

# MACHINE LEARNING FRAMEWORK FOR REVERSE ENGINEERING ANDROID APPLICATIONS

<sup>#1</sup>RACHARLA RESHMA,

<sup>#2</sup>P.SATHISH, *Assistant Professor,*

<sup>#3</sup>Dr.V.BAPUJI, *Associate Professor & HOD,*

*Department of Master of Computer Applications,*

**VAAGESWARI COLLEGE OF ENGINEERING, KARIMNAGAR, TELANGANA.**

**ABSTRACT:** With the widespread usage of smartphones, malicious apps have grown in number. Because Android is so widely used and well-known, dangerous malware routinely targets it. This paper provides an overview of a method for identifying and thwarting malicious software that targets Android applications and devices. To capture numerous features of Android apps, the framework uses a large range of feature classes. Similarity-based or existence-based feature extraction approaches are used to more accurately represent these attributes for malware detection and prevention. This study presents a novel multimodal deep learning methodology that might serve as the foundation for Android malware detection and avoidance. You may make the most of a range of features by using this detection model. Deep neural network model investigations are used to ascertain the model's accuracy. Additionally, the framework is assessed in a variety of methods, including the assessment of model performance over time, the analysis of the value added by different characteristics, and the assessment of the representation of the features in the framework.

**Keywords :** *Machine learning Android malware, intrusion detection , malware detection, , neural network.*

## I. INTRODUCTION

Attacks targeting mobile devices, such as smartphones and tablets, have increased dramatically in recent years. Malware designed to infiltrate mobile devices can cause serious security holes and financial losses, making it a major threat to information technology. G DATA conducted a study in 2017 and found that in the first three months of that year, security researchers uncovered about 750,000 separate cases of Android malware [9]. The creation and dissemination of mobile malware will likely continue in order to commit various cybercrimes

aimed at mobile devices. Android is the most popular mobile OS, and as a result, it is the primary target of mobile malware. An second factor encourages Android malware producers to create more malicious apps: the rising popularity of Android devices.

The ability of the Android OS to allow users to install programs from external sources is the primary cause of this phenomena since it allows attackers to trick Android users into downloading malicious software from their servers.

There have been several studies conducted to find ways to protect Android devices against malware.

Malware detection studies can be roughly divided into two broad categories: those that rely on dynamic analysis [1-8] and those that rely on static analysis [10-14]. Dynamic analysis techniques require close attention to semantic details throughout the execution of any program in a safe setting. Static analysis techniques, on the other hand, can be used to glean syntactic properties without actually running the programs themselves. Static analysis is advantageous because it may be performed without establishing computational overheads or execution environments, unlike other types of analysis. Dynamic analysis provides the benefit of dealing with rogue apps that use obfuscation methods like code packing or encryption.

Numerous attempts have been made in the past to identify Android malware, but they have mostly relied on a small set of features. There is a fixed number of properties that can be expressed by each feature type in an application. In contrast, we present a theoretical framework for doing so that makes use of many data points. These variables were chosen to provide a multidimensional representation of the application's attributes. The proposed framework will first concentrate on extracting and processing the various feature kinds. Then, we'll use vector feature generating methods to refine these characteristics into something more accurate. While malware and safe software do share some characteristics, the efficiency of feature vector development based on similarity or existence in distinguishing between the two is particularly striking. The framework also uses a classification model to define the importance of each component's level of

classification. For a comprehensive set of features and helpful approaches, the deep learning method has been found to be an effective classification algorithm.

## **II. PROPOSED SYSTEM**

This research suggests using a multimodal deep neural network model to verify attributes with a wide range of properties. To improve the quality of the network's output by integrating more variables, it was built utilizing a multimodal deep learning approach. Using both voice and lip shape data, a multimodal deep learning strategy was used in a prior study [15] to recognize human speech. This method makes use of a wide variety of feature types, each of which is fed into its own, independently operating neural network. Each preliminary neural network is connected to the final neural network in order to produce classification results. According to the results of an investigation, multimodal deep learning is an innovative method for spotting Android malware. Malgenome project [16] and Virus Share [17] are two well-known examples of small datasets that have inspired a plethora of studies that use their architecture and huge datasets. By utilizing the deep neural network model, its efficacy may be analyzed and assessed. The choice of feature representation is based on the available features, and the performance of the framework is evaluated over model changes using numerous evaluation methods. This framework outperforms that of competing deep learning approaches when it comes to identifying malicious software.

## **III. SOFTWARE USED**

The machine learning framework in Python can be used to detect and prevent malware attacks on Android devices. Artificial intelligence (AI) using Python for machine learning allows computers to learn without being explicitly taught. The term "machine learning" refers to the method used to create adaptive computer programs.

#### **ANACONDA NAVIGATOR**

The anaconda navigator uses Python, a programming language that is seeing rapid adoption in open data science and is used by many other new platforms in the industry.

Python and R are both part of the Anaconda distribution, which is a free and open-source software package. Machine learning, data science, predictive analytics, and massive data processing are just some of the scientific computing tasks that benefit from its design. Deployment and package management are simplified as a result. To efficiently manage different versions of packages, Conda uses a package management system [19]. Approximately 1400 of the most popular data-science tools are included in the Anaconda distribution, which is used by more than 13 million individuals. All three major platforms (Linux, Windows, and Mac OS) are supported by these bundles [20].

Jupyter, or the Scientific Python Development Environment, is a widely used software found within the Anaconda navigator. Python's prominent libraries, such as SciPy for signal and image processing, NumPy for linear algebra, and matplotlib for interactive 2D/3D graphing, contribute to the language's versatility and robustness as a programming environment. The

language has extensive capabilities, such as a numerical computing environment and interactive testing, editing, introspection, and debugging.

#### **IV. RELATED WORK**

All prior approaches that made use of deep learning algorithms can be characterized by the concept of turns. Windows virus is detected using recurrent neural networks in a study by Razvan Pascanu [21]. The API events were utilized for the detection features implementation. Deep Sign's solution for detecting malware on Windows utilized dynamic API calls and their associated arguments [22]. In this scenario, Deep Sign uses the deep network to determine which files are safe to open and which are dangerous. Joshua Saxe [23] proposes using deep neural networks to detect viruses. This tactic's features included strings, randomness, PE import methods, and Windows binary data. Malware like Trojans may be hard to spot using this technique since they share so many characteristics with common software.

Droid Detector [24] utilized ML methods to spot malware on Android devices. During this process, many different features are retrieved, and the deep network uses these features to make decisions. Android malware can be detected with Wei Yu's [25] method by modeling a neural network with permissions and system calls. The system solely makes use of static features, even if a vast amount of data is available for analysis and detection. The convolution neural network (CNN) is a key

component of Niall McLaughlin's [26] suggested method for Android virus detection.

The system makes advantage of the unprocessed opcode sequences of programs as features. In [27], Hossein Fereidooni introduced ANASTASIA, a tool for identifying malicious Android apps based on their permissions, intents, API calls, and system instructions. A deep neural network is used in this method together with other classifiers. While there is a wealth of new data can be mined from Android apps, the vast majority of existing techniques only make use of a small subset of these data points. In addition, the cases where the older methods augmented the expanded feature sets are ignored.

## **V CONCLUSION**

In this work, we present a literature evaluation of the smart eye blink solution for motor neurone disease (MND) patients, who will benefit from a technique that is both more precise and more responsive than previous methods.

## **REFERENCES**

1] D. Luke, V. Notani, and A. Lakhotia, "Droidlegacy: Automated familial classification of android malware," In Proc. of the ACM SIGPLAN on Program Protection and Reverse Engineering Workshop, pp. 3, 2017.

2] A. Zarni, and W. Zaw, "Permission-based android malware detection," International Journal of Scientific and Technology Research, vol. 2, no. 3, pp. 228-234, 2017.

3] Ch.-Y. Huang, Y.-T. Tsai, and C-H. Hsu, "Performance evaluation on permission-based detection for android malware," Advances in

Intelligent Systems and Applications, vol. 2, pp. 111-120, 2016.

4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," In Proc. the Network and Distributed System Security Symposium (NDSS), vol. 14, pp. 23-26, 2016.

5] D-J. Wu, C-H. Mao, T-E. Wei, H-M. Lee, K-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," In Proc. of the Asia Joint Conference on Information Security (Asia JCIS), pp. 62-69, 2017.

6] S. S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps," In Proc. of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), pp. 204-217, 2017.

7] M. E. Chin, A. P. Felt, K. Greenwood, D. Wagner, "Analyzing inter-application communication in Android," In Proc. of the international conference on Mobile systems, applications, and services, pp.239-252, 2017.

8] P. PF Chan, L. CK Hui, SM. Yiu, "Droidchecker: analyzing android applications for capability leak," In Proc. of the ACM conference on Security and Privacy in Wireless and Mobile Networks, pp. 125-136, 2015.

9] A G DATA Report, "8,400 new android malware samples every day".

10] W. Enck., P. Gilbert, S. Han, V. Tendulkar, B-G. Chun, L. P. Cox, J. Jung, P. Macdaniel, A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on

smartphones,” ACM Transaction on Computer Systems, vol. 32, no. 5, 2017.

11] L. K. Yan, H. Yin, “DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis,” In Proc. of the USENIX Security Symposium, pp. 569-584, 2016.

12] T. Bläsing, L. Batyuk, A-D. Schmidt, S. A. Camtepe, S. Albayrak, “An android application sandbox system for suspicious software detection,” In Proc. of the Malicious and unwanted software (MALWARE), pp. 55-62, 2017.

13] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss, “Andromaly: a behavioral malware detection framework for Android devices,” Journal of Intelligent Information Systems, vol. 38, no. 1, pp. 161-190, 2016.

14] A- D.Schmidt, F. Peters, F. Lamour, C. Scheel, S. A. Camtepe, S. Albayrak, “Monitoring smartphones for anomaly detection,” Mobile Network sand Applications, vol. 14, no. 1, pp. 92-106, 2017.

15] R. Pascanu, J. W. Stroke, H. Sanossian, M. Marinescu, A. Thomas, “Malware classification with recurrent networks,” In Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1916-1920, 2015.

16] O. E. David, N. S. Netanyahu, “Deepsign: Deep learning for automatic malware signature generation and classification,” In Proc. of the International Joint Conference on Neural Networks (IJCNN), pp. 1-8, 2015.

17] J. Saxe, K. Berlin, “Deep neural network based malware detection using two dimensional

binary program features,” In Proc. of the 10th International Conference on Malicious and Unwanted Software (MALWARE), pp. 11-20, 2015.

18] Z. Yuan, Y. Lu, Y. Xue, “Droiddetector: android malware characterization and detection using deep learning,” Tsinghua Science and Technology, vol. 21, no. 1, pp. 114-123, 2016.

19] W. Yu, L. Ge, G. Xu, X. Fu, “Towards Neural Network Based Malware Detection on Android Mobile Devices,” Cybersecurity Systems for Human Cognition Augmentation, pp. 99-117, 2018.

20] N. Mchaughlin, J. Martinez del Rincon, B-J. Kang, S. Yerima, Y. Safaei, E. Trickel, Z. Zhao, A. Doupe, G. Joon Ahn, “Deep Android Malware Detection,” In Proc of the ACM on Conference on Data and Application Security and Privacy (CODASPY), pp. 301-308, 2017.