as benign apps or various types of malwares. The number of instances in each class is shown on the y-axis.

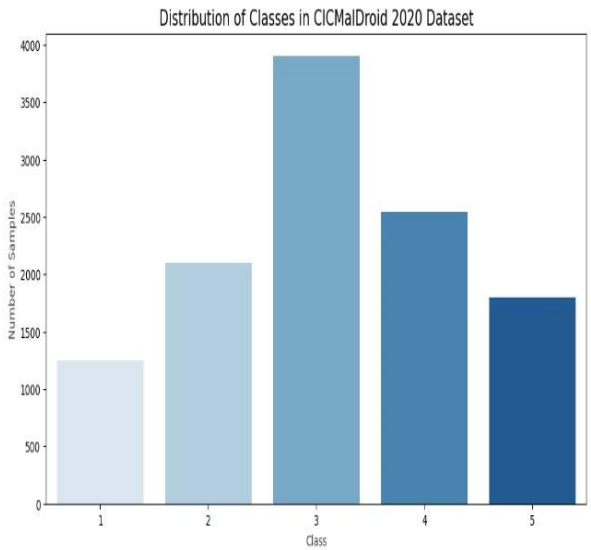

Figure 4: Distribution of Classes in CICMalDroid2020 Dataset.

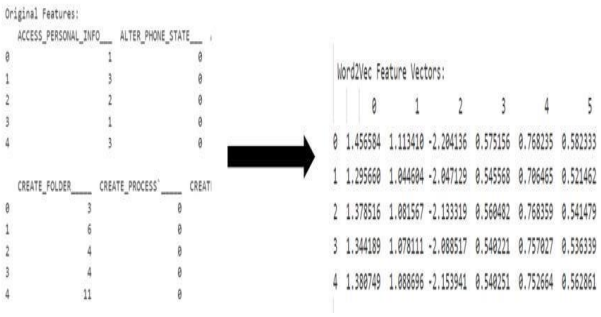

Figure 5: Conversion of original features to word2vec feature vectors.

The fig 5 illustrates the process of converting original features from the CICMaldroid2020 dataset into Word2Vec feature vectors.

Original Features (Left Side): The table shows categorical or numerical values associated with different features like ACCESS_PERSONAL_INFO, ALTER_PHONE_STATE, CREATE_FOLDER, etc.
These are raw, possibly sparse, features representing specific permissions or actions in Android applications.

Word2Vec Feature Vectors (Right Side)
The right side shows the result of applying Word2Vec, a method for converting text-based or categorical features into dense, continuous-valued vectors.
Each row represents a feature vector for a corresponding sample from the original feature table, now transformed into a more compact and informative representation. By capturing the semantic links between features, Word2Vec helps the model identify patterns and similarities in the data.

The conversion helps in reducing dimensionality and improves the ability of models to learn from the data by providing more meaningful representations of features, which is crucial in tasks like malware detection. This transformation is often a preprocessing step before feeding the data into deep learning models, such as your LSTMCNN, to enhance performance.

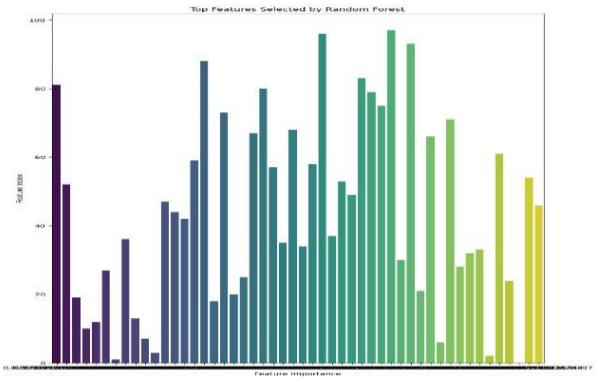

Figure 6: Feature Selection.

The fig 6 depicts the top features selected by the Random Forest algorithm after vectorization in a deep learning model. The x-axis represents the feature importance score, while the y-axis shows the feature index, which corresponds to different features from the dataset after being transformed into numerical vectors.

Figure 7: Confusion Matrix

Confusion matrix, a tool used to evaluate the performance of a classification model by comparing actual and predicted classes shown in fig 7. The matrix displays the results for a multi-class classification task, where each row represents the actual class, and each column represents the predicted class. The confusion matrix analysis demonstrates that the deep learning models achieve varied success across different malware classes. Notably, the highest precision is approximately seventy-nine percent, indicating that the model performs exceptionally well in correctly identifying this particular malware category.

Conclusion:
Conventional methods that depend on dynamic, static, and hybrid assessments are constantly susceptible to malicious software that is constantly changing. Consequently, as the sophistication of malware escalates, deep learning models offer a promising avenue for accurate detection. Studies show that deep learning models such as CNNs, LSTM, Bi-LSTM, CubicLSTM, and hybrid versions are quite effective in spotting dangerous software in Android apps. Notably, Convolutional Neural Networks have emerged as a prominent choice, often achieving superior accuracy compared to alternative detection methods.

References:

[1] Feizollah, A., Anuar, N. B., Salleh, R., & Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. Digital Investigation, 13, 22-37.
[2] Zhang, X., Yang, L., & Gao, Q. (2020). Understanding mobile banking Trojans: A case study on Anubis and Cerberus. Mobile Networks and Applications, 25(1), 207221.
[3] Li, L., Bissyande, T. F., Papadakis, M., & Klein, J. (2017). MaMaDroid: Detecting Android malware by building Markov chains of behavioral models. Proceedings of the ACM

Asia Conference on Computer and Communications Security, 159-170.

[4] Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2017). DL-Droid: Deep learning based Android malware detection using real devices. Proceedings of the 2017 IEEE Conference on Trust, Security and Privacy in Computing and Communications, 244251.

[5] Kang, D., Lee, H., & Im, E. G. (2019). Android malware detection using a multifeature machine learning model. Journal of Information Security and Applications, 44, 23-34.

[6] Zhang, Y., Yang, P., Guo, J., Xue, M., & Liu, C. (2019). A static Android malware detection method based on improved hybrid features. IEEE Access, 7, 104-117.

[7] Wei, F., Roy, S., & Ou, X. (2017). A deeper look at Android runtime permissions. Proceedings of the 2017 IEEE Symposium on Security and Privacy, 42-57.

[8] Zhou, Y., & Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. Proceedings of the 2012 IEEE Symposium on Security and Privacy, 95109.

[9] Mamatha B, Sujatha P Terdal | A SHAP and LIME based Explainable AI Solution for Predicting Chronic Kidney Diseases.